Indonesian Journal of Artificial Intelligence and Data Mining (IJAIDM)
Vol 6, No.2, September 2023, pp. 220 – 230
p-ISSN: 2614-3372 | e-ISSN: 2614-6150

Optimizing Malware Detection using Back Propagation Neural Network and Hyperparameter Tuning

[1]Annisa Arrumaisha Siregar, [2]Sopian Soim [3]Mohammad Fadhli
[1,2,3]Departement of Electrical Engineering, Telecommunication Engineering Study Program,
State Polytechnic of Sriwijaya
Email: [1]annisaarrumaisha19@gmail.com, [2]sopiansoim@gmail.com, [3]mohammad.fadhli@polsri.ac.id

Article Info

Article history:
Received May 17th, 2023
Revised Jul 25th, 2023
Accepted Aug 30th, 2023

Keyword:
Back Propagation
Cyber Threats
Dropout Regularization
GridSearchCV
Hyperparameter Tuning
Malware Detection

ABSTRACT

The escalating growth of the internet has led to an increase in cyber threats, particularly malware, posing significant risks to computer systems and networks. This research addresses the challenge of developing sophisticated malware detection systems by optimizing the Back Propagation Neural Network (BPNN) with hyperparameter tuning. The specific focus is on fine-tuning essential hyperparameters, including dropout rate, number of neurons in hidden layers, and number of hidden layers, to enhance the accuracy of malware detection. A Back Propagation Neural Network (BPNN) with dropout regularization is trained on an extensive dataset as part of the research design. Hyperparameter optimization is conducted using GridSearchCV, with experiments varying learning rates and epochs. The best configuration achieves outstanding results, with 98% accuracy, precision, recall, and F1-score. The proposed approach presents an efficient and reliable solution to bolster cybersecurity systems against malware threats.

Corresponding Author:
Annisa Arrumaisha Siregar,
Department of Electrical Engineering, Telecommunication Engineering Study Program,
State Polytechnic of Sriwijaya
Jl. Srijaya Negara, Bukit Besar, Kecamatan Ilir Barat I, Kota Palembang, Sumatera Selatan.
Email: annisaarrumaisha19@gmail.com

DOI: http://dx.doi.org/10.24014/ijaidm.v6i2.24731

1. INTRODUCTION

The rapid expansion of the internet has become an essential aspect of daily life worldwide. As of April 2023, there were over five billion internet users, constituting approximately 64.6% of the global population [1]. This interconnected web of computers, networks, and interconnected devices has been made possible through continuous technological advancements in computer systems, networks, and mobile devices, leading to a significant surge in internet usage. However, this exponential growth in internet users has also escalated the vulnerability of computer systems and networks to cyberattacks [2]. Among the various cyber threats, one of the most significant risks to the security of computer systems and networks is malware, which stands for malicious software. Malicious programs are specifically designed to infiltrate, cause harm, or steal sensitive information from the targeted system. In the realm of cybersecurity, the detection of malware plays a crucial role in preventing potential data breaches and thwarting cyberattacks. Effectively identifying and combating malware is of utmost importance in safeguarding the integrity and confidentiality of sensitive data.

To address these security challenges effectively, it is crucial to prioritize the development of a sophisticated malware detection system, leveraging the capabilities of machine learning (ML) and deep learning (DL) [3]. Cybersecurity experts firmly believe that ML/DL-based anti-malware software will significantly improve the detection of new malware types and enhance existing scanning engines. Machine learning and deep learning offer distinct advantages in malware detection and analysis tools. Traditional machine learning techniques rely on feature engineering and feature representation techniques, which demand domain knowledge [4]. Currently, deep learning, an enhanced neural network model, has outperformed traditional machine learning in various tasks such as speech recognition, pattern identification, image

classification, and many other classification tasks. Deep learning automatically extracts features at each layer, as it has the capability of learning high-level feature representations through layered training from the lowest to the highest levels [5].

In previous research, several studies on malware detection have been conducted using various techniques. For instance, Saxe and Berlin [6] proposed a malware classification system constructed using a neural network with two hidden layers. These hidden layers were composed of 1024 Parametric Rectified Linear Units (PReLu), while the output layer utilized a sigmoid neuron to classify instances as either malware or benign. They have achieved a 95% detection rate with a 0.1% false positive rate when running an experiment on a 400,000 samples dataset. Huda et al. [7] proposed multilayer perceptron artificial neural network trained with backpropagation algorithm to determine an Android application is malware or benign. The experimental results show that the proposed method is effective, with a relatively high accuracy in recognizing existing malware samples. Pan,et al. [8] proposed a malware classification approach based on dynamic analysis, using dynamic analysis to obtain behavior profiles which are then used to abstract malware features and input the Back Propagation (BP) Neural Network model. The experimental results show the technique is effective and accurate. The methodology has an average accuracy of up to 86%, with non-malware categories predicting accuracy up to 99%. This suggests that the methodology can be easily extended for malware detection.

