An In-depth Study of Activation Functions

SEOYOUNG PARK, Dept. of Applied Artificial Intelligence Sungkyunkwan University, Republic of Korea
GAEUN SEO, Dept. of Applied Artificial Intelligence Sungkyunkwan University, Republic of Korea
MINWOO AHN, Dept. of Data Science Sungkyunkwan University, Republic of Korea
JEONGYEON KIM, Dept. of Applied Artificial Intelligence Sungkyunkwan University, Republic of Korea

Deep learning, grounded in artificial neural networks, heavily relies on activation functions to learn and predict complex data patterns. Activation functions introduce nonlinearity to neural networks, essential for capturing intricate relationships within the data. The widely used Rectified Linear Unit (ReLU) [21] activation function, despite its popularity, has limitations such as the dead ReLU problem, raising questions about its universal applicability. This study investigates whether alternative activation functions or adaptive activation functions could outperform ReLU in specific tasks. We conducted experiments on various neural network architectures, including CNN, ResNet50 [13], and BERT [11], comparing different activation functions' performance. Our findings highlight the necessity of task-specific activation function selection, revealing that no single function is universally optimal. While adaptive activation functions show promise, their benefits are more pronounced in complex models. This report underscores the critical role of activation functions in deep learning and suggests potential directions for future research, including the development of novel activation functions. Code is available at: https://github.com/gaeun0112/aisystem_project

ACM Reference Format:
Seoyoung Park, Gaeun Seo, Minwoo Ahn, and Jeongyeon Kim. 2024. An In-depth Study of Activation Functions. 1, 1 (June 2024), 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Deep learning is a technology based on artificial neural networks that mimics the structure and function of the human brain to learn from and predict data. One of the key components of these neural networks is the activation function. The activation function is a nonlinear function that determines the output of each neuron, significantly impacting the performance and learning capability of deep learning models.

Activation functions are necessary because neural networks cannot learn complex patterns from input data through simple linear transformations. A model using only linear transformations, even with a multi-layer structure, would possess the same expressive power as a simple linear model, failing to capture nonlinear relationships. Therefore, activation functions introduce nonlinearity to the input values, enabling the model to learn a wide range of complex patterns.

Currently, the most widely used activation function in deep learning models is the Rectified Linear Unit (ReLU). The ReLU function performs a linear transformation for positive inputs and sets the output to zero for negative inputs,

Authors' Contact Information: Seoyoung Park, Dept. of Applied Artificial Intelligence and Sungkyunkwan University, Seoul, Republic of Korea, seoyoungp@skku.edu; Gaeun Seo, Dept. of Applied Artificial Intelligence and Sungkyunkwan University, Seoul, Republic of Korea, gaeun0112@g.skku.edu; Minwoo Ahn, Dept. of Data Science and Sungkyunkwan University, Seoul, Republic of Korea, mwahn99@skku.edu; Jeongyeon Kim, Dept. of Applied Artificial Intelligence and Sungkyunkwan University, Seoul, Republic of Korea, wjddus0203@g.skku.edu.

Manuscript submitted to ACM

helping to mitigate the vanishing gradient problem and accelerating the learning process. However, ReLU also has its drawbacks, such as the dead ReLU problem, and it does not guarantee optimal performance in all scenarios.

We questioned whether the widespread use of the ReLU function is indeed the best choice. While ReLU generally shows good performance, other activation functions might yield better results for specific tasks or datasets. To test this hypothesis, we conducted experiments by performing the same tasks with various activation functions and comparing their performance.

Additionally, we focused on adaptive activation functions. Adaptive activation functions learn and apply the optimal activation function based on the data during the training process, potentially enhancing model performance. We aimed to determine whether adaptive activation functions could outperform fixed activation functions for specific tasks.

This report, based on our experimental results, aims to shed light on the importance of activation function selection in deep learning models. We discuss how activation functions have been proposed, why certain functions are predominantly used in the current AI industry, and, if further development of activation functions is needed, which aspects should be explored. Through this exploration, we seek to provide a comprehensive and in-depth understanding of activation functions.

2 Related Work & Background

2.1 Related Works

Studies comparing the performance of the activation functions(AFs) continued. Chigozie et al. [3] analyzed the winning architectures of ImageNet competition and other remarkable architectures and showed that the ReLU (in hidden layers) and the Softmax (in output layer) activation functions are the dominant AFs used in practical deep learning applications. However, variants of the ReLU, including LReLU, PReLU and RReLU performs better than the ReLU but some of these functions lack theoretical justifications to support their state-of-the-art results. Also, the ELU's has been highlighted as a faster learning AF compared to their ReLU counterpart, and this assertion has been validated by Pedamonti [16], after an extensive comparison of some variants of the ELU and ReLU AF on the MNIST recognition dataset. The newer activation functions seem to outperform the older AFs like the ReLU, yet even the latest deep learning architectures rely on the ReLU function. Current practices does not use the newly developed state-of-the-art AFs but depends on the tested and proven AFs, thereby underlining the fact that the newer activation functions are rarely used in practice [3].

Steffen et al. [19] observed the change in performance according to the activation function in three NLP tasks (Sentence Classification, Document Classification, and Sequence Tagging). MLP, CNN, and RNN were used as learning models for each task, and 21 activation functions were noted. As a result of the experiment, it was found that the penalized tan, which was relatively not famous, was the most stable in several tasks, and that the function could successfully replace the activation functions inside the LSTM cell.

Akash et al. [1] conducts a comparative research of various activation functions, which play a significant role in determining the performance of deep learning models, focusing on their characteristics and performance. It discusses the role of activation functions, the characteristics of representative functions, and presents experimental results. In experiments using the CIFAR-10 dataset, ReLU and Leaky ReLU achieved high accuracy, while Swish showed better performance in certain scenarios despite its higher computational cost. Consequently, the choice of activation function should vary depending on the neural network architecture, dataset, and problem characteristics. Since no specific function guarantees optimal performance in all situations, the paper emphasizes the importance of carefully selecting the appropriate function for each project.

Adaptive activation functions are activation functions that allow the network to dynamically adapt to the data. While these include parameterized activation functions like PReLU and PELU, they share one parameter set for the entire layer or network. In 2015, Agostinelli et al. [6] proposed a novel form of piecewise linear activation function where each neuron's activation function is learned independently. This approach demonstrated better performance compared to models using fixed ReLU functions, achieving performance improvements of 7.51% on CIFAR-10 and 30.83% on CIFAR-100. The study suggests introducing adaptive activation functions, which operate as learning parameters rather than fixed hyperparameters for each neuron, to enhance the performance of deep learning models

The research on nonlinear activation functions has evolved over time. Initially, it was common to use the sigmoid as the nonlinear activation function for constructing neural networks. Subsequently, Hornik et al.(1989) [10] and Cho & Saul (2010) [4] demonstrated that using sigmoid functions in sufficiently large neural networks can approximate arbitrary complex functions. Secondly, the paradigm of using activation functions shifted with the introduction of ReLU (rectified linear unit), an unsaturated rectified linear function used as an activation function. Jarrett et al. (2009) [12] and Glorot et al. (2011) [7] emphasized the advantages of ReLU over traditional sigmoid, recommending its use to alleviate the vanishing gradient problem and speed up the training of deep neural networks. Following ReLU, various activation functions were explored to complement its limitations. Despite proposals like Leaky ReLU, most did not show significant performance improvements. Due to the difficulty in generalizing the performance of activation functions based on model structure or dataset characteristics, newly proposed activation functions did not replace ReLU as sigmoid was replaced. Currently, the third era of activation functions considers the use of adaptive activation functions. This stems from the exploration of alternatives to fixed activation functions. Previous efforts include genetic and evolutionary algorithms (Yao) [22] attempting to select activation functions for each neuron from predefined sets, along with Turner & Miller's strategy [20] of combining this with a single scaling learning parameter during training. Additionally, Agostinelli et al. [6] introduced strategies where each layer learns its activation function independently.

