Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 8220453, 19 pages
https://doi.org/10.1155/2022/8220453

Research Article

A Hybrid Improved Neural Networks Algorithm Based on L2 and Dropout Regularization

Xiaoyun Xie,[1,2] **Ming Xie**,[1] **Ata Jahangir Moshayedi**,[1]
and Mohammad Hadi Noori Skandari[3]

[1]*School of Information Engineering, Jiangxi University of Science and Technology, No. 86, Hongqi Ave, Ganzhou 341000, Jiangxi, China*
[2]*School of Electronic Information Engineering, Gannan University of Science and Technology, Ganzhou 341000, China*
[3]*Faculty of Mathematical Sciences, Shahrood University of Technology, Shahrood, Iran*

Correspondence should be addressed to Ata Jahangir Moshayedi; ajm@jxust.edu.cn

Received 17 June 2022; Revised 20 October 2022; Accepted 27 October 2022; Published 3 November 2022

Academic Editor: Zuowei Cai

Small samples are prone to overfitting in the neural network training process. This paper proposes an optimization approach based on L2 and dropout regularization called a hybrid improved neural network algorithm to overcome this issue. The proposed model was evaluated based on the Modified National Institute of Standards and Technology (MNIST, grayscale-$28 \times 28 \times 1$) and Canadian Institute for Advanced Research 10 (CIFAR10, RGB - $32 \times 32 \times 3$) as the training data sets and data applied to the LeNet-5 and Autoencoder neural network architectures. The evaluation is conducted based on cross-validation; the result of the model prediction is used as the final measure to evaluate the quality of the model. The results show that the proposed hybrid algorithm can perform more effectively, avoid overfitting, improve the accuracy of network model prediction in classification tasks, and reduce the reconstruction error in the unsupervised domain. In addition, employing the proposed algorithm without increasing the time complexity can reduce the effect of noisy data and bias and improve the training time of neural network models. Quantitative and qualitative experimental results show that the accuracy of using the proposed algorithm in this paper with the MNIST test set has an improvement of 2.3% and 0.9% compared to L2 regularization and dropout regularization, respectively, and based on the CIFAR10 data set, the accuracy improvement of 0.92% compared with L2 regularization and 1.31% concerning dropout regularization. The reconstruction error of using the proposed algorithm in this paper with the MNIST data set has an improvement of 0.00174 and 0.00398 compared to L2 regularization and dropout regularization, respectively, and based on the CIFAR10 data set, the accuracy improvement of 0.00078 compared with L2 regularization and 0.00174 concerning dropout regularization.

1. Introduction

Generally, convolutional neural networks are used in the image classification field to extract feature information in images, and fully connected layers are used to build classifiers. However, due to a large number of weight parameters in the fully connected layer, it is easy to cause the problem of overfitting in the case of small samples, which makes neural networks overparameterized. The review of available research papers shows that to solve the overfitting of neural networks during training, Hinton et al. [1] first proposed a dropout regularization method to prevent overfitting. This method introduces dropout during network model training, which can randomly suppress the activation values of some neurons. Therefore, the neural network structure obtained by each training iteration is different and can be regarded as a subset of different neurons. The working principle of dropout can be understood from the perspective of model averaging. In the final prediction, all neurons are retained to participate in the test, and the average approximation of all trained models is obtained to prevent overfitting and improve model prediction accuracy. The remarkable effect of the dropout regularization technique is widely used in various types of neural networks. With the same target, Zhou and Luo [2] proposed a method to prevent overfitting, which selects the probability of node deletion according to the size of the activation value. The network deletes the node with a lower activation value with a higher probability of retaining more nodes with a higher activation value and enhances the model's feature extraction ability. During testing, all deleted nodes are retained, and all training parameters are restored, which achieves the purpose of combining multiple networks and embodies the idea of model averaging. Zhong et al. [3] alleviated the overfitting of the model by proposing a multi-scale fusion method to optimize dropout. The method uses the genetic algorithm to find the optimal scale, then further updates the parameters in the network according to the optimal scale to obtain prediction submodels, and finally fuses these submodels into the final prediction model with a certain weight. This is the actual idea of model averaging. Ghiasi et al. [4] proposed the DropBlock method to alleviate model overfitting. In this method, during the model's training process, the units in the continuous area of the feature map are discarded together, and the number of discarded units gradually increases as the number of iterations increases. This can improve the accuracy and robustness of network model predictions. Cheng et al. [5] proposed an improved model averaging method to prevent overfitting. The early dropout was applied to the fully connected layer, but the training phase of this method introduced dropout in the pooling layer, which made the unit value of the pooling layer sparse. In the test phase, the probability of the unit value selected by the pooling layer dropout during training is multiplied by the probability of each unit value in the pooling area as a double probability. The sparsity effect of the stage pooling layer dropout can be better reflected on the test stage pooling layer. Inspired by DropBlock, Pham and Le [6] proposed the method of auto dropout. Unlike DropBlock, this method automatically learns the dropout mode on each channel and layer during the model training process, automates the design process of the dropout mode, and divides the original continuous dropout area into small areas. This causes to have better model generalization performance [7]. Gomez et al. [8] proposed a new method (targeted dropout) to sparse the network. The method is based on dropout, in which the network is sparsed by randomly discarding some neurons. While targeted dropout is a purposeful sparse network, the idea is to rank weights or units according to an approximate measure of importance (such as size) and then apply dropout to those sets of units that are considered least useful to achieve a better effect. Many researchers studied the problem of preventing overfitting and have made great contributions.

