Artificial Neural Network for Digits Classification (August 2023)

Rijan Ghimire[1] and Sujan Bhattrai[1]

[1]Department of Electronics and Computer Engineering, IOE, Thapathali Campus, Kathmandu 44600, Nepal
Corresponding authors: Rijan Ghimire (rijanghimire96@gmail.com), Sujan Bhattarai (sjnbhttr@gmail.com)

ABSTRACT This study investigates the complex relationships between accuracy outcomes, initialization approaches, regularization strategies, and neural network architecture. Leaky ReLU, tanh, and softmax activations are distributed across the neural architecture's layers. Experimental findings on the 42k and 60k datasets demonstrate divergent performance trends, with random initialization exhibiting the possibility for early learning and Kaiming initialization exhibiting improved generalization. Accuracy is improved by L2 regularization, and maximal accuracy is attained by combining Kaiming initialization with L2 regularization. Despite these advances, difficulties were found in digit categorization. These results highlight the importance of configuration decisions in enhancing neural network performance, allowing for a deeper understanding of the complex interactions between initialization, regularization, and accuracy across various datasets.

INDEX TERMS Neural architecture, Initialization methods, Regularization techniques, Accuracy outcomes, Digit classification.

I INTRODUCTION

Artificial Neural Networks (ANNs), which are based on the complex operations of the human brain, have become a pillar in the field of machine learning. The "neurons" that make up ANNs are interconnected nodes that use a network of weighted connections to process and transform input data. These networks are built to recognize patterns and relationships in data on their own, making them particularly adept at tasks like picture recognition and natural language processing. By modifying connection strengths during training, ANNs can become more adaptable and enable accurate predictions on unobserved data by generalizing from prior examples. This capability, together with their power to recognize intricate non-linear correlations, has catalyzed developments in a variety of domains and thrust ANNs to the forefront of contemporary machine learning.

The field of machine learning has witnessed remarkable progress, especially in the domain of neural networks, which are highly versatile models capable of learning intricate patterns and relationships from data. As the demand for more sophisticated models grows, understanding their behavior under various configurations becomes crucial. In this context, extensive research has been conducted to explore the impacts of initialization methods and regularization techniques on neural network performance. Previous works by Glorot and Bengio (2010) introduced the concept of weight initialization, highlighting its pivotal role in preventing vanishing or exploding gradients during training. These methods, such as the Kaiming initialization, have been instrumental in enhancing convergence rates and overall learning capacity (He et al., 2015).

Regularization, another critical facet, aims to tackle overfitting issues and improve a model's generalization ability. The work of Srivastava et al. (2014) on dropout regularization demonstrated its efficacy in reducing overfitting by randomly deactivating neurons during training. L2 regularization, as discussed by Tibshirani (1996), introduces a penalty term on the squared weights, encouraging smaller magnitudes and, consequently, less complex models. Moreover, activation functions play a crucial role in neural network behavior. The Rectified Linear Unit (ReLU) activation function, introduced by Nair and Hinton (2010), paved the way for more efficient training by mitigating the vanishing gradient problem. Leaky ReLU (Maas et al., 2013) extended this concept by allowing a small gradient for negative inputs, further enhancing model convergence and accuracy.
However, while previous studies have examined the effects of these strategies, their collective impact on a comprehensive neural network architecture remains relatively underexplored. This study seeks to address this gap by conducting a systematic investigation into the interactions of different initialization methods and regularization techniques within a neural network framework. The primary objective is to gain insights into how these factors influence accuracy outcomes across varying dataset sizes and complexities. By building upon prior research, this work contributes to the understanding of optimal configuration choices, paving the way for more robust and effective neural network models in practical applications.