2.2 Background

2.2.1 Activation Functions.

Sigmoid function is often used in output layer for predicting probability. It has been applied successfully in binary classification problems. However, it has several drawbacks such as gradient vanishing, slow convergence, and unstable training.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Unlike Sigmoid, which has a range from 0 to 1, tanh(hyperbolic tangent) has a range from -1 to 1, so the average of the data is closer to 0, making it more efficient than sigmoid. However, like sigmoid, there is a limitation in that when the input value is very large or small, the gradient approaches 0 so gradient vanishing occurs.

$$tanh(x) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

ReLU [21] has been the most widely used AF for deep learning applications with state-of-the-art results to date. It shows better performance compared to the Sigmoid. Because of simple computation, the advantage of ReLU is fast learning. However, ReLU easily overfits compared to the Sigmoid. Also, it is sometimes fragile during training thereby causing some of the gradients to die.

$$ReLU(x) = max(0, x)$$

To resolve this issue, the Leaky ReLU [2] was proposed.

$$LReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}$$

In contrast to ReLU, ELU [5] have negative values which allows them to push mean unit activations closer to zero like batch normalization but with lower computational complexity. Mean shifts toward zero decrease the gap between the normal gradient and the unit natural gradient and, thereby speed up learning. In contrast to LReLU and PReLU, ELU have a clear saturation plateau in its negative regime, allowing them to learn a more robust and stable representation.

$$ELU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}$$

SeLU [8] is the activation function of Self-normalizing Neural Networks(SNNs), which induce self-normalizing properties. While batch normalization requires explicit normalization, neuron activations of SNNs automatically converge towards zero mean and unit variance.

$$SeLU(x) = \tau \begin{cases} x, & \text{if } x > 0 \\ \alpha e^x - \alpha, & \text{if } x \leq 0 \end{cases}$$

GELU(Gaussian Error Linear Units) is a high-performing neural network activation function that nonlinearity wights inputs by their value, rather than gates inputs by their sighn as in ReLUs. It perform better than ReLU and ELU across various tasks.

$$GELU\ (x) = xP\ (X \leq x) = x\Phi\ (x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right],$$

2.2.2 *Models.*

CNN(Convolutional Neural Networks) [17] is used to extract features from images (and videos), employing convolutions as their primary operator.

ResNet50 [13] is a deep neural network consisting of 50 layers of Residual Network. A key feature of ResNet50 is the skip connection, also known as shortcut connection, which enables the network to address the vanishing gradient problem and achieve higher accuracy even as it becomes deeper.

BERT [11] is a model using the encoder structure of Transformer, it is possible to understand the meaning of a sentence because it was pre-trained using the masked language model method and the next sentence prediction method with bidirectional characteristics. It is mainly fine-tuned and used to fit several tasks of the pre-trained model learned with a lot of data.

3 Experimental Comparison of Activation Functions

3.1 CNN + MNIST & CIFAR-10

MNIST(Modified National Institute of Standards and Technology) [15] is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples. They are images of 28*28 grayscale pixels. CIFAR-10(Canadian Institute for Advanced Research, 10 classes) [14] is a tiny image dataset, which consists of 32*32 color images. It has 6000 images per class with 5000 training and 1000 testing images per class. 10 classes consist of airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck).

The model used in this experiment is from the structure of pytorch cifar10 tutorial [17]. Only the channel dimension was adjusted according to the input data. For the experimental setting, we used two optimizers, Adam (lr=0.001) and SGD (lr=0.001, momentum=0.9). The epoch was set to 50, and the early stopping was added(patience=5 based on validation loss). Also, batch size was set to 64. The weight of the model was initialized to xavier uniform. The experiment was run on Google's colaboratory (T4 GPU) using the PyTorch framework. All parameter values used default values.

3.2 ResNet50 + MNIST & CIFAR-10

The datasets and experimental settings are identical to those stated for the CNN experiment. In this experiment, we used the existing ResNet50 model while adding additional configurations, such as adjusting the number of input image channels and output classes based on the given dataset.

3.3 BERT + SST2 & CoLA

Two datasets were used in this experiment. The first data, CoLA, is a dataset of 23 linguistic publications and consists of 8,551 training data. Second, SST2 is the movie comment data, and each comment's degree of positivity is scored, and it is composed of 6,7349 training data. Because both SST2 and CoLA don't have labels of test data, we randomly sampled 1,000 samples from train set in order to make test data. By experimenting with two datasets with large differences in the amount of data in the same environment, the effect of the activation function on performance was also analyzed according to the amount of data.

We experimented by changing the activation function of BERT's feed forward network block from gelu to several activation functions, and used bert-base-uncased weight. We used the basic activation functions relu, tanh, sigmoid, and swish, and the variants of relu, elu, leaky relu, and squared relu. In addition, we used penalized tanh, which had the most stable performance in Steffen et al., and glu and swig, which are known to have good performance in Transformer series models. In the case of CoLA, epoch 3, batch size 16 and learning rate 5e-5 were applied, and AdamW was used as the optimizer. In the case of SST2, only the epoch was set to 1 in the CoLA setting. The experiment was conducted on NVIDIA L4 among Colab additional computing resources, and we used wandb for visualization and pytorch and transformers for model learning in the experiment.

4 Results and Discussions

4.1 CNN + MNIST & CIFAR-10

4.1.1 Results.

The results of experiment 3.1 are shown in Table 1, 2, 3, 4, 5, 6, 7, 8, and Figure 1, 2, 3, 4.

	Accuracy	Loss	F1 Score
Sigmoid	94.22	0.207	0.942
ReLU	98.35	0.054	0.983
LReLU	98.26	0.057	0.982
ELU	98.78	0.042	0.988
SeLU	98.81	0.038	0.988

Table 1. Test accuracy, loss, f1-score of experiment 3.1 (MNIST + SGD)

	Accuracy	Loss	F1 Score
Sigmoid	28.45	1.945	0.236
ReLU	62.30	1.137	0.624
LReLU	62.25	1.114	0.619
ELU	65.42	1.023	0.654
SeLU	62.98	1.083	0.629

Table 2. Test accuracy, loss, f1-score of experiment 3.1 (CIFAR-10 + SGD)

	Accuracy	Loss	F1 Score
Sigmoid	98.84	0.038	0.988
ReLU	98.49	0.052	0.985
LReLU	98.50	0.051	0.985
ELU	98.54	0.058	0.985
SeLU	98.74	0.055	0.987

Table 3. Test accuracy, loss, f1-score of experiment 3.1 (MNIST + Adam)

	Accuracy	Loss	F1 Score
Sigmoid	61.57	1.114	0.607
ReLU	61.51	1.193	0.611
LReLU	62.88	1.132	0.628
ELU	63.70	1.148	0.638
SeLU	61.83	1.142	0.617

Table 4. Test accuracy, loss, f1-score of experiment 3.1 (CIFAR-10 + Adam)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	50	27	27	41	25
Total Time	727.057	393.458	393.710	596.763	364.672
Average Time	14.541	14.573	14.582	14.555	14.587

Table 5. Training time(sec) of experiment 3.1 (MNIST + SGD)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	50	42	38	34	35
Total Time	715.625	603.538	545.274	487.569	503.182
Average Time	14.312	14.370	14.349	14.340	14.377

Table 6. Training time(sec) of experiment 3.1 (CIFAR-10 + SGD)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	21	10	8	10	12
Total Time	272.171	126.832	101.477	127.259	151.900
Average Time	12.961	12.683	12.685	12.726	12.658

Table 7. Training time(sec) of experiment 3.1 (MNIST + Adam)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	40	14	12	13	12
Total Time	495.139	174.472	148.793	158.959	151.252
Average Time	12.378	12.462	12.399	12.228	12.604

Table 8. Training time(sec) of experiment 3.1 (CIFAR-10 + Adam)

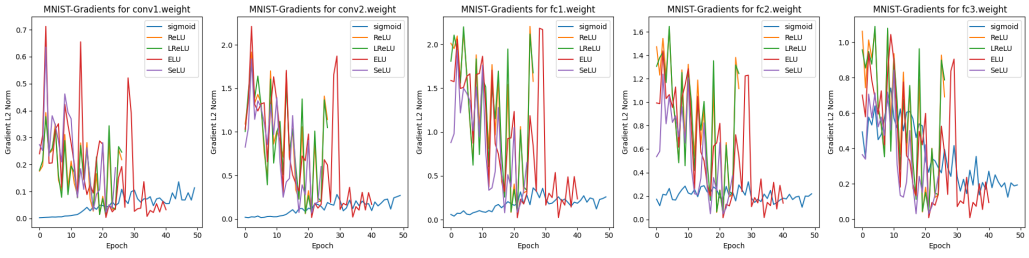

Fig. 1. L2 norm of Gradients per layer in experiment 3.1 (MNIST + SGD)