In addition to the methods to prevent overfitting mentioned in the above literature, the L2 regularization term can also be added to the loss function. The trainable weight parameters can be attenuated (weight decay) to reduce the dependence on a certain feature. Improve the model's generalization ability, which can prevent overfitting [9]. Start with a small sample problem [10]; in order to avoid overfitting during model training to the greatest extent and improve the generalization ability of the network model, it is particularly important to have a powerful regularization technique [7]. As the above survey shows to improve the model's generalization of parameters such as performance, accuracy, and training time, the impact of noisy data and complexity reduction of the neural network on model training should be considered more and investigated. This research paper proposes an improved hybrid algorithm based on L2 and dropout regularization concerning the mentioned parameters.

In the hybrid proposed algorithm, the dropout method is initially used to reduce the number of updatable parameters, which can speed up the training of the network model. Then, L2 constraints are hired on these parameters to attenuate the weight parameters, reduce the dependence on a certain feature, and enhance the robustness of feature selection. Finally, the loss function is reconstructed to optimize the update process of the parameters in the network model training to prevent overfitting. For the convenience of subsequent description, this paper calls it the hybrid algorithm. The hybrid algorithm can be applied to various classification tasks requiring fully connected layers. The present study would contribute to the existing knowledge in several different ways, as follows:

(1) This study proposes a new hybrid algorithm to solve the overfitting problem caused by the small number of samples in the neural network training process. As known, the large dataset required for training a reliable model is one of the crucial obstacles to implement the neural network. However, such datasets are not easily accessible because of the required time, cost, and energy for preparing them. Therefore, an alternative approach would be making the best use of the data in hand rather than straightforwardly increasing the costly dataset size. With this in mind, the results of this study would be used in different disciplines, including engineering, medicine, and geosciences, where neural networks might be potentially used.

(2) The proposed algorithm acts on the fully connected neural network layer and concentrates on improving prediction accuracy and reducing the reconstruction error, which can effectively reduce data noise and error. This is very crucial considering the approaches that researchers have put forth recently where they translate the developed networks into user-friendly apps for the convenient use of fellow researchers [11].

(3) The proposed algorithm was compared with two data sets of the MNIST and CIFAR10 with different dimensions and image types. Considering the wide usage of these two well-known datasets by different researchers, the comparison results prove the efficiency of the introduced method.

(4) The proposed algorithm with the supervised (LeNet-5) and unsupervised RAE neural network architecture

algorithms are compared to investigate the prediction accuracy reconstruction error, respectively.

In general, the authors believe that based on the conducted assessments of the obtained results, as mentioned in the upcoming sections and the observed satisfying performance, the introduced method would efficiently contribute to solving various problems in different fields of science.

The paper is set as follows: first, the two algorithms of L2 and dropout and the proposed hybrid algorithm are described in section 2; then, in section 3, the process of hyperparameter estimation with the LeNet-5 and Rough Auto Encoder (RAE), based on two datasets (MNIST and CIFAR10) and each algorithm's performance (without regularization, L2 regularization, dropout regularization, and hybrid techniques) were experimented and analyzed. In section 4, the obtained results are compared and shown. Finally, the paper is concluded in section 5.

2. Algorithm Introduction

This section provides detailed descriptions of L2 regularization [12] and dropout regularization [13]. The core idea of the algorithm is described according to the mathematical theory behind it, and the pseudocode is given later to illustrate the working principle of the algorithm. Finally, the improved algorithm is described in detail in the third module, and the pseudocode is also given.

2.1. L2 Algorithm. The L2 regularization works with the principle of adding a regularization term (also known as a penalty term) to the loss function and participating in the model's training process. The L2 constraint [14, 15] imposes a greater penalty on a larger weight. The larger the weight parameter value, the greater the attenuation, thereby reducing the dependence on a certain feature. Therefore, the absolute value of the weight parameter in the network layer tends to decrease, and there will be no particularly large value to avoid the problem of overfitting and improve the model's ability. The general mathematical expression of the loss function of the neural network that introduces the regularization term during the training process of the model is shown in the following:

$$J(\theta) = L(\theta) + \sum_{l=1}^{k}\left(\lambda \sum_{i=0}^{m}\sum_{j=0}^{n}\left(\theta_{ij}^{l}\right)^{2}\right) 0 \le i \le m, \quad 0 \le j \le n 1 \le l \le k. \tag{1}$$

As (1) shows, $J(\theta)$ means the loss function after adding the $L2$ regularization term to the loss function $L(\theta)$, where the parameter λ (its value is between 0 and 1) is used to control the penalty on the weight parameter, the parameter λ does not participate in the training of the network model and is a hyperparameter, and θ_{ij} represents the weight parameter in the network. k means that the network has k fully connected layers. n means that each layer has n neurons.