Makandar and Patrot [9] explored the classification of malware samples by converting them into grayscale images and using texture features for analysis. They extracted a total of 512 feature vectors using Gabor Wavelet Transform (GWT) and the Generic Fourier Descriptor (GIST) to observe the malware behavior. From these feature vectors, they selected 320 two-dimensional features and applied an Artificial Neural Network (ANN) for classification. The choice of using a feed-forward artificial neural network was driven by its capability to identify hidden patterns within complex data, such as images, sounds, and signals, while also providing implementation flexibility. For the training process, the authors utilized the backpropagation algorithm. Despite their dedicated efforts, they encountered challenges in effectively handling high-dimensional feature vectors, which resulted in a suboptimal accuracy of 96.35% for the malware classification task. Babak et al. [10] developed a neural network model for classifying PE files, employing the Gradient Descent algorithm with a decayed learning rate to update the weights during training. The model utilized the softmax activation function for the output neurons. With a dataset comprising 4,000 samples (3,000 malicious PE files and 1,000 benign), the implemented neural network achieved impressive performance metrics. The average accuracy attained was 97.8%, with a precision of 97.6% and a recall of 96.6%. These findings demonstrate the efficacy of their approach in accurately classifying PE files into malicious and benign categories. The utilization of the Gradient Descent algorithm and softmax activation function showcases the model's ability to handle complex data and effectively distinguish between potentially harmful and benign files.

Previous research has demonstrated the promising efficacy of Back Propagation Neural Networks (BPNNs) in malware detection, owing to their ability to learn intricate patterns and relationships within data. Nonetheless, the performance of BPNNs is heavily dependent on hyperparameters, which dictate the network's architecture and learning process. In this study, our objective is to optimize the malware detection process by fine-tuning key hyperparameters of the Back Propagation Neural Network through Hyperparameter Tuning. Our focus centers on three critical hyperparameters: the dropout rate, the number of hidden units in each layer, and the number of hidden layers. Leveraging dropout regularization helps prevent overfitting and enhances the model's generalization capability. Systematically evaluating various combinations of these hyperparameters, we endeavor to identify the optimal settings that will yield improved accuracy in malware detection.

To achieve this optimization, we propose a novel approach by utilizing a Back Propagation Neural Network (BPNN) with dropout regularization. The inherent capability of BPNN architecture in handling complex data makes it a suitable candidate for classifying malware. While the existing literature has explored the use of Artificial Neural Networks (ANNs) for malware classification, the specific application of BPNNs with dropout regularization to optimize malware detection using Back Propagation Neural Networks and Hyperparameter Tuning remains relatively underexplored. This study seeks to make a substantial contribution to the advancement of cybersecurity systems against malicious threats, ultimately culminating in an optimized and efficient malware detection solution.

2. RESEARCH METHOD

2.1. Research Design

The research design for this study is structured to optimize malware detection using Back Propagation Neural Network (BPNN) in conjunction with Hyperparameter Tuning. The process follows a systematic and well-defined flow aimed at enhancing the accuracy and efficiency of malware detection. The first step involves acquiring the dataset containing malware detection data. For this purpose, publicly available data was collected, specifically utilizing the dataset for the classification of malware with PE Headers sourced from github.com [11]. This dataset serves as the foundation for classifying PE files into two categories: malware and benign

(legitimate). The dataset used in the study is extensive, comprising over 138,000 instances. Among these, 96,724 samples were identified as malware, while 41,323 samples were labeled as benign (legitimate).

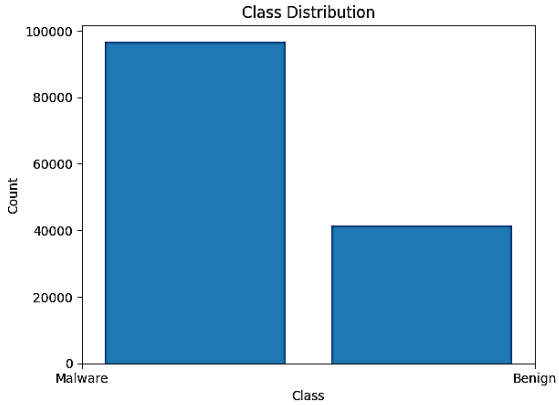

Figure 1. Class Distribution

Once the dataset is obtained, preprocessing techniques are applied to cleanse the data and prepare it for further analysis. This involves handling missing values and normalizing the dataset. The normalization method employed is MinMaxScaler. This method transforms each value in the dataset to a range between 0 and 1, where 0 represents the minimum value in the dataset, and 1 represents the maximum value. During the normalization process using MinMaxScaler, each value in the dataset is subtracted by the minimum value in the dataset, and the result is divided by the difference between the maximum and minimum values [12]. As a result, the range of values for all attributes in the dataset becomes uniform, allowing attributes with different value ranges to be treated fairly during analysis and modeling [12].