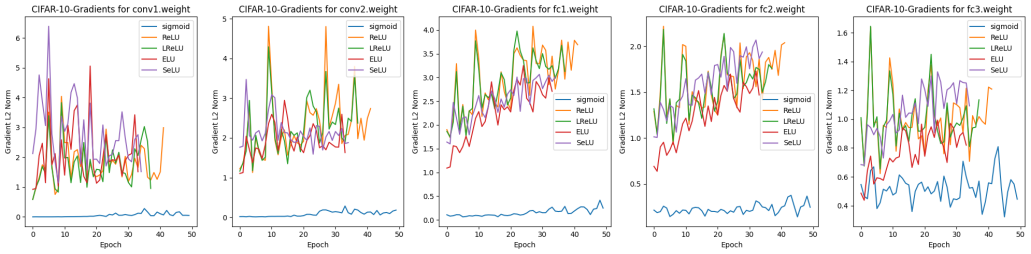

Fig. 2. L2 norm of Gradients per layer in experiment 3.1 (CIFAR-10 + SGD)

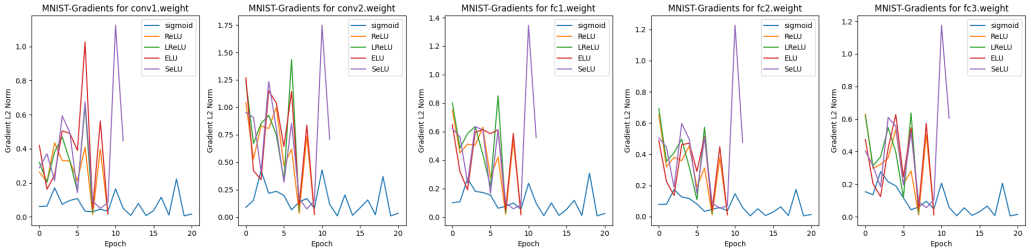

Fig. 3. L2 norm of Gradients per layer in experiment 3.1 (MNIST + Adam)

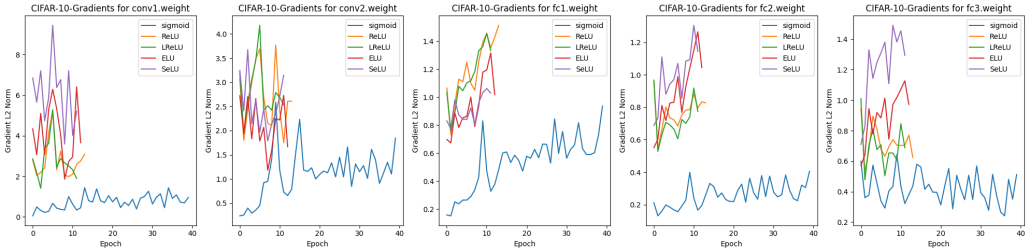

Fig. 4. L2 norm of Gradients per layer in experiment 3.1 (CIFAR-10 + Adam)

4.1.2 Discussions.

As shown in Figure 1, 2, 3, and 4, gradient values of the Sigmoid is close to 0 compared to other activation functions. This proves that the Sigmoid is vulnerable to gradient vanishing. Also, looking at the epoch, the Sigmoid shows the largest value in all experiments. This shows the slow convergence of the sigmoid. In the results of ReLU, it can be seen that it easily overfits compare to the Sigmoid. In all experiments with Adam, the test accuracy of ReLU showed the lowest results. In the case of experiments using SGD, it is judged that the Sigmoid experiment was underfitted because training was completed before early stopping, making comparison impossible. When comparing Test Loss, in the experiment with Adam, Sigmoid's loss was the lowest, which is presumed to be because the weight initialization best fits Sigmoid. Nevertheless, in the experiment with SGD, SeLU and ELU showed the lowest loss values. Test accuracy shows that the results of ELU are the best value in all experiments that learned CIFAR-10.

4.2 ResNet50 + MNIST & CIFAR-10

4.2.1 Results.

The results of experiment 3.2 are shown in Table 9, 10, 11, 12, 13, 14, 15, 16, and Figure 5, 6, 7, 8.

	Accuracy	Loss	F1 Score
Sigmoid	98.43	0.062	0.984
ReLU	98.64	0.051	0.986
LReLU	98.77	0.047	0.987
ELU	98.75	0.054	0.987
SeLU	98.71	0.054	0.986

Table 9. Test accuracy, loss, f1-score of experiment 3.2 (MNIST + SGD)

	Accuracy	Loss	F1 Score
Sigmoid	58.74	1.625	0.584
ReLU	59.35	1.560	0.592
LReLU	58.02	1.620	0.577
ELU	58.72	1.525	0.586
SeLU	58.51	1.574	0.584

Table 10. Test accuracy, loss, f1-score of experiment 3.2 (CIFAR-10 + SGD)

	Accuracy	Loss	F1 Score
Sigmoid	50.73	1.831	0.486
ReLU	10.60	3.775	0.024
LReLU	98.45	0.049	0.984
ELU	99.04	0.039	0.990
SeLU	98.68	0.042	0.986

Table 11. Test accuracy, loss, f1-score of experiment 3.2 (MNIST + Adam)

	Accuracy	Loss	F1 Score
Sigmoid	78.75	0.796	0.787
ReLU	78.07	0.832	0.779
LReLU	79.00	0.709	0.789
ELU	76.20	0.971	0.762
SeLU	78.68	0.878	0.786

Table 12. Test accuracy, loss, f1-score of experiment 3.2 (CIFAR-10 + Adam)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	11	15	18	24	15
Total Time	473.077	646.402	775.221	1033.866	645.946
Average Time	43.007	43.093	43.067	43.077	43.063

Table 13. Training time(sec) of experiment 3.2 (MNIST + SGD)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	13	12	12	12	12
Total Time	739.400	682.195	682.265	682.163	682.137
Average Time	56.876	56.849	56.855	56.846	56.844

Table 14. Training time(sec) of experiment 3.2 (CIFAR-10 + SGD)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	12	10	14	41	26
Total Time	549.269	457.417	639.980	1874.073	1188.108
Average Time	45.772	45.741	45.712	45.709	45.696

Table 15. Training time(sec) of experiment 3.2 (MNIST + Adam)

	Sigmoid	ReLU	LReLU	ELU	SeLU
Epoch	15	15	12	15	15
Total Time	883.641	883.913	707.658	883.448	884.382
Average Time	58.909	58.927	58.971	58.896	58.958

Table 16. Training time(sec) of experiment 3.2 (CIFAR-10 + Adam)

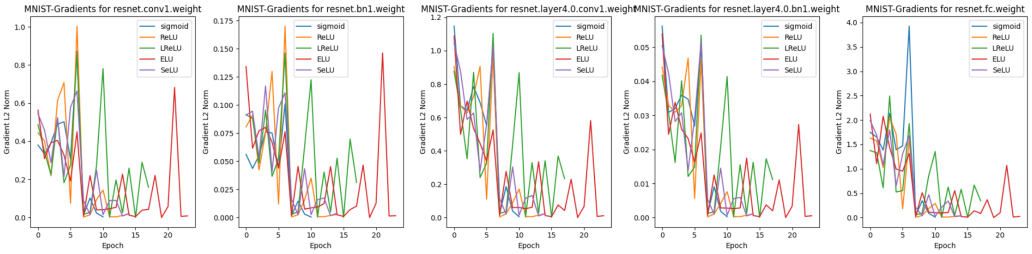

Fig. 5. L2 norm of Gradients per layer in experiment 3.2 (MNIST + SGD)