In the process of error backpropagation (BP) [16], the calculation formula for the partial derivative of a weight parameter in the network model is as follows:

$$\frac{\partial J}{\partial \theta_{ij} l} = \frac{\partial L}{\partial \theta_{ij} l} + 2\lambda\theta_{ij} 0 \le i \le m, \quad 0 \le j \le n \lambda \in (0,1). \tag{2}$$

Taking the weight parameters of the $l-th$ layer as an example, it can be concluded that the gradient vector of the weight parameter is in a certain layer of the network model.

$$\begin{aligned}
&\left(\frac{\partial J}{\partial \theta_{00}^{l}}, \frac{\partial J}{\partial \theta_{01}^{l}}, \frac{\partial J}{\partial \theta_{02}^{l}}, \ldots, \frac{\partial J}{\partial \theta_{10}^{l}}, \frac{\partial J}{\partial \theta_{11}^{l}}, \frac{\partial J}{\partial \theta_{12}^{l}}, \ldots, \frac{\partial J}{\partial \theta_{ij}^{l}}\right), \\
&= \begin{pmatrix} \frac{\partial L}{\partial \theta_{00}^{l}} + 2\lambda\theta_{00}^{l}, \frac{\partial L}{\partial \theta_{01}^{l}} + 2\lambda\theta_{01}^{l}, \frac{\partial L}{\partial \theta_{02}^{l}} + 2\lambda\theta_{02}^{l}, \ldots, 0 \le i \le m, \quad 0 \le j \le n 1 \le l \le k, \\ \frac{\partial L}{\partial \theta_{10}^{l}} + 2\lambda\theta_{10}^{l}, \frac{\partial L}{\partial \theta_{11}^{l}} + 2\lambda\theta_{11}^{l}, \frac{\partial L}{\partial \theta_{02}^{l}} + 2\lambda\theta_{12}^{l}, \ldots, \frac{\partial L}{\partial \theta_{ij}^{l}} + 2\lambda\theta_{ij}^{l} \end{pmatrix}.
\end{aligned} \tag{3}$$

The calculation of a parameter update in the network model is represented as in the following equation:

$$\begin{aligned}
\theta_{ij}^{l} &= \theta_{ij}^{l} - lr\frac{\partial J}{\partial \theta_{ij}^{l}}, \\
&= \theta_{ij}^{l} - lr\left(\frac{\partial L}{\partial \theta_{ij}^{l}} + 2\lambda\theta_{ij}^{l}\right) 0 \le i \le m, \quad 0 \le j \le n 1 \le l \le k.
\end{aligned} \tag{4}$$

The parameter lr in (4) is the step size of the parameter update when using the gradient descent algorithm, also called the learning rate. In summary, the update of all weight parameters of a certain layer in the network model can be obtained, as shown in (5), which is expressed in the vector form.

$$\theta^l = \left[\left(\theta_{00}^l - lr\frac{\partial J}{\partial \theta_{00}^l}, \theta_{01}^l - lr\frac{\partial J}{\partial \theta_{01}^l}, \theta_{02}^l - lr\frac{\partial J}{\partial \theta_{02}^l}, \ldots, \theta_{10}^l - lr\frac{\partial J}{\partial \theta_{10}^l}, \theta_{11}^l - lr\frac{\partial J}{\partial \theta_{11}^l}, \theta_{12}^l - lr\frac{\partial J}{\partial \theta_{12}^l}, \ldots, \theta_{ij}^l - lr\frac{\partial J}{\partial \theta_{ij}^l}\right)\right]$$

$$= \begin{bmatrix} \theta_{00}^l - lr\left(\frac{\partial L}{\partial \theta_{00}^l} + 2\lambda\theta_{00}^l\right), \theta_{01}^l - lr\left(\frac{\partial L}{\partial \theta_{01}^l} + 2\lambda\theta_{01}^l\right), \theta_{02}^l - lr\left(\frac{\partial L}{\partial \theta_{02}^l} + 2\lambda\theta_{02}^l\right), \ldots, \\ \theta_{10}^l - lr\left(\frac{\partial L}{\partial \theta_{10}^l} + 2\lambda\theta_{10}^l\right), \theta_{11}^l - lr\left(\frac{\partial L}{\partial \theta_{11}^l} + 2\lambda\theta_{11}^l\right), \theta_{12}^l - lr\left(\frac{\partial L}{\partial \theta_{12}^l} + 2\lambda\theta_{12}^l\right), \ldots, \\ \theta_{12}^l - lr\left(\frac{\partial L}{\partial \theta_{ij}^l} + 2\lambda\theta_{ij}^l\right) \end{bmatrix}. \tag{5}$$

It can be seen from (5) mathematical formulas, in the training process of the model, the L2 regularization term $R(\theta)$ is introduced based on the loss function $L(\theta)$, and the regularization term can control the weight parameters (weight decay) to prevent overfitting, reduce the dependence on all features, improve the generalization ability of the model, and improve the accuracy of the model prediction. Using L2 regularization can reduce the dependence on some weight parameters to prevent overfitting, but the network structure cannot be optimized, and the neural network with a complex network and many weight parameters is helpless. The pseudocode description of the L2 regularization is described as Algorithm 1.