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

The preprocessed dataset is divided into two subsets: the training set and the testing set. The training set is used for model training, allowing the Back Propagation Neural Network (BPNN) to learn from the data. The testing set, on the other hand, is used to evaluate the model's performance on new and independent data. The data is split in an 80-20 ratio, with 80% allocated to the training set and 20% to the testing set. This ensures that the model learns from a significant portion of the data and can effectively identify intricate patterns and relationships. The testing set provides an unbiased assessment of the model's ability to generalize to new and unseen data, reflecting its real-world performance.

The core of the research design lies in the architecture of the BPNN model and the utilization of Dropout Regularization. The model consists of multiple layers, each comprising a certain number of neurons, with different activation functions. The input layer of the neural network is determined by the number of features present in the preprocessed dataset. In this study, the number of input neurons corresponds to the number of features, facilitating the extraction of relevant information from the data. Subsequently, the model includes multiple hidden layers, each containing a specified number of hidden units. These hidden layers play a vital role in learning intricate patterns and relationships within the data, enabling the model to detect and classify malware instances effectively. The number of hidden layers and hidden units are hyperparameters that undergo tuning during the optimization process.

To mitigate the risk of overfitting, dropout regularization is applied after each hidden layer. Dropout refers to the process of dropping out units (both hidden and visible) in an artificial neural network. Dropping out units involves temporarily removing these units from the neural network along with all the connections that lead to and from the respective units [13]. The dropout rate, which is another hyperparameter, determines the proportion of neurons that are randomly deactivated during each training iteration. By using this regularization technique, the model becomes more robust and avoids excessive reliance on specific neurons, thereby enhancing its generalization capability.

Furthermore, the activation function used in the input layer and hidden layers is Rectified Linear Unit (ReLU), which allows the model to handle non-linear relationships in the data effectively. The ReLU activation function, proposed by [14], is founded on robust biological and mathematical principles. Initially introduced in 2011, it demonstrated significant advancements in training deep neural networks. The function operates by

setting values below 0 to 0, effectively using f(x) = max(0, x). In simpler terms, it outputs 0 when x < 0 and a linear function when x ≥ 0.

$$f(x) = max(0, x) = \{xi, if\ xi \geq 0;\ 0, if\ xi < 0 \quad (2)$$

The output layer, which is responsible for handling the binary classification problem (distinguishing between malware and benign samples), uses the sigmoid activation function. The sigmoid function, also known as the logistic function, is a non-linear activation function that maps the output of a neuron to a range between 0 and 1 [15] [16] [17]. This mapping allows the sigmoid activation function to interpret the output as a probability, indicating the likelihood of a particular class being present in the input data. Mathematically, the sigmoid activation function can be expressed as follows:

$$f(x) = \frac{1}{1+e-x} \quad (3)$$

where x is the input to the activation function, and $f(x)$ is the output in the range [0, 1]. When the input x is positive, the output $f(x)$ tends to 1, and when x is negative, the output $f(x)$ tends to 0. This property makes the sigmoid function suitable for binary classification problems, where the goal is to classify data into one of two distinct classes.

The advantages of using the sigmoid activation function in the output layer for binary classification tasks are twofold. Firstly, the sigmoid function offers a natural interpretation of the output as a probability score, facilitating the establishment of a decision threshold, typically set at 0.5, for making class predictions. In our study, we implemented this by classifying input data as benign if the output probability is greater than 0.5, and as malware if the probability is less than or equal to 0.5. This thresholding mechanism simplifies the decision-making process, promoting clarity and logical reasoning in binary classification tasks.

Secondly, the sigmoid activation function's ability to compress the output in the range [0, 1] helps mitigate the issue of vanishing gradients during backpropagation [17]. Vanishing gradients occur when the gradient becomes too small as it propagates backward through the layers during training. This can hinder the learning process, especially in deep networks. The sigmoid function's derivative ranges from 0 to 0.25, which helps avoid extremely small gradients compared to other activation functions with larger derivative ranges.

In order to optimize the performance of the neural network model, we employ the Adam optimizer, an adaptive learning rate algorithm widely used in training deep neural networks [18]. The Adam optimizer combines the advantages of both the Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSprop) [19]. By dynamically adjusting the learning rates for each parameter during training, Adam is particularly effective in handling sparse gradients and non-stationary objective functions [20] [21], leading to faster convergence and better generalization on unseen data.

The key advantage of the Adam optimizer lies in its ability to maintain two moving average estimates: the first moment (mean) of the gradients and the second moment (uncentered variance) of the gradients [19]. These estimates are calculated using exponential moving averages and bias correction, effectively mitigating the impact of noisy gradients and improving optimization performance [19]. Furthermore, Adam incorporates bias correction during the initial optimization steps, especially beneficial when dealing with parameters having large gradients. This correction ensures a smoother optimization process, preventing overshooting or oscillation during parameter updates.