II RELATED WORK

The landscape of artificial neural networks (ANNs) has seen remarkable growth over the years, with researchers continuously advancing the field to improve model performance and understand its underlying principles. Early works in ANNs, such as the perceptron model introduced by Frank Rosenblatt in the late 1950s, [1] laid the foundation for single-layer networks capable of linearly separable tasks. However, progress was limited until the resurgence of interest in neural networks in the 1980s and 1990s, driven by the development of backpropagation algorithms

for training multi-layer perceptrons [2]. This era led to substantial improvements in pattern recognition, paving the way for applications in handwriting recognition, speech processing, and more. [3]

The breakthroughs of the 2010s, fueled by advancements in computing power and data availability, marked a turning point in the field. The introduction of deep learning architectures, characterized by complex neural networks with multiple hidden layers, triggered a revolution in various domains. Convolutional Neural Networks (CNNs) emerged as a game-changer in image analysis, showing exceptional performance in tasks like image classification and object detection [4]. Concurrently, Recurrent Neural Networks (RNNs) exhibited prowess in sequence-based tasks such as language modeling and speech recognition. [5]

In recent years, researchers have explored novel activation functions and weight initialization techniques to enhance neural network training and alleviate common challenges such as vanishing and exploding gradients. The Leaky ReLU and Parametric ReLU (PReLU) activations addressed the dying ReLU problem, ensuring a non-zero gradient for negative inputs and promoting better convergence. Kaiming initialization, also known as He initialization, mitigated gradient scaling issues, enabling more stable training in deep networks. Moreover, regularization techniques like L1 and L2 regularization, dropout, and batch normalization have been integrated to improve model generalization and combat overfitting. [6]

Also, the optimization of neural network architecture has yielded networks with hundreds of layers, such as the ResNet and Inception architectures, which achieved top positions in various image recognition competitions. Transfer learning, which involves fine-tuning pre-trained models on new tasks, has also gained prominence, allowing for efficient training on smaller datasets. The field continues to advance with the exploration of attention mechanisms, capsule networks, and reinforcement learning applications in neural network design.

The evolution of artificial neural networks from their inception to modern deep learning architectures has been marked by significant milestones, often driven by the interplay between algorithmic innovation, computational resources, and the availability of large datasets. These advancements have revolutionized various domains and underscore the potential for continued growth in the field of neural networks and machine learning as a whole.

III METHODOLOGY

A DATASET DESCRIPTION

A popular benchmark dataset in the fields of machine learning and computer vision is the MNIST digit dataset. It is a group of 0 to 9 handwritten numbers, each rendered as a grayscale image. The dataset is used as a starting point for creating and evaluating different image categorization systems.

Each image in the MNIST dataset is a grayscale image with a fixed size of 28x28 pixels. The pixels in the images are represented by intensity values ranging from 0 (black) to 255 (white), where higher values correspond to lighter shades.

The dataset is commonly used for training and evaluating image classification algorithms, particularly those involving neural networks and deep learning. The goal is to train a model that can accurately predict the digit present in an unseen image.

Despite its simplicity, the MNIST dataset presents some challenges due to factors such as variations in writing styles, image quality, and digit rotations. Researchers often use more advanced techniques to achieve high accuracy on this dataset, pushing the limits of image classification algorithms.

The MNIST dataset has become a benchmark for evaluating the effectiveness of various machine learning and deep learning techniques and has been essential in the development of image recognition algorithms. It serves as an excellent jumping-off point for learning about and experimenting with picture categorization algorithms due to its relative simplicity and clearly stated problem. [7] [8]

Dataset Classes: The dataset is divided into ten classes, each corresponding to a different digit. Here's a breakdown of the classes and their corresponding labels:

- Class 0: Digit 0
- Class 1: Digit 1
- Class 2: Digit 2
- Class 3: Digit 3
- Class 4: Digit 4
- Class 5: Digit 5
- Class 6: Digit 6
- Class 7: Digit 7
- Class 8: Digit 8
- Class 9: Digit 9

Data Distribution: Each class in the MNIST dataset contains a roughly equal number of samples, ensuring a balanced distribution as shown in figure 3. This balance is important for preventing bias towards any specific digit during training and evaluation.