2.2. Dropout Algorithm. The dropout [16] regularization method was first proposed by Hinton et al. [1] to solve the overfitting problem of neural networks during training. Dropout works on the idea that it is added to the neural network in the training process, and some neurons are inhibited by randomly generating a probability vector of 0 and 1 (p_vector) to act on the activation unit [17]. The network structure of each training iteration is different, while all neurons are preserved in the test. In each iterative training process, the output of the activation value of the inhibited neuron is zero, and the connection weight parameter with the inhibited neuron does not participate in the update process. Each neuron in the neural network is inhibited with a certain probability. This mechanism of inhibiting neurons can optimize the structure of the network model and reduce the weight parameters that can be updated during the model's training process. Because of this mechanism, the network produces a different network structure each time after it is trained. Therefore, each network structure can be regarded as a subset of different neurons. If it is iterated n times, there are n different network model structures that will be randomly generated, and the n different network models will jointly determine the final prediction result of the model. This technique is also called model averaging [18, 19]. In this way, the overfitting problem is prevented, which is how the dropout regularization technique works. The core formula of the dropout algorithm is shown in the following formulas (6)–(9).

$$p\_\text{vector}^{\text{layer}} \sim \text{Bernoulli}(p), \tag{6}$$

$$\tilde{a}^{\text{layer}} = p\_\text{vector}^{\text{layer}} * a^{\text{layer}}, \tag{7}$$

$$z^{\text{layer}+1} = \left[\left(\left(\tilde{a}^{\text{layer}}\right).T\right).\text{dot}\left(\theta^{\text{layer}+1}\right)\right] + b^{\text{layer}+1}, \tag{8}$$

$$a^{\text{layer}+1} = RELU\left(z^{\text{layer}+1}\right), \tag{9}$$

In equation (6), the function Bernoulli (p) is to generate a probability vector between 0 and 1. (7) indicates that the probability vector is applied to the activation unit of the layer-th to obtain a new activation value $\tilde{a}^{\text{layer}}$and (9) uses the RELU activation [20] function to enhance the nonlinear expression ability. The dropout algorithm can optimize the neural network's structure, suppress some neurons' activation values, and reduces the number of network weight parameters that can be updated during the training process. However, the dependence on some features cannot be alleviated during each iteration, and only some activated units are suppressed. In addition, due to the huge number of parameters, the effect of simply using dropout is not obvious for large networks. Algorithm 2 shows the pseudocode description of the dropout algorithm.

2.3. Hybrid Algorithm. The hybrid algorithm is an improved algorithm derived from L2 regularization and dropout regularization. While using L2 regularization for weight decay, dropout is introduced to optimize the network structure in the model training process. Using dropout can inhibit some neurons to reduce the dependence between neurons, thinning the connections between neurons and not relying too much on certain features to prevent overfitting better as the structural level optimization. For example, a neuron responsible for a key feature may fail during an iteration due to a neuron being inhibited. Therefore, the network must find other important features that improve the network model's generalization ability from another

```
Initialization:
        Initialize the neural network's weight parameters$\theta$,
        regularization parameter$\lambda$,
        regularization term$R(\theta) = 0$,
        and parameter learning rate$lr$.
# Iterate over the parameters of each layer,
        $l$ represents a certain layer,
        $k$ represents all layers.
For $l$ in $k-1$:
    # Get the number of neurons in this layer. $l$ _nums_of_ neurons = len ($l$)
    # Get the number of neurons in the next layer. $l$ _next_nums_of_ neurons = len ($l+1$)
    for$i$in range ($l$ _nums_of_ neurons):
        for$j$in range ($l$ _next_nums_of_ neurons):
          $R(\theta) = R(\theta) + (\theta^l_{ij})^2$
        End for.
    End for.
End for.
# Get the regularization term. $R = \lambda \times R(\theta)$
Suppose $L(\theta)$ is our loss function, then $J(\theta) = L(\theta) + R(\theta)$: final cost function.
Repeat do:
        for epoch in epochs:
            # Parameter update in units of mini_batch.
                    for mini_batch in range(mini_batches):
                      # repeat do, parameter update process.
                      $\Delta\theta^l_{ij} = \partial L/\partial\theta^l_{ij} + \partial R/\partial\theta^l_{ij} = \partial L/\partial\theta^l_{ij} + 2\lambda\theta^l_{ij}$
                      $\theta^l_{ij} = \theta^l_{ij} - lr \times \Delta\theta^l_{ij} = \theta^l_{ij} - lr \times (\partial L/\partial\theta^l_{ij} + 2\lambda\theta^l_{ij})$
                    End for
End for
```