The Adam optimizer updates the model's parameters based on the gradients of the loss function with respect to those parameters. It combines the concepts of momentum and RMSprop to achieve adaptive learning rates for each parameter during training. The formulas for updating the parameters using the Adam optimizer are as follows [20]:

1. Moving average of the first moment (mean) of the gradients (m):

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t \quad (4)$$

2. Moving average of the second moment (uncentered variance) of the gradients (v):

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . {g_t}^2 \quad (5)$$

3. Bias-corrected first moment estimate:

$$\widehat{m}_t = \frac{m_t}{1-{\beta_1}^t} \quad (6)$$

4. Bias-corrected second moment estimate:

$$\hat{v}_t = \frac{v_t}{1-\beta_2{}^t} \quad (7)$$

5. Update rule for each parameter (θ):

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_{t+\epsilon}}}.\hat{m}_t \quad (8)$$

where:
m_t represents the moving average of the gradients' first moment at time step t.
v_t represents the moving average of the gradients' second moment at time step t.
g_t represents the gradients of the loss function with respect to the parameters at time step t.
β_1 and β_2 are the exponential decay rates for the first and second moment estimates, respectively (typically set to 0.9 and 0.999).
η is the learning rate, which controls the step size of the parameter updates.
ϵ is a small constant (usually 10^{-8}) added to the denominator for numerical stability.

The Adam optimizer is renowned for its adaptive learning rate capability, which efficiently optimizes deep neural networks by adjusting learning rates based on historical gradients [18]. This adaptability, coupled with the integration of momentum and RMSprop concepts, firmly establishes Adam as a powerful and widely employed algorithm in the field of deep learning. Notably, its effectiveness extends to large datasets and high-dimensional feature spaces, rendering it a valuable asset for deep learning research [19].

In this research, we employ the binary cross-entropy loss function as a vital element in training neural networks for binary classification tasks. The loss function, also known as a cost function or objective function in machine learning, quantifies the disparity between the model's predicted values and the true labels within the training data. The primary purpose of the loss function is to guide the optimization process by providing a measure of how well the model is performing on the task at hand [22].

Binary cross-entropy is a specific type of loss function well-suited for binary classification tasks, where the objective is to categorize data into one of two possible classes (e.g., malware or benign). The binary cross-entropy loss function computes the cross-entropy between the predicted probabilities and the true binary labels of the data samples. It assesses the dissimilarity between the model's predictions and the actual labels, taking into account both positive and negative samples. The formula for binary cross-entropy is given as:

$$Binary\ Cross-Entropy = -\frac{1}{N}\sum_{i=1}^{N}(y_i.log(p_i) + (1-y_i).log(1-p_i)) \quad (9)$$

One of the advantages of using binary cross-entropy in the research being conducted is its suitability for binary classification tasks and its compatibility with the sigmoid activation function in the output layer. The sigmoid activation function restricts the model's output to a probability value between 0 and 1, allowing for the interpretation of predictions as class probabilities for binary classification. This aligns well with the binary cross-entropy loss, which measures the alignment between predicted probabilities and true binary labels, incentivizing the model to produce confident predictions for the correct class.

Additionally, binary cross-entropy provides a clear and interpretable measure of the model's performance. Lower values of binary cross-entropy indicate better alignment of the model's predictions with the true binary labels, implying improved classification accuracy. Thus, the choice of binary cross-entropy as the loss function in the research supports effective training of the neural network and accurate classification of samples into the two classes, namely malware and benign, enabling reliable analysis and detection in the domain of malware detection.

The model in this research undergoes training and hyperparameter tuning using a custom loop with GridSearchCV (Grid Search Cross-Validation), a powerful technique for hyperparameter optimization in machine learning [23]. GridSearchCV systematically explores a specified hyperparameter grid to identify the combination that yields the best model performance on the testing data. The hyperparameters being tuned in this research are the dropout rate, number of hidden units, and number of hidden layers. The dropout rate is a regularization technique that randomly drops out a fraction of neurons during training to prevent overfitting. The number of hidden units refers to the number of neurons in each hidden layer, and the number of hidden layers determines the depth and complexity of the neural network architecture.

GridSearchCV exhaustively searches all possible hyperparameter combinations within the grid. Each combination is trained and evaluated through cross-validation, where the dataset is split into multiple folds,

and each fold acts as a validation set. This approach ensures a robust and unbiased evaluation of the model's performance [24]. The key advantage of using GridSearchCV in this research is its ability to automate hyperparameter tuning and explore a wide range of combinations, leading to the identification of optimal hyperparameters for improved testing data accuracy. GridSearchCV systematically compares various hyperparameter combinations, aiding researchers in making informed decisions regarding neural network architecture and configuration. Furthermore, GridSearchCV's evaluation on multiple folds of the dataset ensures the selection of hyperparameters that generalize well to new data and minimize overfitting risks.