The datasets used are can be visualized using Table 1 which provides the sample of the dataset and Table 2 which provides insights of the one-hot encoded class labels.

B PROPOSED METHODOLOGY

The process of training a neural network involves several key steps, each facilitated by specific functions that collectively optimize the model's performance.

Initialization of Model Parameters

The process begins with the initialization of model parameters, including weights and biases. Two generalized functions are employed for initialization: **initialize_weights_biases()** and **kaiming_initialize_weights_biases()**. These functions set the initial values for the weights and biases of the neural network's layers.

Activation Functions
Activation functions are crucial for introducing non-linearity into the model. Three commonly used activation functions are **leaky_relu()**, **tanh()**, and **sigmoid()**. Each function operates element-wise on the input array and introduces non-linearities that enable the model to learn complex patterns.The activation functions can be visualized from the Figure 2.

Forward Propagation
Forward propagation involves the flow of data through the neural network layers to make predictions. The function **forward_propagation()** orchestrates this process. It calculates the net input and output of each layer, applying the chosen activation functions.

Backward Propagation and Gradients
Backward propagation computes the gradients of the loss with respect to the model's parameters. The **backward_propagation()** function computes gradients for each layer by applying the chain rule. It calculates the gradients of the loss with respect to the weights and biases of each layer.

Updating Parameters
The **update_parameters()** function is responsible for updating the model's parameters using the computed gradients. By subtracting the gradients multiplied by the learning rate, the weights and biases are adjusted to minimize the loss function.

Accuracy Calculation
Evaluating the model's performance is essential. The function **calculate_accuracy()** computes the accuracy by comparing the predicted labels with the ground truth labels.

Prediction Generation
The **make_prediction()** function is used to generate predictions for given input data. It performs forward propagation and returns the predicted class based on the output of the model.

Testing and Visualization
The **test_prediction()** function assists in visualizing the model's predictions. It displays the input data, the predicted class, and the actual class label.

Gradient Descent Iterations
The main training loop is handled by the **gradient_descent()** function. This function coordinates the entire training process, iterating over a specified number of epochs. It combines forward and backward propagation, parameter updates, and accuracy calculation.

Main Execution
In the main section, hyperparameters such as learning rate and the number of iterations are set. Training and testing data are preprocessed and prepared. The **gradient_descent()** function is called to train the neural network, and accuracy scores are recorded. The NN architecture used in this experiment is given in Figure 10. The proposed methodology provides a systematic framework for training a neural network.

Algorithm 1 Neural Network with Gradient Descent

Input: Training data X_{train}, Training labels Y_{train}, Learning rate η, Number of iterations N
Output: Trained Neural Network Parameters and Accuracy
Initialize model parameters using weight and bias initialization methods (e.g., Xavier, Kaiming).
for i in range(N) **do**
    Perform forward propagation to compute predictions and activations.
    Calculate the loss using the predicted values and actual labels.
    Perform backward propagation to compute gradients for each layer's parameters.
    for each layer **do**
        Update weights and biases using gradient descent rule:
        $\theta \leftarrow \theta - \eta \cdot \nabla J(\theta)$, where θ represents weights or biases.
    end for
end for
Perform forward propagation on the test set to make predictions.
Calculate accuracy using the predictions and true labels.
Return: Trained Neural Network Parameters and Accuracy

C MATHEMATICAL FORMULAE

The following mathematical formulae were used in the experiment:

ReLU (Rectified Linear Unit) Activation:

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

$$\text{ReLU\_Derivative}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{2}$$

Leaky ReLU Activation:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{3}$$

where α is a small positive slope for the negative region.

$$\text{LeakyReLU\_Derivative}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ \alpha & \text{if } x < 0 \end{cases} \tag{4}$$