Algorithm 1: L2 regularization algorithm.

```
Input: activation value, note as $a$.
Initialization:
        Initialize the neural network's weight parameters $\theta$,
        bias parameters $b$,
        and neuron drop rate $p$.
for epoch in epochs:
# Generate probabilistic vectors for each layer of neurons used to simulate neuronal inactivation.
    for layer in layers
        # According to drop rate $p$ generate $p$_vector$^{\text{layer}}$.
        $p$_vector$^{\text{layer}} \sim$ Bernoulli ($p$).
    End for.
    x ← random min-batch from Dataset $D$.
    Repeat do:
      for $x$ in D: # Use $x$ to represent mini_batch, Use $D$ to represent all mini_batches.
            for layer in layers-1:
                $\tilde{a}^{\text{layer}} = p$_vector$^{\text{layer}} * a^{\text{layer}}$.
                $z^{\text{layer}+1} = [((\tilde{a}^{\text{layer}}).T).\text{dot}(\theta^{\text{layer}+1})] + b^{\text{layer}+1}$
            if the layer is not the output layer:
                    $a^{\text{layer}+1} = RELU(z^{\text{layer}+1})$.
            else:
                $a^{\text{layer}+1} = \text{Softmax}(z^{\text{layer}+1})$.
                # $a^{\text{layer}+1}$ note as $\hat{y}$
              $\hat{y} = \text{Softmax}(z^{\text{layer}+1})$
            End for.
      End for.
    Pass: parameter update using gradient descent.
End for.
```

Algorithm 2: Dropout regularization algorithm.

perspective. The use of L2 regularization can prevent overfitting by attenuating the updatable parameters, which is an optimization at the parameter level. Therefore, the hybrid algorithm is proposed to solve the overfitting problem better so that the neural network can learn robust and concurrent features, improve the accuracy of network model prediction, and improve the stability of the model training process. A new and improved algorithm for preventing overfitting is proposed to combine optimization at the parameter and network structure levels. The general regularization term is denoted as $R(\theta)$ and defined as follows:

$$R(\theta) = \sum_{l=1}^{k}\left(\lambda\sum_{i=0}^{m}\sum_{j=0}^{n}\left(\theta_{ij}^{l}\right)^{2}\right)0 \le i \le m, \quad 0 \le j \le n1 \le l \le k. \tag{10}$$

The parameter λ (its value is between 0 and 1) is used to control the penalty on the weight parameter; the parameter λ does not participate in the training of the network model, and it is a hyperparameter, θ_{ij} represents the weight parameter in the network. In (10), k means that the network has k fully connected layers and n shows that each layer has n neurons. We know that the last step of the network model is to calculate the error between the predicted value $\hat{y}$ and the true value y. The $\hat{y}$ is defined as follows:

$$\hat{y} = \text{Softmax}\left(\left[\left(\left(a^{k-1}\right).T\right).\text{dot}\left(\theta^{k}\right)\right] + b^{k}\right)\hat{y} \in [0,1]. \tag{11}$$

In Equation (11), a^{k-1} represents the activation value vector of the $(k-1)th$ layer, θ^{k} represents the weight parameter of the $(k)th$ layer, and b^{k} represents the bias vector composed of all neurons in the $(k)th$ layer. Finally, use the Softmax function to get the output value $\hat{y}$ of the model. The loss function used in this paper is the multi-class cross-entropy loss function, denoted as $L(\hat{y}, y)$, then

$$L(\hat{y}, y) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\right], \tag{12}$$

where, n represents the number of samples in each training iteration, and it is used to find the average loss for each batch size, the loss function J of the network model is shown in the following:

$$\begin{aligned} J &= L(\hat{y}, y) + R(\theta), \\ &= -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\right] + \sum_{l=1}^{k}\left(\lambda\sum_{i=0}^{m}\sum_{j=0}^{n}\left(\theta_{ij}^{l}\right)^{2}\right). \end{aligned} \tag{13}$$

In each epoch, a set of probability vectors p_vector containing only 0 and 1 are randomly generated to suppress some neurons.

$$p\_\text{vector}^{l} \sim \text{Bernoulli}(p) 1 \le l \le k. \tag{14}$$

In equation (14), p is a hyperparameter drop rate. The loss function of the final network model using the hybrid algorithm is represented in the following:

$$\begin{aligned} J = &-\frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\right] \\ &+ \sum_{l=1}^{k}\lambda\sum_{i=0}^{m}\sum_{j=0}^{n}\left(p\_\text{vector}_{i}^{l} * \theta_{ij}^{l}\right)^{2} 0 \le i \le m, \quad 0 \le j \le n1 \le l \le k, \end{aligned} \tag{15}$$

where, $p\_\text{vector}_{i}^{l}$ represents the element in the probability vector corresponding to the neuron in the layer l, acting on the weight parameter connected to the neuron. The core formula involved in forwarding propagation is shown in the following equations:

$$z^{l+1} = \left[\left(\left(a^{l} * p\_\text{vector}^{l}\right).T\right)\right].\text{dot}\left(\theta^{l+1}\right) + b^{l+1}, \tag{16}$$

$$a^{l+1} = RELU\left(z^{l+1}\right) 0 \le l \le k-2, \tag{17}$$

$$\hat{y} = a^{l+1} = \text{Softmax}\left(z^{l+1}\right) l = k-1. \tag{18}$$

In (16), the probability vector acts on the activation unit of a certain layer. The activation value is subjected to a dot product operation to achieve partial neuron inhibition (17), and the activation function used in implementing this algorithm is rectified linear units (RELU) [21]. Based on (18), the SoftMax [21] function is used to get the final output value of the network model. Finally, the parameters are updated using the gradient descent algorithm [22] and also Adam (adaptive learning rate) as the optimizer [23]. It can be seen from the above formula that the hybrid algorithm attenuates the weight parameters in the network model through L2 regularization and, at the same time, it reduces the complexity of the network model in each iterative training process. By introducing dropout to optimize the network model's structure, the network model's generalization ability can be enhanced, and the effect of preventing overfitting can be better achieved. The pseudocode description of the hybrid algorithm is repressed in Algorithm 3.