Once the best model has been selected, the training process proceeds with the epochs and learning rate specified for the experimentation. In this research journal, we conduct a series of experiments to explore the impact of different learning rates on the model's performance. Specifically, we test four learning rates: 0.1, 0.05, 0.01, and 0.001. The learning rate plays a crucial role in the optimization process, determining the step size for updating the model's parameters during training. By examining various learning rates, we aim to identify the most suitable rate that maximizes the model's convergence and classification accuracy. Additionally, we perform the training process for two different epoch values, namely 100 and 200. This variation in the number of epochs allows us to observe how the model's performance evolves over time and how additional training iterations affect its accuracy and other evaluation metrics.

For the evaluation of the model's performance, we choose essential metrics, including accuracy, precision, recall, and F1-score. These metrics serve as comprehensive measures of the model's classification capabilities in distinguishing between malware and benign samples. Accuracy provides an overall assessment of the model's correctness, while precision evaluates its ability to correctly identify positive samples. Recall measures the model's capability to detect all positive instances in the dataset, and F1-score strikes a balance between precision and recall, providing a more nuanced understanding of the model's performance.

By systematically analyzing the model's behavior and performance across different learning rate s and epochs using these evaluation metrics, our research aims to gain valuable insights into the optimal hyperparameter configurations that lead to superior malware detection accuracy. The findings from this study will contribute to advancing the field of cybersecurity and fostering the development of more robust and accurate malware detection systems.

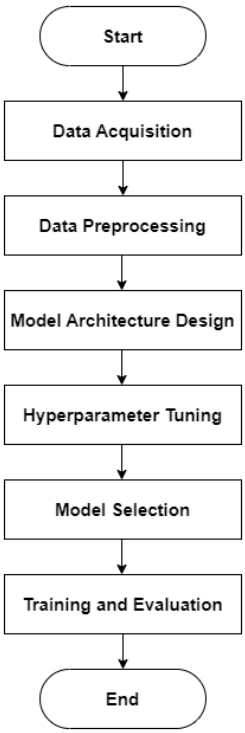

Figure 2. Research Design Flowchart

2.2. Model Evaluation

The model evaluation process is crucial to assess the effectiveness of the proposed method. It aims to measure the model's capability in accurately classifying malware and benign samples. The evaluation involves employing well-established performance metrics, namely accuracy, precision, recall, and F1-scores, to provide an objective and comprehensive analysis of the model's performance.

Accuracy assesses the overall correctness of the model's predictions by measuring the percentage of correctly classified samples. Precision evaluates the model's ability to correctly identify positive samples (benign) out of all predicted positive instances. Recall measures the model's capacity to detect all positive samples from the entire dataset of positive samples. F1-score strikes a balance between precision and recall, providing a harmonic mean that indicates the model's overall effectiveness in binary classification tasks.

Table 1. Performance Metrics for Evaluation

Metric	Formula	Description
True Positive (TP)	count()	Number of samples correctly classified as benign
False Positive (FP)	count()	Number of samples incorrectly classified as benign
True Negative (TN)	count()	Number of samples correctly classified as malware
False Negative (FN)	count()	Number of samples incorrectly classified as malware
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Proportion of correctly predicted samples
Precision	$\frac{TP}{TP+FP}$	How well the model predicts benign samples as benign
Recall/TPR	$\frac{TP}{TP+FN}$	How well the model is able to find all benign samples
F1-score	$2 \times \frac{Precision \times Recall}{Precision + Recall}$	Harmonic mean of precision and recall, measures the balance between precision and recall

By utilizing these performance metrics, the research ensures a rigorous and systematic evaluation of the model's classification capabilities. The results obtained from the evaluation process will play a significant role in validating the proposed approach and making informed decisions about its practical applicability in the field of malware detection.

3. RESULTS AND ANALYSIS

In this section, we present the outcomes and analysis of our research aimed at optimizing malware detection using the Back Propagation Neural Network (BPNN) in conjunction with hyperparameter tuning. Following the selection of the best model through GridSearchCV, we conducted multiple experiments by varying the learning rates and epochs. Specifically, we tested learning rates of 0.1, 0.05, 0.01, and 0.001, while setting epochs to 100 and 200.

The experimental results from the various learning rates and epochs are presented in Table 2. The table provides a comprehensive analysis of the model's performance, presenting key metrics such as accuracy, precision, recall, and F1-score. These metrics play a crucial role in evaluating the model's capability to accurately detect and distinguish between malware and benign samples. Additionally, we meticulously recorded the computational time required for each experiment, allowing us to thoroughly assess the model's training efficiency.