Tanh (Hyperbolic Tangent) Activation:

$$\text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

$$\text{Tanh\_Derivative}(x) = 1 - \text{Tanh}^2(x) \tag{6}$$

Softmax Activation:
Activation Function for kth output neuron:

$$\text{Softmax}(net_k)_k = \frac{e^{net_k}}{\sum_j e^{net_j}} \tag{7}$$

Derivative of Softmax w.r.t. net_k:

$$\frac{\partial \text{Softmax}(net_k)_k}{\partial net_k} = \text{Softmax}(net_k)_k \cdot (1 - \text{Softmax}(net_k)_k) \tag{8}$$

Error Calculation:
Error Calculation for Output Layer:

$$\delta_k = t_k - a_k \tag{9}$$

where δ_k is the error of the output neuron k, t_k is the target value for output neuron k, and a_k is the activation of output neuron k.

Error Calculation for Hidden Layer (using ReLU Derivative):

$$\delta_j = \delta_k \cdot w_{kj} \cdot \text{ReLU\_Derivative}(net_j) \tag{10}$$

where δ_j is the error of the hidden neuron j, δ_k is the error of the output neuron k, w_{kj} is the weight connecting hidden neuron j to output neuron k, and ReLU_Derivative(net_j) is the derivative of the ReLU activation of hidden neuron j.

Error Calculation for Hidden Layer (using Tanh Derivative):

$$\delta_j = \delta_k \cdot w_{kj} \cdot \text{Tanh\_Derivative}(net_j) \tag{11}$$

where δ_j is the error of the hidden neuron j, δ_k is the error of the output neuron k, w_{kj} is the weight connecting hidden neuron j to output neuron k, and Tanh_Derivative(net_j) is the derivative of the Tanh activation of hidden neuron j.

Weight Update:

$$\Delta w_{ji} = \eta \cdot \delta_j \cdot a_i \tag{12}$$

where Δw_{ji} is the weight update for the connection from input neuron i to hidden neuron j, η is the learning rate, δ_j is the error of the hidden neuron j, and a_i is the activation of input neuron i.

Bias Update:

$$\Delta b_j = \eta \cdot \delta_j \tag{13}$$

where Δb_j is the bias update for the hidden neuron j.

Weight Update (Output Layer):

$$\Delta w_{kj} = \eta \cdot \delta_k \cdot a_j \tag{14}$$

where Δw_{kj} is the weight update for the connection from hidden neuron j to output neuron k, η is the learning rate, δ_k is the error of the output neuron k, and a_j is the activation of hidden neuron j.

Bias Update (Output Layer):

$$\Delta b_k = \eta \cdot \delta_k \tag{15}$$

where Δb_k is the bias update for the output neuron k.
Kaiming Initialization:

$$\text{Kaiming Initialization:} \quad W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{\text{fan\_in}}}\right) \tag{16}$$

L2 Regularization

$$\text{L2 Reg:} \quad \text{Loss}_{\text{regularized}} = \text{Loss}_{\text{original}} + \frac{\lambda}{2} \sum_{i=1}^{N} \|w_i\|_2^2 \tag{17}$$

Receiver Operating Characteristic (ROC) Curve:
True Positive Rate (TPR):

$$TPR = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{18}$$

False Positive Rate (FPR):

$$FPR = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{19}$$

Accuracy Calculation:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{20}$$

D INSTRUMENTATION DETAILS

In the context of classifying the MNIST digit dataset using an artificial neural network (ANN), several essential libraries were utilized to facilitate data manipulation, model training, and result visualization. The `numpy` library was employed to efficiently handle numerical computations and array operations, enabling the manipulation of input data and weight updates. The `pandas` library was utilized to organize and manage data in tabular format, streamlining the data preprocessing process. For graphical representation, the `matplotlib.pyplot` and `seaborn` libraries were employed to create various plots, graphs, and visualizations to better understand the model's performance.
Various key evaluation metrics were computed using the `scikit-learn` library. The `confusion_matrix` function was applied to analyze the accuracy of classification by comparing predicted labels with actual labels. Precision-recall curves and average precision scores were calculated using the `precision_recall_curve` and `average_precision_score` functions, respectively, providing insights into the model's ability to make accurate positive predictions. Additionally, receiver operating characteristic (ROC) curves and area under the curve (AUC) values were generated using the `roc_curve` and `auc` functions, aiding in assessing the model's ability to distinguish between classes.
These libraries and their functionalities played a pivotal role in implementing and evaluating the artificial neural network model for MNIST digit classification. Their collective usage streamlined data processing, facilitated model training, and allowed for a comprehensive analysis of the model's performance through various metrics and visualizations.