3. Experimental Results and Analysis

The experiment section is performed on the small sample problem to investigate the overfitting phenomenon and compare the algorithm performance efficiency. The PyCharm IDE is used for coding all algorithms [23], in order to verify the effectiveness of the proposed algorithm, the Modified National Institute of Standards and Technology databases (MNIST) dataset [24] and the Canadian Institute for Advanced Research, 10 classes (CIFAR10) dataset [25, 26] with respect to open-source accessibility. The MNIST dataset contains the image of handwritten digits from 0 to 9 in 10 categories with the grayscale format and $28 \times 28 \times 1$ dimension, and the CIFAR10 dataset (collected by Alex et al.) consists of 10 categories of $32 \times 32 \times 3$ different RGB images. To use both datasets (MNIST and CIFAR10), initially, the data were randomly mixed to be randomly distributed, and then, a total of 1000 samples were selected, of which 800 samples were used for training, and 200 samples were used for testing and validation. The L2 and

```
Input: the feature map extracted by the convolutional neural network makes
        the Flatten operation is a one-dimensional feature vector; note as $a^0$.
Initialization: initialize the neural network's weight parameters $\theta$, bias parameters $b$, regularization parameter $\lambda$, and neuron drop
rate $p$.
for epoch in epochs:
        # Generate probabilistic vectors for each layer of neurons
          used to simulate neuronal inactivation.
        for layer in layers:
          # According to drop rate $p$ generate $p\_\text{vector}^{\text{layer}}$.
          $p\_\text{vector}^{\text{layer}} \sim \text{Bernoulli}(p)$.
        End for.
        X ← random min-batch from Dataset $D$.
        Repeat do:
        for $X$ in D: # Use $X$ to represent mini_batch, and use $D$ to represent all mini_batches.
            for layer in layers-1:
              if$a^{\text{layer}+1}$ is the layer of output; note as $\hat{y}$:
                   $\hat{y} = \text{Softmax}(a^{\text{layer}}\theta^{\text{layer}+1} + b^{\text{layer}+1})$
                     Loss = Cross Entropy $(\hat{y}, y)$.
              else:
                   $z^{\text{layer}+1} = (a^{\text{layer}} \otimes p\_\text{vector}^{\text{layer}}) \cdot \theta^{\text{layer}+1} + b^{\text{layer}+1}$.
                     $a^{\text{layer}+1} = RELU(z^{\text{layer}+1})$.
            End for.
            # layer-th consists of $n$ neurons
            #Turn $p\_\text{vector}_{n\times 1}^{\text{layer}}$ into $p\_\text{vector}_{n\times 1}^{\text{layer}}$
              $p\_\text{vector}_{n\times 1}^{\text{layer}} \leftarrow p\_\text{vector}_{n\times 1}^{\text{layer}}$
            #L2 regularization term note as $\mathbf{R}(\theta)$.
            $R(\theta) = \lambda\sum_{\text{layer}}^{\text{layers}}((p\_\text{vector}^{\text{layer}}.\text{dot}(\theta^{\text{layer}})))^2$.
            $J = \text{Cross Entropy}(\hat{y}, y) + \lambda\sum_{\text{layers}}^{\text{layers}}((p\_\text{vector}^{\text{layers}}.\text{dot}(\theta^{\text{layers}})))^2$.
        End for.
        Pass : update parameters by backpropagation.
End for.
```

Algorithm 3: Hybrid algorithm.

dropout and the proposed hybrid algorithm mainly work on the fully connected layer. Then, the LeNet-5 neural network [27] with the convolutional and fully connected layers and rough auto encoder (RAE) [28] neural network fully connected layers in their architecture were selected to estimate the hyperparameter [29]. Afterwards, in LeNet-5 and rough auto encoder (RAE) neural network, the 5-fold cross-validation method is used to evaluate the performance of each model (L2 and dropout and the proposed hybrid algorithm), and finally, the average value is taken as the standard to measure the model's performance [30].

The following experiments in 4 steps for each data set are set up for comparison and verification to ensure comparability between experiments. The one-factor variable control method is used. The following experiments are set up in four steps to compare and verify each data set and ensure comparability between experiments. In this experiment, the one-factor variable control method is used in each test, the accuracy of the training set (average accuracy), the accuracy of the validation set (average validation accuracy), the loss value of the training set (average loss), and the loss value of the validation set (average validation loss) are calculated and compared. Tests on both datasets include Step 1, test without using any overfitting prevention method and regularization; Step 2, use the L2 regularization; Step 3, test based on the dropout regularization method and investigate the effect of using the dropout in this network model that suppresses some neurons; Step 4, use the hybrid algorithm to conduct experiments and observe the changes in the model during the training process which is shown as in the following:

3.1. Experiment on the MNIST Dataset.

3.1. Experiment on the MNIST Dataset. As mentioned before, initially, the MNIST data set, with the help of LeNet-5 and RAE neural network used to determine the appropriate hyperparameters as described in the following:

3.1.1. LeNet-5 Network Architecture vs. MNIST Dataset. Since the dimensions of the MNIST dataset are $28 \times 28 \times 1$, and the input to the LeNet-5 network is 32×32, the image data need to be resized to $32 \times 32 \times 3$ to be applied to the LeNet-5 network for training. Therefore, the outermost edges of the image data with two zero layers are considered [31].