Table 2. Experimental Results of Malware Detection using BPNN with Hyperparameter Variations

learning rate	Epoch	Best Hyperparameter according to GridSearchCV			Acc (%)	Prec (%)	Recall (%)	F1-score (%)	Computational Time (s)
		Dropout Rate	Number of Neurons in the Hidden Layer	Number of Hidden Layers					
0.1	50	0.2	20	1	97	97	97	97	1144.91
0.05	50	0.2	20	1	97	97	97	96	1189.57
0.01	50	0.1	10	2	97	97	97	97	734.31
0.001	50	0.2	20	2	97	97	97	97	1290.57
0.1	100	0.1	20	1	97	97	97	97	2292.28
0.05	100	0.1	20	1	97	97	97	97	2346.54
0.01	100	0.1	20	1	97	97	97	97	1421.77
0.001	100	0.1	20	2	97	97	97	97	1581.69
0.1	**200**	**0.1**	**20**	**1**	**98**	**98**	**98**	**98**	**6325.65**
0.05	200	0.1	20	2	96	97	96	97	5414.67
0.01	200	0.1	20	2	97	97	97	97	7464.18
0.001	200	0.2	20	1	97	97	97	97	5459.82

From the table, we observed that different hyperparameter values influence the model's performance in detecting malware. For instance, varying the learning rate and Epochs led to variations in the model's

accuracy, precision, recall, and F1-score. Notably, higher values for these metrics indicate a more accurate and reliable detection system. Furthermore, the Dropout Rate, Number of Neurons in Hidden Layer, and Number of Hidden Layers also played a crucial role in shaping the model's performance. By tuning these hyperparameters, the model achieved significant improvements in detecting both malware and benign samples.

Upon analyzing the table, several noteworthy trends become evident. Firstly, the hyperparameter combination with a learning rate of 0.1, 200 epochs, and a dropout rate of 0.1, along with a single hidden layer containing 20 neurons, consistently produced the highest performance across all metrics, achieving an outstanding accuracy, precision, recall, and F1-score of 98%. Moreover, this configuration demonstrated a reasonable computational time of 6325.65 seconds, striking an effective balance between computational efficiency and model performance.

Secondly, the combination with a learning rate of 0.01, 50 epochs, and a dropout rate of 0.2, along with a single hidden layer containing 20 neurons, also exhibited competitive results, achieving 97% accuracy, precision, recall, and F1-score. Additionally, this configuration demonstrated a relatively low computational time of 734.31 seconds, further highlighting its efficiency.

Nevertheless, it is essential to consider other combinations as well. For instance, the hyperparameter combination with a learning rate of 0.05, 100 epochs, and a dropout rate of 0.1, along with a single hidden layer containing 20 neurons, achieved 97% performance across all metrics, while requiring a computational time of 2346.54 seconds. Although the computational time is higher compared to the previously mentioned configurations, it still offers acceptable performance.

On the other hand, some configurations, such as a learning rate of 0.05 with 200 epochs and a dropout rate of 0.1, along with two hidden layers, each containing 20 neurons, resulted in slightly lower performance, with accuracy, precision, recall, and F1-score falling in the range of 96-97%. Although the computational time was moderate at 5414.67 seconds, this combination may not be the most optimal choice for achieving the highest recall and F1-score.

The computational time varied significantly between experiments due to the influence of hyperparameter configurations. Longer training times were observed for configurations with more epochs and complex network structures. The increased number of epochs necessitates more iterations during the training process, consuming additional computational resources and time. Similarly, network structures with more hidden layers and neurons result in higher parameter counts, leading to increased computations during forward and backward passes. Consequently, configurations with more complex structures require longer training times. Conversely, configurations with fewer epochs and simpler network structures may offer shorter training times but could compromise optimal performance.

In summary, the best hyperparameter combination should be determined by a careful trade-off between performance metrics (accuracy, precision, recall, F1-score) and computational time. The configuration with a learning rate of 0.1 and epochs of 200 stands out as the most promising choice, offering the highest overall performance and computational efficiency. Nevertheless, other combinations should also be considered, as they may provide viable alternatives depending on the specific requirements and constraints of the malware detection system.