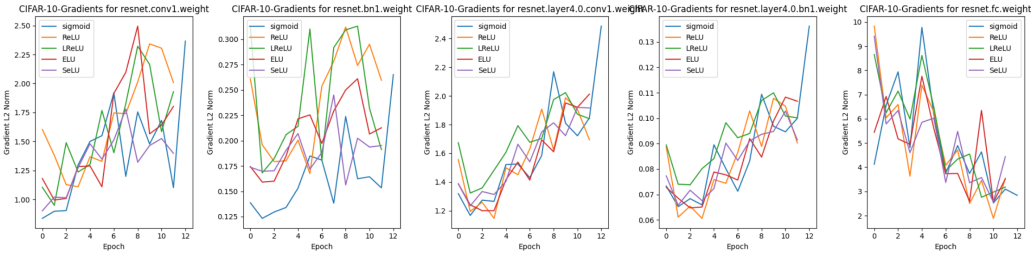

Fig. 6. L2 norm of Gradients per layer in experiment 3.2 (CIFAR-10 + SGD)

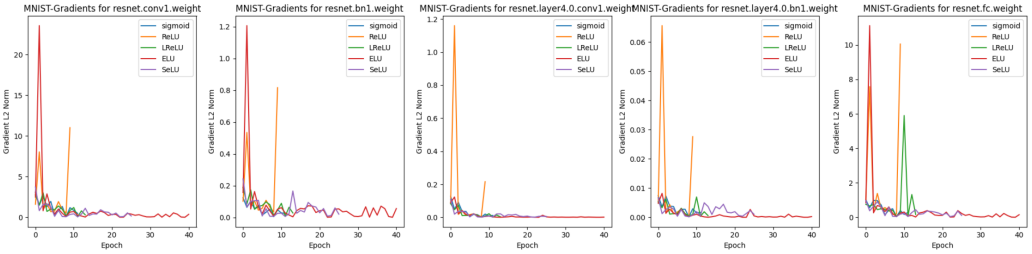

Fig. 7. L2 norm of Gradients per layer in experiment 3.2 (MNIST + Adam)

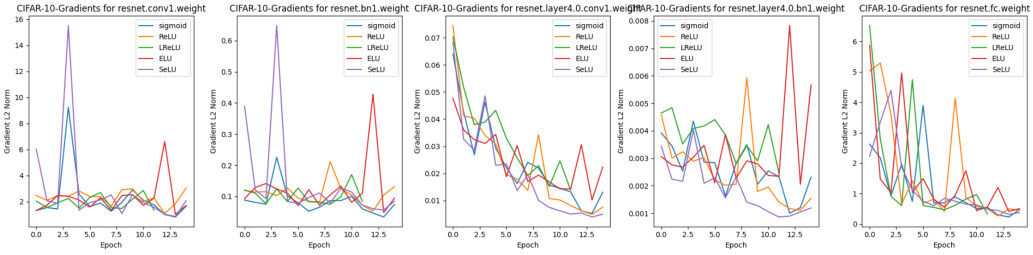

Fig. 8. L2 norm of Gradients per layer in experiment 3.2 (CIFAR-10 + Adam)

4.2.2 *Discussions.*

When looking at the Figure about gradients 7, it can be observed that when using Adam on the MNIST dataset, the issue of gradient vanishing occurs across various activation functions. Upon analysis, Adam utilizes adaptive learning rates for each parameter to enhance the efficiency of gradient descent. However, as training progresses, it may fail to maintain appropriate learning rates for all parameters. This problem becomes more prominent, especially in deep and complex networks like ResNet50, potentially exacerbating the issue of gradient vanishing.

When examining the epoch section 13, 14, 15, 16, it is evident that Sigmoid and ReLU converge quickly on MNIST, while ELU converges very slowly. This implies that Sigmoid and ReLU perform better on simpler datasets like MNIST. ELU, although relatively slow to converge, may not suffer from significant gradient vanishing issues on simpler datasets. On CIFAR-10, all activation functions converge at a similar pace, with ELU and SeLU particularly converging rapidly. This indicates that these activation functions effectively address the problem of gradient vanishing and can deliver better performance on complex datasets.

When it comes to the results of test loss and test accuracy 9, 10, 11, 12, particularly when using Adam, ELU exhibited the highest performance on MNIST (acc: 99.04, loss: 0.039). When SGD was used, LReLU showed the best performance on MNIST, with both the lowest loss and highest accuracy. Conversely, on CIFAR-10, LReLU showed the lowest performance. LReLU, due to its ability to have a small gradient for negative inputs, can capture nonlinear features better, especially when the data has such characteristics. Therefore, LReLU may perform well on simpler datasets like MNIST. However, there were minor differences in the loss and accuracy of activation functions, and when Adam was used, LReLU showed marginally the best performance. Thus, it can be concluded that the combination of optimizer and activation function ultimately affects overall performance.

4.3 BERT + SST2 & CoLA

4.3.1 *Results.*

Activation Function	ReLU	GELU	Tanh	Sigmoid	ELU	LReLU
Cola Sum Time	834.501	852.309	850.663	850.752	850.756	850.666
Cola Mean Time	278.167	284.103	283.554	283.584	283.585	283.555
SST Time	2462	2521.148	2491.348	2491.151	2487.66	2490.723

Table 17. Total learning time & Average learning time of each activation function (Part 1)

Activation Function	Swish	ReLU_2	Penalized Tanh	GLU	SwiGLU
Cola Sum Time	851.106	948.305	968.713	753.114	785.672
Cola Mean Time	283.702	316.102	322.904	251.038	261.891
SST Time	2490.962	2791.172	2856.536		2323.362

Table 18. Total learning time & Average learning time of each activation function (Part 2)

Activation Function	ReLU	GELU	Tanh	Sigmoid	ELU	LReLU
Cola Test Accuracy	0.712	0.806	0.708	0.708	0.708	0.683
Cola Test F1	0.7192	0.7938	0.587	0.587	0.587	0.6345
SST Test Accuracy	0.938	0.945	0.558	0.558	0.558	0.931
SST Test F1	0.9381	0.9449	0.3997	0.3997	0.3997	0.931

Table 19. Test Accuracy & F1 Score of each activation function (Part 1)

Activation Function	Swish	ReLU_2	Penalized Tanh	GLU	SwiGLU
Cola Test Accuracy	0.708	0.708	0.708	0.708	0.708
Cola Test F1	0.587	0.587	0.587	0.587	0.587
SST Test Accuracy	0.558	0.558	0.558	0.558	0.558
SST Test F1	0.3997	0.3997	0.3997	0.3997	0.3997

Table 20. Test Accuracy & F1 Score of each activation function (Part 2)

In terms of time(Table 17, 18), consistent with conventional wisdom, ReLU was the fastest in both data sets, except for GLU and swiglu, which had to reduce the dimension of the output layer due to the nature of the activation function. The activation function that took the longest for both data was penalized tanh. When a penalty is given when x is less than 0, the reformation of the graph is suppressed compared to before, causing the gradient loss problem to become more serious, so the execution time of the function was the longest.

In terms of accuracy and f1 score(Table 19, 20) GELU performed best on all data. Because the pretrained weight of the model was imported, it can be analyzed that the performance of GELU, the previously set activation function, is the highest. Additionally, except for ReLU, leaky ReLU, and GELU, all have the same accuracy and f1 score, confirming that those cases were not learned. This will be discussed in detail in the [gradient vanishing] section.