(1) LeNet-5 hyperparameters calculation. Based on the LeNet-5 network architecture, experiments are carried out using the MNIST data set. The experimental results determine the hyperparameters through the 5-fold cross-validation method, and the average value is finally taken as a suitable hyperparameter. The neural network's performance with different values of the hyperparameter learning rate and

hyperparameter weight_decay is shown in Table 1, and drop rate (p) (in Table 2, respectively. In Table 1, the average parameters accuracy of train data (average train acc), average train loss, average value accuracy (average val acc), and average value loss (average val loss) for both cases of learning rate and weight decay based on LeNet-5 are presented. It should be mentioned that the ideal value for the hyperparameter values should be selected based on the best performance for average train accuracy (AT_ACC) and average value accuracy (Av_Acc) near the 1 and the average train loss (AT_Loss), as well as average val loss (AV_ Loss), should have a value close to zero that should be chosen.

Table 1 indicates the learning rate and weight decay values from 0.0001 to 0.01 with the step between 0.001 and 0.0001, respectively, for the LeNet-5 neural network architecture. In order to select better hyperparameters, the first three top-performing hyperparameters (with respect to AT_ACC and Av_Acc near 1, AT_Loss, and AV_ Loss close to zero) were selected [29], and the average was taken as the final hyperparameter. As shown, the learning rate (LeNet-5) for hyperparameter values of 0.002, 0.001, and 0.0006 have the best performance. Then, 0.0012 as the average value of these three hyperparameters is considered the final learning rate hyperparameter (LeNet-5). Similarly, the weight decay values of 0.0002, 0.0004, and 0.0005 with an average of 0.00037 is considered the final weight decay hyperparameter. The drop rate (p) hyperparameter from 0.1 to 0.9 value with the step of 0.1 for the LeNet-5 neural network is presented in Table 2.

As illustrated in Table 2, the 0.2, 0.3, and 0.4, with an average of 0.3 selected as the best performance for the hyperparameter drop rate (p) of the LeNet-5 neural network.

(2) LeNet-5 performance and cross-validation five-fold. According to selected hyperparameters with the LeNet-5 architecture, the performance of each model (without regularization, L2 regularization, dropout regularization, and the proposed hybrid algorithm) is verified by cross-validation five-fold [32] along with the average train accuracy, average train loss, average accuracy value, and average loss value which, are represented in Tables 3 and 4.

According to the extracted average value (Tables 3 and 4), each algorithm's result performance is shown in Figures 1–4 with four steps, without using any overfitting prevention method and regularization, L2 regularization, dropout regularization method, and hybrid algorithm as follows:

The Step 1 (without using any overfitting prevention method and regularization) results in Figure 1 show that severe overfitting occurs because the accuracy rate on the training set has reached 0.9773, which is much higher than the accuracy rate on the validation set, which is about 0.903. The error between the training and validation set accuracy is about 0.0743, indicating that the training data are perfectly fitted. The loss value of the validation set is also higher than the loss value of the training set. The loss value of the training set is 1.4843, and the loss value of the value set is 1.5597. The error between the training set loss value and the validation set loss value is about 0.0754.

As can be seen from Step 2 (L2 regularization), the results in Figure 2 show the problem of overfitting. However, the problem of overfitting in step 2 is alleviated compared with that in step 1, and the phenomenon of overfitting is alleviated to a certain extent. The accuracy on the training set is 0.9778, and the accuracy on the validation set is about 0.92. The error between the training and validation set accuracy is about 0.0578. The loss value on the training set (value 1.4831) differs from the loss value on the validation set (value 1.5407) by about the value of 0.0576.

From the Step 3 results in Figure 3 (dropout regularization method), it can be observed that the overfitting problem is greatly improved when compared with both steps 1 and 2. Although there is a gap between the accuracy rate on the training set (value 0.9631) and the accuracy rate on the validation set (value 0.934), the gap will not be as large as in steps 1 and 2. The error between the training and validation set accuracy is about 0.0291. There is still a little overfitting, and it can be seen from the training set loss (value 1.5027) and the validation set loss (value 1.5276) that overfitting is gradually highlighted; the loss value on the training set differs from the loss value on the validation set by about 0.249, which shows that as the number of iterations increases, overfitting becomes more and more serious.