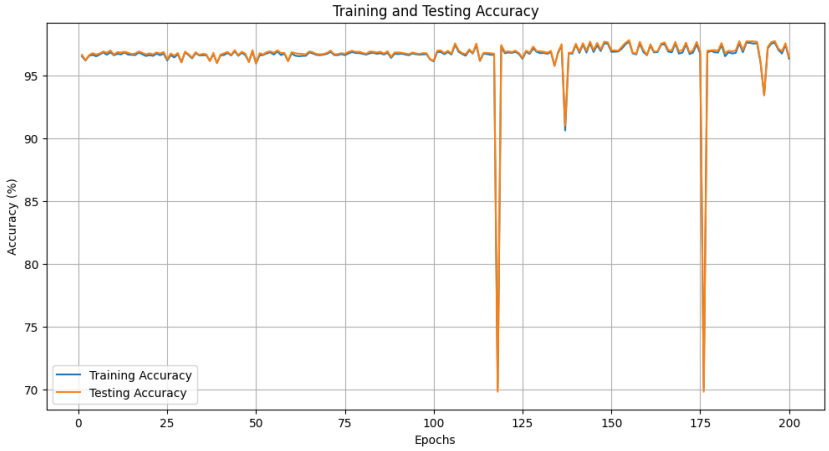

Figure 3. Comparison graph of accuracy for two different experimental settings. In condition (a), a learning rate of 0.1 was used in combination with 200 epochs, while in condition (b), a learning rate of 0.05 was employed with 200 epochs.

The implications of this finding are significant for malware detection using neural network models. The hyperparameter combination with a learning rate of 0.1 and 200 epochs demonstrates superior performance and computational efficiency. To comprehensively validate the advantages of this configuration, further research is warranted, utilizing a dataset with a greater diversity of features. Evaluating the model's performance on a more varied dataset will enhance the reliability and applicability of the findings to real-world scenarios. By conducting this additional research, the study can provide stronger support for the proposed hyperparameter configuration, leading to more confident decisions in developing high-performing and efficient malware detection systems.

4. CONCLUSION

In this research, we addressed the critical challenge of optimizing malware detection using the Back Propagation Neural Network (BPNN) in conjunction with hyperparameter tuning. With the rapid expansion of the internet and the consequent rise in cyber threats, developing a sophisticated malware detection system leveraging machine learning (ML) and deep learning (DL) techniques has become a priority to safeguard computer systems and networks.

Our study focused on fine-tuning key hyperparameters of the BPNN model, namely the dropout rate, the number of neurons in hidden layers, and the number of hidden layers, to enhance the accuracy of malware detection. To achieve this, we employed a Back Propagation Neural Network (BPNN) with dropout regularization, as it showed promise in handling complex data and classification tasks. The core of the research focused on hyperparameter optimization using GridSearchCV, a powerful technique for systematically exploring various combinations of hyperparameters to identify the optimal configuration. The experiments involved varying learning rates and epochs to understand their impact on the model's performance.

Through rigorous experimentation and analysis, we identified the hyperparameter combination with a learning rate of 0.1 and 200 epochs as the most promising choice. This configuration consistently demonstrated outstanding performance, achieving an accuracy, precision, recall, and F1-score of 98% across all metrics. Moreover, it exhibited reasonable computational efficiency, striking an effective balance between model performance and training time. The implications of our findings are significant for the field of malware detection using neural network models. The proposed hyperparameter configuration with a learning rate of 0.1 and 200 epochs exhibits superior performance and computational efficiency, making it a valuable asset in developing effective malware detection systems.

However, further research and validation on datasets with greater feature diversity are warranted to strengthen the reliability and applicability of the results in real-world scenarios. In conclusion, our research successfully optimized malware detection using the BPNN with hyperparameter tuning. By leveraging deep learning techniques and systematically fine-tuning key hyperparameters, we developed a more robust and accurate malware detection solution. The proposed hyperparameter configuration demonstrates outstanding performance while maintaining computational efficiency, contributing to the field of cybersecurity and fortifying the security of computer systems and networks against malicious threats.

Future research in this area should focus on validating the proposed hyperparameter configuration on diverse datasets and exploring the integration of other advanced ML and DL techniques to further enhance the model's capabilities. Despite the promising results, our research acknowledges the limitations inherent in the

dataset used for experimentation. To address this concern, utilizing more diverse and comprehensive datasets can ensure the model's robustness. Additionally, other hyperparameter optimization techniques beyond GridSearchCV should be explored to fine-tune the model's performance even further.

By adopting the insights from this study, the field of cybersecurity can benefit from more reliable and efficient malware detection systems, ultimately fortifying the security of computer systems and networks against the ever-evolving landscape of cyber threats.

REFERENCES

[1] "Internet and social media users in the world 2023 | Statista." https://www.statista.com/statistics/617136/digital-population-worldwide/ (accessed Jul. 01, 2023).

[2] D. K. Bhattacharyya and J. K. Kalita, *Network Anomaly Detection: ML Perspective*. 2014.

[3] L. Li *et al.*, "On Locating Malicious Code in Piggybacked Android Apps," *J. Comput. Sci. Technol.*, vol. 32, no. 6, pp. 1108–1124, 2017, doi: 10.1007/s11390-017-1786-z.

[4] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust Intelligent Malware Detection Using Deep Learning," *IEEE Access*, vol. 7, no. c, pp. 46717–46738, 2019, doi: 10.1109/ACCESS.2019.2906934.