4.3.2 Discussions.

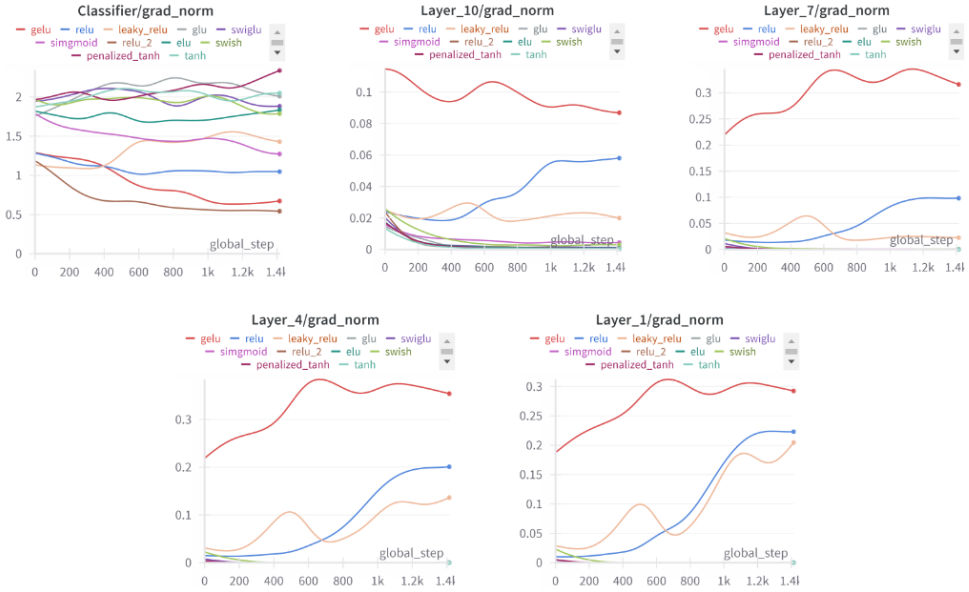

Fig. 9. L2 gradient norm for each major layer 3.3

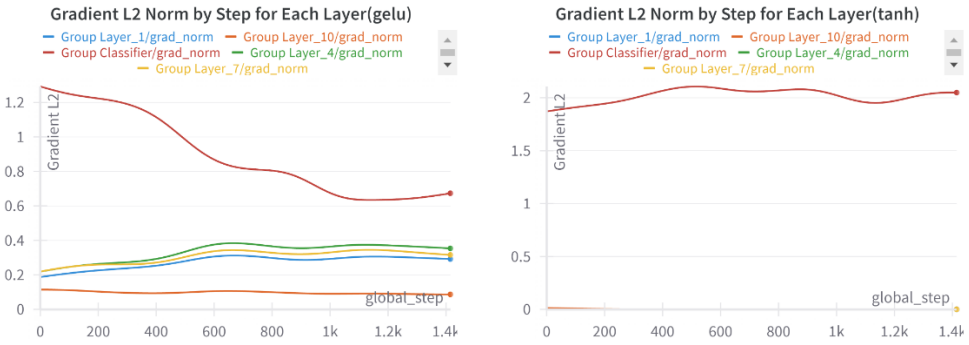

Fig. 10. GELU's gradient L2 norm by layer vs tanh's gradient L2 norm by layer 3.3

[Gradient Vanishing Problem] The reason activation functions other than ReLU, leaky ReLU, and GELU did not learn well is because of gradient vanishing. Looking at Figure 9, you can see the l2 norm values of the gradient values of some layers of Bert. Here, in the classifier layer, the gradient norms of GELU, ReLU, and leaky ReLU are lower than those of other activation functions, but even after layer 10, you can see that only the gradient norm values of ReLU, leaky ReLU, and GELU are higher than those of other functions. , even if you go to layer 1, you can see that the gap widens further. Since the gradient becomes extremely small from the output layer to the input layer, it can be seen that functions that have not been learned are experiencing gradient vanishing problems.

Even when we compare the gradient L2 Norm for each layer of GELU, which has undergone the most stable learning, and tanh, which has not learned the least, we can see that the gradient vanishing problem occurs (Figure 10). In gelu, the difference in gradient L2 Norm values for each layer is large. On the other hand, the gradient L2 Norm of tanh is close to 0 except for the classifier layer. Considering that the functions in which gradient vanishing does not occur are functions of the ReLU series that solve the gradient vanishing problem, this can also be analyzed as consistent with the theory.

[Pretrained Weight Problem] Since pretrained weights are weights stored based on the activation function of the model's existing structure, in order to compare performance between actual activation functions, the correct process is to start with pretraining while changing the activation function. In fact, GLU and swiglu have already proven in existing research that they show superior performance in transformers-based models, and also have actually been used in FFN (Feed Forward Network) of LLM (Large Language Model) such as llama2, but they showed no effect in this experiment. It is very likely that this result is due to the use of pretrained weights. Although we were unable to proceed with the pretraining process in this experiment due to limitations in computing resources, it is expected that different results will be obtained if we conduct an experiment that changes the activation function from pretrain as a follow-up study.

5 Deep Experiment on Activation Functions

In this section, we explore three main aspects: 1) Visualization of the evolution of activation function shapes per epoch for a 3-layer CNN architecture applied to CIFAR-10. 2) Comparison of the accuracy between the optimized set of activation functions discovered and when activation functions are fixed to ReLU or Leaky ReLU. 3) Observation of whether the gap in accuracy widens or narrows as the number of layers in the architecture increases.

5.1 Adaptive Activation Function Design

When using linear functions, the parameters can be trained to fit the input data, whereas when using nonlinear functions, one must pre-determine and start training among several nonlinear functions. A simple idea to learn the appropriate shape of the activation function during training is to learn the slope a and bias b for the left segment and the slope a and bias b for the right segment, based on a split point H. However, this method has the disadvantage that the number of parameters to be learned increases by 2 for each additional H. To address this issue, I adopted the adaptive activation function formula recently proposed by Agostinelli et al [6]. Here we define the adaptive piecewise linear (APL) activation unit. Our method formulates the activation function $h_i(x)$ of an APL unit i as a sum of hinge-shaped functions,

$$h_i(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s) \tag{1}$$

The result is a piecewise linear activation function. The number of hinges, S, is a hyperparameter set in advance, while the variables a_i^s, b_i^s for $i \in \{1, ..., S\}$ are learned using standard gradient descent during training. The a_i^s variables control the slopes of the linear segments, while the b_i^s variables determine the locations of the hinges.

The number of additional parameters that must be learned when using these APL units is $2SM$, where M is the total number of hidden units in the network. This number is small compared to the total number of weights in typical networks.

5.2 Experimental Design

In terms of experimental code, there exist four experiments numbered from 0 to 3. Experiment 0 verifies the role of S as a hyperparameter. It establishes the correlation between the increase in S and the fluctuation in accuracy to determine its significance. Experiment 1 visualizes the training and testing losses, as well as the accuracy changes over 10 epochs for a 3-layer architecture with S=5, using three different activation functions: APL units, ReLU, and LeakyReLU. This experiment is conducted in line with the referenced paper. Experiment 2 selects a subset of APL units appropriately trained from Experiment 1 and visualizes their activation function shapes at specific epochs (0 (pre-training), 1, 3, 5, and 10 epochs). This aims to verify if the identified activation function shapes exhibit generalization. If they do, these shapes could potentially be introduced as new activation functions. Experiment 3 examines how the gap in accuracy changes as the depth of the network varies, using an architecture where the user can define the number of layers (3, 5, 7). If the gap increases, it suggests the necessity of applying appropriate 'customized' activation functions to each layer. Conversely, if the gap decreases, it indicates that selecting the most suitable activation function, such as ReLU or its variants, for each layer might be a better approach. Based on these findings, insights into the current direction of activation function research can be provided.

5.3 Architecture Design

Our first network was loosely based on the network used in (Srivastava et al., 2014). [18] It had 3 convolutional layers with 96, 128, and 256 filters, respectively. Each kernel size was 5x5 and was padded by 2 pixels on each side. The convolutional layers were followed by a max-pooling, average-pooling, and average-pooling layer, respectively; all with a kernel size of 3 and a stride of 2. The two fully connected layers had 2048 units each. We applied dropout (Hinton et al., 2012) [9] to the network as well. The probability of a unit being dropped after a pooling layer was 0.25 for all pooling layers. The probability of a unit being dropped for the fully connected layers was 0.5 for both layers. The final layer was a softmax classification layer. This networks if for code 0 to 2.

Our second network generates a network containing the user-specified number of convolutional layers to process input images. Each convolutional layer employs 3x3 kernel sizes to process input images into 64 filters, and applies the user-selected activation function (ReLU, LeakyReLU, or APLLayer). Unlike the first architecture, the number of filters remains constant at 64. The network utilizes AdaptiveAvgPool2d to reduce spatial dimensions on average and employs Flatten to convert feature maps into 1-dimensional vectors. Finally, linear layers are utilized to transform 1-dimensional feature vectors into 10 classes for output. By supporting various activation functions and convolutional layer numbers, the architecture allows users to experiment with diverse network configurations. Our first network was designed to address the fact that as the number of layers increases, the size of the feature maps becomes smaller than 1x1. Therefore, this network was newly devised for Experiment code3.