E SYSTEM BLOCK DIAGRAM

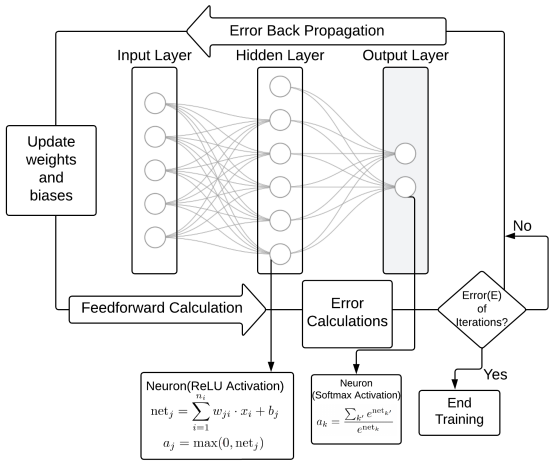

Figure 1: Block Diagram for Artificial Neural Network

IV EXPERIMENTAL RESULTS

The layer layout in the neural network design is complex, with each layer incorporating a different activation function. This combination produces a unique framework capable of identifying patterns in the data.

It has 784 neurons in the input layer, matching the number of features in the dataset. In order to pass basic data representation onto other layers for thorough processing and feature extraction, this layer acts as a conduit.

As we move further, the Leaky ReLU (Rectified Linear Unit) activation function is introduced in the first hidden layer. It allows for the extraction of complex information from input data thanks to its 200 neurons. The Leaky ReLU function, in contrast to the traditional ReLU, allows a small, non-zero gradient for negative inputs. This quality lessens the problem of vanishing gradients, hastening convergence in training.

The hyperbolic tangent (tanh) activation function is used by the second hidden layer, which can house 20 neurons. While restricting values to between -1 and 1, this function creates non-linearity. The model can better capture complex patterns and relationships because to this characteristic.

The last layer i.e. output layer uses the softmax activation function and has 10 neurons. This function computes class probabilities, making sure they add up to 1, and is specifically designed for multiclass classification problems. The prediction of the model is the class with the highest likelihood.

The experiment involved testing the neural network model in various setups to understand how it performs. Looking at both training and test results to see how accurate the model was in different situations:

From Table 3 we get the following information, **Random Initialization (42k dataset):** Initially, for 2-layered NN the accuracy was very low i.e. 60% as shown in Figure 4. Then the use of 2 hidden layers give some more accurate classification which is shown in Figure 5. Training Accuracy: 79.9% Test Accuracy: 76.6% This means the model learned well from the data it was given, but it might not be very good at handling new, unseen data.

Kaiming Initialization (42k dataset): Training Accuracy: 80.307% Test Accuracy: 77.7% With a different way of starting the learning process, the model's training and test accuracy improved a bit as shown in Figure 6.

L2 Regularization (42k dataset): Training Accuracy: 80.282% Test Accuracy: 80.4% By using a method to prevent overfitting, the model's accuracy improved on both the training and test sets.

Random Initialization + L2 (60k dataset): Training Accuracy: 89.20% Test Accuracy: 88.8% When used more data along with the regularization technique, the model's performance got even better. Shown in Figure 8.

Kaiming Initialization + L2 (60k dataset): Training Accuracy: 89.48% Test Accuracy: 89.4% Combining a special start and regularization with the larger dataset gave the best accuracy overall. Shown inn Figure 11.