[5] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 1225–1234, 2016, doi: 10.1145/2939672.2939753.

[6] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," *Proc. 2015 10th Int. Conf. Malicious Unwanted Softw.*, 2016.

[7] F. Al Huda, W. F. Mahmudy, and H. Tolle, "Android malware detection using backpropagation neural network," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 4, no. 1, pp. 240–244, 2016, doi: 10.11591/ijeecs.v4.i1.pp240-244.

[8] Z.-P. Pan, C. Feng, and C.-J. Tang, "Malware Classification Based on the Behavior Analysis and Back Propagation Neural Network," *ITM Web Conf.*, vol. 7, p. 02001, 2016, doi: 10.1051/itmconf/20160702001.

[9] A. Makandar and A. Patrot, "Malware analysis and classification using Artificial Neural Network," *Int. Conf. Trends Autom. Commun. Comput. Technol. I-TACT 2015*, 2016, doi: 10.1109/ITACT.2015.7492653.

[10] B. B. Rad, M. K. H. Nejad, and M. Shahpasand, "Malware classification and detection using artificial neural network," *J. Eng. Sci. Technol.*, vol. 13, no. Special Issue on ICCSIT 2018, pp. 14–23, 2018.

[11] D. Jayanth, "Malware-Detection-in-PE-files-using-Machine-Learning," 2021. https://github.com/DasariJayanth/Malware-Detection-in-PE-files-using-Machine-Learning/tree/master/PE_Header(exe%2C dll files)

[12] G. Ciaburro, V Kishore Ayyadevara, and A. Perrier, *Hands-On Machine Learning on Google Cloud Platform*, vol. 6, no. August. Packt Publishing Ltd, 2018.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 6, pp. 1929–1958, 2014, doi: 10.1016/0010-4361(73)90803-3.

[14] R. H. R. Hahnioser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex- inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, 2000, doi: 10.1038/35016072.

[15] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. doi: 10.1108/978-1-83909-694-520211010.

[16] F. Chollet, *Deep Learning with Python*, First Edit. Manning Publications, 2017. doi: 10.1007/978-1-4842-5364-9.

[17] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022, doi: 10.1016/j.neucom.2022.06.111.

[18] M. Liu, D. Yao, Z. Liu, J. Guo, and J. Chen, "An Improved Adam Optimization Algorithm Combining Adaptive Coefficients and Composite Gradients Based on Randomized Block Coordinate Descent," *Comput. Intell. Neurosci.*, vol. 2023, pp. 1–14, 2023, doi: 10.1155/2023/4765891.

[19] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent.*, pp. 1–15, 2015.

[20] S. Ruder, "An overview of gradient descent optimization algorithms," pp. 1–14, 2016, [Online]. Available: http://arxiv.org/abs/1609.04747

[21] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journa lof Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[22] M. C. Dickson, A. S. Bosman, and K. M. Malan, "Hybridised Loss Functions for Improved Neural Network Generalisation," *Pan-African Artif. Intell. Smart Syst. PAAISS 2021. Lect. Notes Inst. Comput. Sci. Soc. Informatics Telecommun. Eng.*, vol. 405, pp. 169–181, 2022, doi: 10.1007/978-3-030-93314-2_11.

[23] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: 10.1016/j.neucom.2020.07.061.

[24] E. Elgeldawi, A. Sayed, A. R. Galal, and A. M. Zaki, "Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis," *Informatics*, vol. 8, no. 4, pp. 1–21, 2021, doi: 10.3390/informatics8040079.

BIBLIOGRAPHY OF AUTHORS

Annisa Arrumaisha Siregar, currently an active final semester student in the Department of Electrical Engineering, specializing in the Telecommunication Engineering Study Program, at State Polytechnic of Sriwijaya. The author is interested in the fields of artificial intelligence and cyber security.

Sopian Soim, currently a lecturer in the Department of Electrical Engineering, specializing in the Telecommunication Engineering Study Program, at State Polytechnic of Sriwijaya. The author completed a Bachelor's Degree in Electrical Engineering at Universitas Sriwijaya in 1997 and then pursued a Master's Degree in Telecommunication Engineering at Institut Teknologi Sepuluh Nopember, graduating in 2007. The author's research interests lie in the areas of wireless communication, data science, artificial intelligence, and cyber security.

Mohammad Fadhli, currently a lecturer in the Department of Electrical Engineering, specializing in the Telecommunication Engineering Study Program, at State Polytechnic of Sriwijaya. The author completed a Bachelor's Degree in Electronics Engineering at Universitas Negeri Padang in 2013 and then pursued a Master's Degree in Multimedia Telecommunication at Institut Teknologi Sepuluh Nopember, graduating in 2015. The author's research interests lie in the areas of wireless communication, wireless sensor networks, and the Internet of Things.