5.4 Results

Through code 0, we confirmed that there is no linear relationship between S and accuracy as S increases. Subsequently, in the following codes, we adopted S=5 consistently, as described in the reference paper.

Values of S	Accuracy
S=1	69.64
S=2	77.53
S=3	76.51
S=4	81.24
S=5	80.65
S=10	81.33

Table 21. Table : Classification accuracy on CIFAR-10 for varying values of S. The table provides evidence supporting the assertion that S is a hyperparameter.

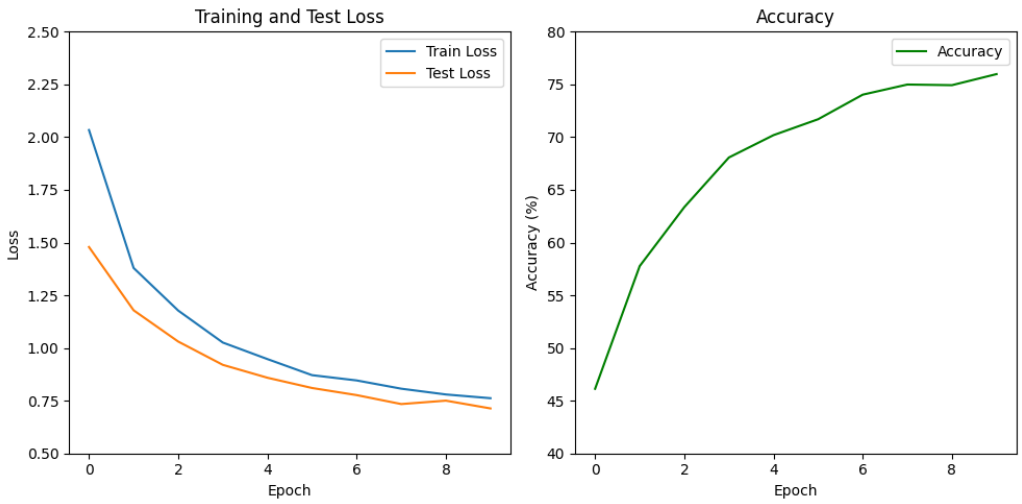

Fig. 11. training and test loss & accuracy with APL unit

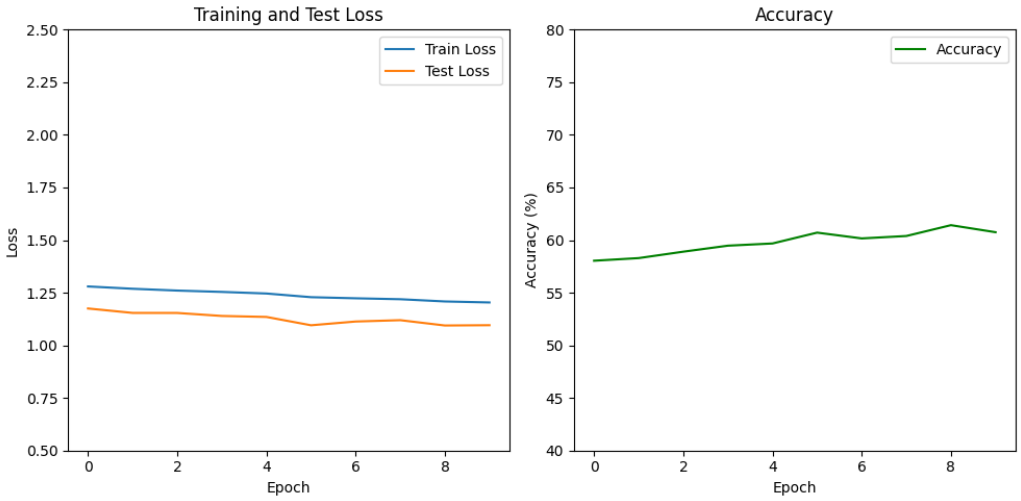

Fig. 12. training and test loss & accuracy with Relu

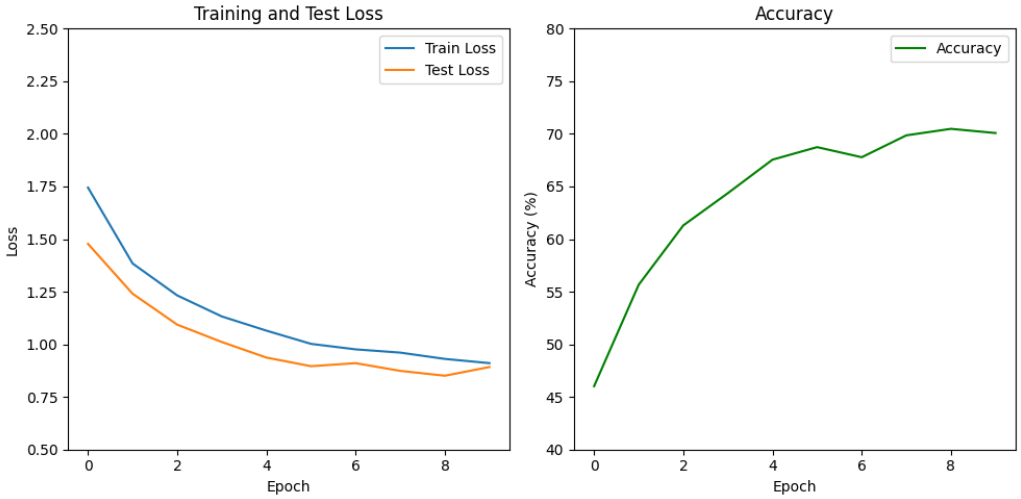

Fig. 13. training and test loss & accuracy with Leaky Relu

Through Code 1, we visualized the training and test loss, as well as accuracy, when using APL units optimized per neuron compared to fixed ReLU or Leaky ReLU activations. We observed variability in the training and test loss and accuracy during the initial epochs across multiple runs; however, beyond five epochs, we noted stability in these metrics. Table 22 provides an overview of the results. An intriguing observation is that while ReLU and Leaky ReLU initially exhibit better performance compared to APL units, the latter rapidly achieves higher accuracy as epochs progress. Leaky ReLU stands intermediate between APL units and ReLU in terms of stability and performance. Detailed graphs can be found in Figure 11, 12 and 13.

	First Epoch's Train Loss	Last Epoch's Test Loss	Final Accuracy
APL unit	2.0337	0.7134	75.98
ReLU	1.2797	1.0956	60.25
Leaky ReLU	1.7447	0.8923	70.08