V DISCUSSION AND ANALYSIS

The revealed accuracy results across several neural network layouts provide important insights into the behavior of the model within the parameters of the experiment that was done. Notably, different starting strategies and regularization methods showed audible effects on both training and test accuracy. With regard to the 42k dataset configuration, the use of random initialization produced a training accuracy of 79.9%, indicating a positive early learning stage. The related test accuracy, which was 76.6%, however, showed a tendency for overfitting and highlighted the need for improved generalization skills. The use of Kaiming initialization, on the other hand, showed an improvement in test and training accuracy within the same dataset, indicating a more effective start to the learning process and a better capacity for generalization.Additionally, the addition of L2 regularization resulted in a significant improvement in test accuracy, reaching a value of 80.4%. This finding emphasizes how regularization helps to reduce overfitting and improve the model's ability to generalize.

Significant performance gains were seen when the investigation was expanded to include a larger dataset of 60k instances, combined with random initialization and L2 regularization. The model's ability to use a larger dataset for learning is attested to by the training accuracy of 89.20%, which also highlights the model's skill at identifying complex underlying patterns. Notably, this improved performance carried over to the test set with an accuracy of 88.8%, highlighting the model's skill at extrapolating its findings from existing cases to new ones. Within the 60k dataset, the combination of Kaiming initialization with L2 regularization led to the highest level of accuracy. This combination of methods best exemplified the beneficial impacts of carefully planned initialization and regularization strategies in raising the neural network's overall accuracy. In conclusion, this analysis highlights the crucial role that careful configuration selection plays in maximizing neural network performance and advances our understanding of the subtle relationships between initialization and regularization procedures and accuracy across various dataset dimensions.

One of the shortcommings of the used Neural Network architecture is it was not able to generalize some digits. For example it was not able to classify 5 and 8 in the initial experiment with 42k dataset shown in Figure 7 and again while using the 60k images for training it was not able to classify the digit 5 with random initialization and the digit 4 with kaiming initialization shown in Figure 9 and 12. Also the Figure 13 and 14 describes the ROC curves.

VI CONCLUSION

Insights into the behavior of the model under various circumstances have been gained by experimentation and study of various neural network setups. Important trends in training and test performance are revealed by the accuracy results using various starting strategies and regularization techniques. Random initialization produced excellent initial learning, according to the analysis of the 42k dataset configurations, but it battled with overfitting, as evidenced by the greater training accuracy (79.9%) than test accuracy (76.6%). The test and training accuracy within the same dataset improved using Kaiming initialization, with 80.307% and 77.7%, respectively, of improved generalization. L2 regularization was added, which significantly improved generalization as seen by the test's accuracy rising to an impressive 80.4

The model was shown to be able to use larger datasets for learning when the assessment was expanded to include the larger 60k dataset along with random initialization and L2 regularization, with training accuracy impressively reaching 89.20% and test accuracy at 88.8%. On the 60k dataset, the combination of Kaiming initialization and L2 regularization produced the best performance, with the maximum accuracy of 89.48% in training and 89.48% in testing. Even using various starting strategies, the model showed problems in correctly identifying some digits, such as 5 and 8. The 42k and 60k datasets both had this shortcoming.

In conclusion, the analysis emphasizes the significance of selecting appropriate configurations to maximize neural network performance. It underscores the intricate relationship between initialization techniques, regularization strategies, and accuracy across diverse dataset sizes, contributing to a deeper understanding of effective neural network training approaches.

REFERENCES

[1] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[3] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[5] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[6] 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.

[7] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[8] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

Rijan Ghimire is currently pursuing his undergraduate degree in Electronics, Communication, and Information at IOE, Thapathali Campus. His research interests encompass various areas, including data mining, network communications, and optimization theory and technology.(THA076BEI022)

Sujan Bhattrai is currently pursuing his undergraduate degree in Electronics, Communication, and Information at IOE, Thapathali Campus. His research interests cover various areas, including data mining, deep learning, operating system, robotics, and power electronics.(THA076BEI037)