Table 22. Overview of Results Obtained Through Code 1

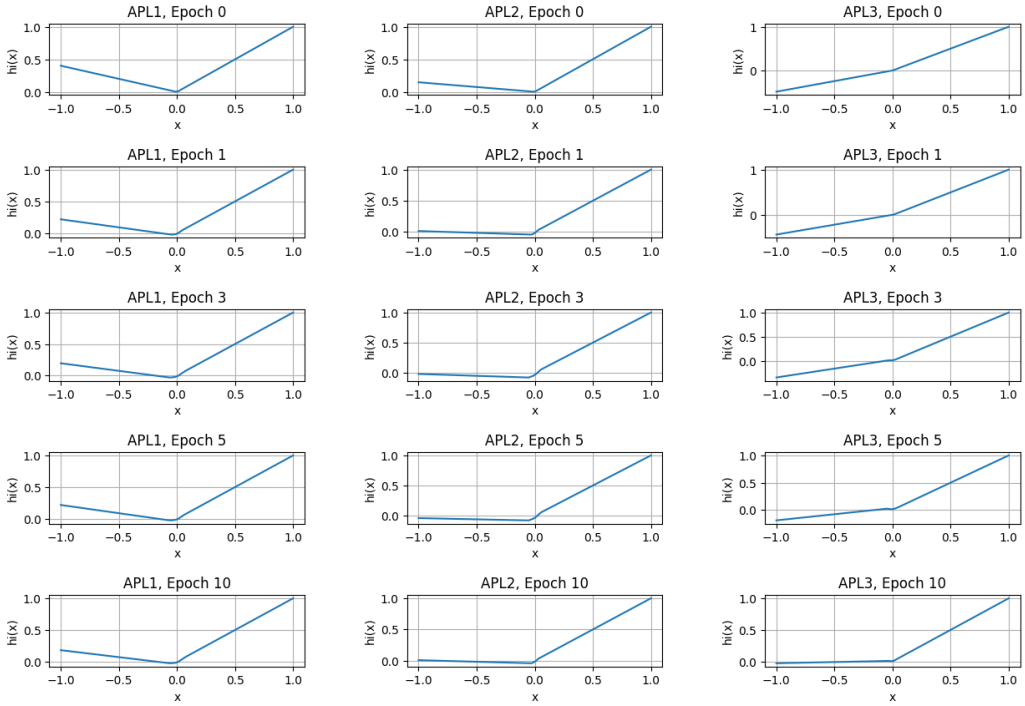

Fig. 14. visualization our actvation function

Next, we visualized how the APL unit, achieving superior performance (75.98%) compared to ReLU and Leaky ReLU, evolved and converged as epochs progressed. Given that there are 98, 128, and 256 neurons in layers 1, 2, and 3, respectively, it is not feasible to visualize the activation functions for all neurons. Therefore, we visualized the shapes for the first neuron in each layer before training (0), and after 1, 3, 5, and 10 epochs. All information for each neuron at each epoch is saved in params.zip, available on our GitHub repository for those interested in examining shapes for different neurons and epochs. The visualization results are shown in Figure 14. The value of S=5, determining the inflection points, creates a concave shape around 0, resembling the shape of ELU. We observed that if the initial random APL shape was concave, it remained concave after 10 epochs, and if it was convex initially, it continued to exhibit a convex shape thereafter. Particularly, APL3 closely resembled the shape of Leaky ReLU. The presence of numerous neurons exhibiting a concave shape after 10 epochs is intriguing, considering the limited research on concave activation functions thus far.

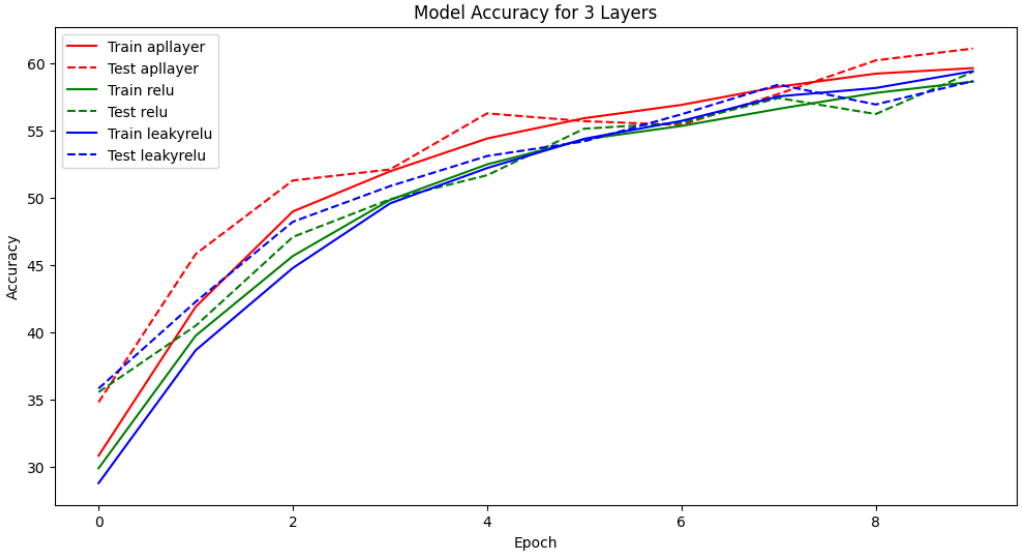

Fig. 15. accuracy on 3 layers

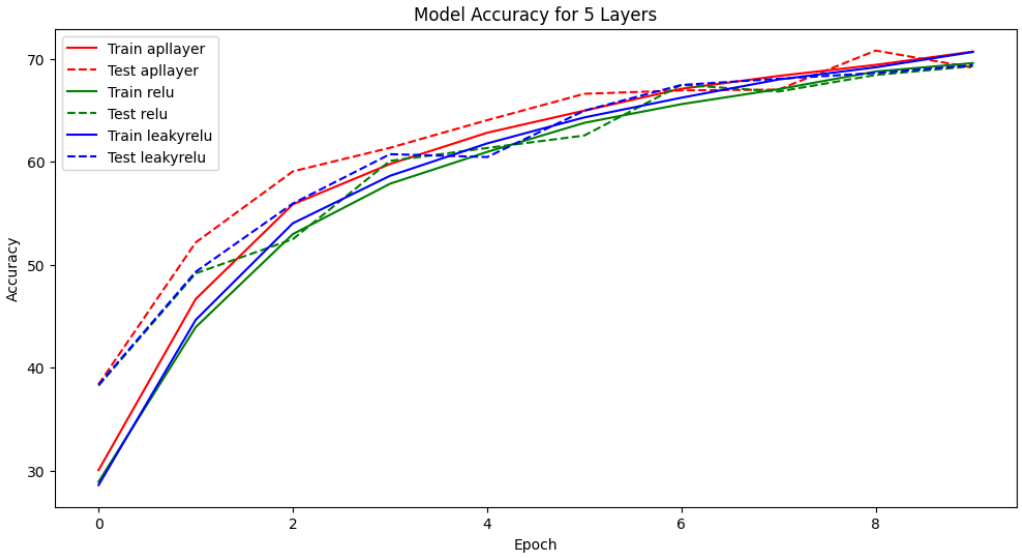

Fig. 16. accuracy on 5 layers

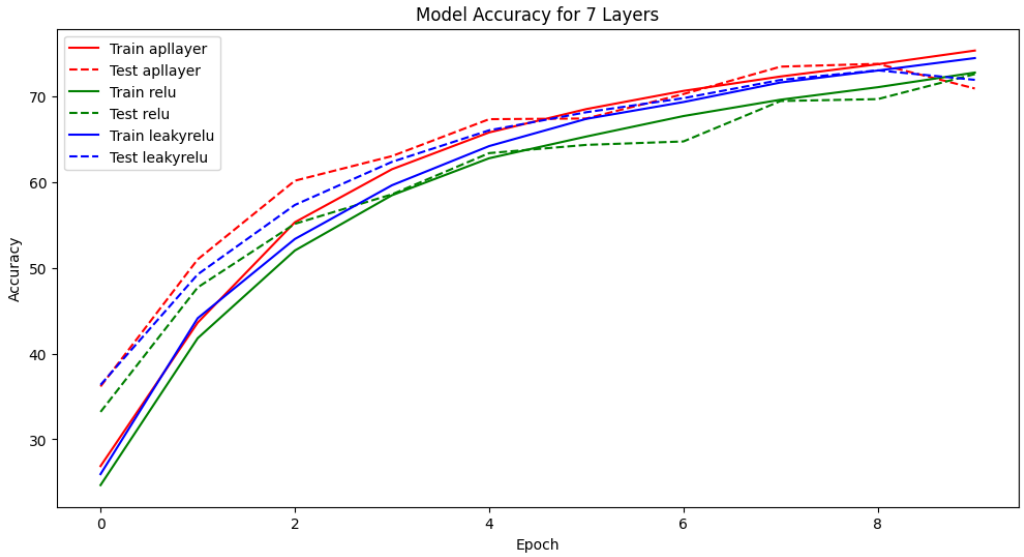

Fig. 17. accuracy on 7 layers

Finally, we investigated how the gap in accuracy varies with the depth of the layers depending on the type of activation function. The results can be observed in Figure 15, 16 and 17. As the depth of the layers increased, an overall increase in accuracy was observed. However, no significant findings regarding the gap in accuracy based on the type of activation function were made. This implies that the rate of increase in expressive power for ReLU and Leaky ReLU with increasing depth of the layers is similar to the rate of increase in expressive power when using the most appropriate activation function for each layer. However, as the depth of the layers increased, the gap narrowed slightly, and at 10 epochs in the 7th layer, an inverse pattern between ReLU and Leaky ReLU was observed. Nevertheless, to demonstrate this more accurately, further visualization with deeper layers and more epochs is necessary. But still, considering that the training time for APL units was approximately twice as long as using fixed activation functions, it was deemed more efficient to simply search through various nonlinear activation functions as hyperparameters.

5.5 Discussions

Using ReLU and Leaky ReLU fixed per layer exhibits stability, showing low training loss and decent performance from the first epoch. However, when training with APL units tailored per layer, despite the initially lower performance, significantly higher performance is achieved overall. Our experiments were conducted for up to 10 epochs under the same conditions. Graphically, it is anticipated that APL units could achieve even higher accuracy with additional training. Nonetheless, a lingering question remains whether these results are specific to our environment settings (model architecture and dataset). Even when visualizing the results with a new model architecture, APL units consistently exhibit higher accuracy, albeit with a small margin. Moreover, when extending the number of layers, albeit minimal, ReLU and Leaky ReLU show the possibility of accuracy reversal. Considering that APL units are likely to improve in performance with extended training time, approximately twice as long, it's understandable why researchers have commonly relied on stable results from ReLU and Leaky ReLU and why the use of activation functions has reached a plateau. We visualized the shapes of the learned activation functions, which closely resembled Leaky ReLU, albeit with

slight variations in the negative portion's slope. However, it's essential to note that these results may be influenced by the Kaiming He initialization used when assigning initial activation functions. The Kaiming He initialization was adopted following the configuration used in Caffe, which was referenced by the authors of the paper we relied on for our activation function formulas. Conducting additional experiments with random initialization, S=0 or 1, or exploring more diverse activation functions, such as those resembling y=x but with more complexity, could provide further insights.

6 Conclusion

About experiment 3.1, results vary depending on the optimizer and dataset. Also, despite the weight initialization technique favorable to only Sigmoid activation function, it did not show consistent results. Therefore, it is difficult to find the best AF in specific domain, and it is still best to find various optimal combinations according to the existing learning method. In addition, even if the theoretical proof of the latest activation function is lacking, it is necessary to find the optimal activation function for the target task through several experiments.

To address more about the experiment about ResNet50 3.2, further examining the visualization of gradients concerning Adam and MNIST, considering the ResNet50 utilized in this experiment, it's designed for large-scale image classification tasks. Hence, when applied to simple datasets like MNIST, the model may become overly complex, leading to the problem of gradient vanishing. Therefore, the issue of gradient vanishing with Adam and MNIST can be attributed to the combination of choosing Adam as the optimizer and the fact that ResNet50 might be overly complex for MNIST. Based on the results of epochs for different activation functions, it can be concluded that selecting the appropriate activation function depending on the dataset's complexity is crucial. Sigmoid or ReLU may be more efficient for simpler datasets, whereas using activation functions like ELU or SeLU might yield better performance on complex datasets. The insights from the test loss and test accuracy results indicate that the combination of LReLU with the optimization algorithm affects overall performance differently compared to other activation functions. Ultimately, the optimal activation function may vary depending on the dataset and the optimization algorithm used.

In the case of BERT, because pretrained weights were loaded, performance tended to decrease when the activation function used in the pretraining process was changed. In addition, activation functions excluding the relu series were not learned well due to a serious gradient vanishing problem, and experiments could not be conducted in various hyper parameter settings due to lack of computing resources and time. Therefore, in future experiments, it is necessary to learn by changing the activation function from the pretrain process and conduct experiments in various hyperparameter settings. Through this project, we approached the significant goal of task-specific generalization of activation functions. In this process, we gained an understanding of the origins of commonly used activation functions and explored why the usage of activation functions predominantly converges to ReLU and Leaky ReLU through various experiments. In Section 5 5, we went beyond the application of existing activation functions by introducing adaptive activation functions using nn.Module and conducted diverse experiments. While the concept of suggesting suitable activation functions for each layer yielded superior results compared to using predefined fixed activation functions, we observed that the gap is not substantial when employing more complex (deeper layers) architectures for overall accuracy improvement. Thus, we could understand the current trend in the artificial intelligence industry of using predefined fixed activation functions (ReLU, LeakyReLU) that show consistent results from the first epoch and do not require long training times. Nevertheless, there remains potential to explore completely new forms of activation functions in conjunction with biological perspectives. To experiment with this, it is necessary to construct new structures instead of existing ones like nn.Module and implement complex operations such as backpropagation from scratch. This report encapsulates a deep

understanding of activation functions, which are fundamental components supporting the outstanding achievements of artificial intelligence.

Through this project, we approached the significant goal of task-specific generalization of activation functions. In this process, we gained an understanding of the origins of commonly used activation functions and explored why the usage of activation functions predominantly converges to ReLU and Leaky ReLU through various experiments. In Section 5 5, we went beyond the application of existing activation functions by introducing adaptive activation functions using nn.Module and conducted diverse experiments. While the concept of suggesting suitable activation functions for each layer yielded superior results compared to using predefined fixed activation functions, we observed that the gap is not substantial when employing more complex (deeper layers) architectures for overall accuracy improvement. Thus, we could understand the current trend in the artificial intelligence industry of using predefined fixed activation functions (ReLU, LeakyReLU) that show consistent results from the first epoch and do not require long training times. Nevertheless, there remains potential to explore completely new forms of activation functions in conjunction with biological perspectives. To experiment with this, it is necessary to construct new structures instead of existing ones like nn.Module and implement complex operations such as backpropagation from scratch. This report encapsulates a deep understanding of activation functions, which are fundamental components supporting the outstanding achievements of artificial intelligence.

References

[1] 2021. *Bio-inspired Neurocomputing*. Springer Singapore. https://doi.org/10.1007/978-981-15-5495-7

[2] Andrew Y. Ng Andrew L. Maas, Awni Y. Hannun. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

[3] Anthony Gachagan Stephen Marshall Chigozie Nwankpa, Winifred Ijomah. 2018. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. (2018). https://arxiv.org/abs/1811.03378

[4] Youngmin Cho and Lawrence Saul. 2010. Large Margin Classification in Infinite Neural Networks. *Neural Computation* 22, 10 (2010).

[5] Sepp Hochreiter Djork-Arné Clevert, Thomas Unterthiner. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). https://arxiv.org/abs/1511.07289

[6] Peter Sadowski Pierre Baldi Forest Agostinelli, Matthew Hoffman. 2015. LEARNING ACTIVATION FUNCTIONS TO IMPROVE DEEP NEURAL NETWORKS. ICLR.

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, Vol. 15. JMLR WCP, 315–323.

[8] Andreas Mayr Günter Klambauer, Thomas Unterthiner. 2017. Self-Normalizing Neural Networks. https://arxiv.org/pdf/1706.02515v5

[9] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. *arXiv preprint* (2012). arXiv:arXiv:1207.0580

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer Feedforward Networks are Universal Approximators. In *Neural Networks*, Vol. 2. 359–366.

[11] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL.

[12] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. 2009. What is the Best Multi-stage Architecture for Object Recognition?. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2146–2153.

[13] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. 2015. Deep Residual Learning for Image Recognition. CVPR.

[14] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009). https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[16] Dabal Pedamonti. 2018. Comparison of Non-Linear Activation Functions for Deep Neural Networks on MNIST Classification Task. (2018). http://arxiv.org/abs/1804.02763

[17] PyTorch. [n. d.]. Training a Classifier. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[19] lryna Gurevych Steffen Eger, Paul Youssef. 2018. Is it Time to Swish? Comparing Deep Learning Activation Functions Across NLP tasks. EMNLP.

[20] Andrew James Turner and Julian Francis Miller. 2014. Neuroevolution: Evolving Heterogeneous Artificial Neural Networks. *Evolutionary Intelligence* (2014), 1–20.

[21] Geoffrey E. Hinton Vinod Nair. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

[22] Xin Yao. 1999. Evolving Artificial Neural Networks. *Proc. IEEE* 87, 9 (1999), 1423–1447.