It can be seen from the Step 4 (hybrid algorithm) results in Figure 4 and Table 4 that the results in this step are the best among all other steps. There are very large improvements in Accuracy and Loss. The error between the training set accuracy (value 0.971) and the validation set accuracy (value 0.943) is about 0.028, and the loss value on the training set (value 1.4918) differs from the loss value on the validation set (value 1.5127) by about 0.0209. Moreover, it also has stability during the training process, which ensures the stability of the model during the retraining process and prevents the overfitting problem to a large extent.

(3) Time complexity. The cross-validation method is used to compare the time complexity performance of the tested method (without regularization, with L2 regularization, dropout regularization, and hybrid algorithm) LeNet-5 neural network, and the average time is calculated. The time complexity of the models is measured by comparing the average time spent on model training with the LeNet-5 neural network. The training time of each model on the MNIST dataset is defined in Table 5.

Table 5 shows the 5-fold cross-validation based on the LeNet-5 that represented the small average time difference with the L2 regularization as the maximum and the hybrid algorithm (26.27 seconds) as the least value. The result shows that although the proposed hybrid algorithm complicates the loss function, the time performance does not deteriorate. Compared with L2 regularization, the time performance is improved by 0.99 seconds.

3.1.2. Rough Auto Encoder (RAE) vs. MNIST Dataset.

Along with the LeNet-5, another neural network architecture named rough auto encoder (RAE) was used to compare the proposed hybrid algorithm with the same MNIST

Table 1: Hyperparameter, learning rate, and weight decay for LeNet-5. Hyperparameter values (H_v), average train accuracy (AT_ACC), average value accuracy (Av_Acc), average train loss (AT_Loss), and average val loss (AV_ Loss).

H_v	Learning rate (LeNet-5)				Weight decay (LeNet-5)			
	AT_ACC	AT_Loss	Av_Acc	AV_ loss	AT_ACC	AT_Loss	Av_Acc	AV_ loss
0.01	0.13875	2.322401	0.128	2.338074	0.10725	2.302468	0.071	2.303356
0.009	0.165	2.296151	0.137	2.32942	0.10275	2.302424	0.072	2.303542
0.008	0.25075	2.210388	0.241	2.229398	0.10425	2.302377	0.073	2.303468
0.007	0.55875	1.902463	0.526	1.932271	0.10775	2.30228	0.068	2.303894
0.006	0.565	1.89632	0.543	1.917714	0.104	2.30241	0.074	2.30361
0.005	0.903	1.557816	0.842	1.621634	0.28075	2.142584	0.239	2.152453
0.004	0.94	1.521228	0.875	1.592745	0.27775	2.144539	0.235	2.156772
0.003	0.9625	1.498606	0.909	1.557244	0.2825	2.139606	0.241	2.153521
0.002	**0.988**	**1.472952**	**0.927**	**1.535847**	0.81025	1.647499	0.732	1.716683
0.001	**0.98275**	**1.478603**	**0.926**	**1.539544**	0.984	1.480389	0.927	1.543032
0.0009	0.9675	1.493069	0.918	1.551816	0.9825	1.482059	0.935	1.532924
0.0008	0.98	1.481654	0.913	1.551816	0.98475	1.478317	0.931	1.535013
0.0007	0.978	1.483917	0.904	1.566854	0.97375	1.490902	0.907	1.559097
0.0006	**0.98225**	**1.479007**	**0.921**	**1.544858**	0.98575	1.476456	0.93	1.523315
0.0005	0.9745	1.487192	0.908	1.559124	**0.99275**	**1.46911**	**0.936**	**1.526182**
0.0004	0.93725	1.524032	0.866	1.609874	**0.9755**	**1.487383**	**0.939**	**1.523864**
0.0003	0.9255	1.536564	0.864	1.600408	0.98	1.482993	0.88	1.582227
0.0002	0.959	1.507509	0.87	1.597393	**0.9865**	**1.475653**	**0.938**	**1.522716**
0.0001	0.8755	1.595188	0.781	1.692779	0.9875	1.473937	0.921	1.537909

Table 2: Hyperparameter drop rate (p) for LeNet-5 average train accuracy (AT_ACC), average train loss (AT_Loss), average value accuracy (Av_Acc), and average train loss (AT_Loss).

p	AT_ACC	AT_Loss	Av_Acc	AT_Loss
0.1	0.98375	1.478031	0.928	1.533589
0.2	**0.9815**	**1.479785**	**0.931**	**1.529648**
0.3	**0.97325**	**1.488882**	**0.938**	**1.522759**
0.4	**0.958**	**1.503502**	**0.933**	**1.529574**
0.5	0.941	1.520582	0.930	1.521176
0.6	0.87425	1.58673	0.929	1.535103
0.7	0.7705	1.691532	0.904	1.557894
0.8	0.615	1.846899	0.872	1.599164
0.9	0.30975	2.138675	0.536	1.962181

Table 3: Average cross-validation based on five folds for the parameters average train accuracy (T_ACC), train loss (T_Loss), value accuracy (V_Acc), and train loss (T_Loss).

Cross validation	Without regularization				L2 regularization				Dropout regularization			
	T_ACC	T_Loss	V_Acc	T_Loss	T_ACC	T_Loss	V_Acc	T_Loss	T_ACC	T_Loss	V_Acc	T_Loss
One-fold	0.99125	1.4698	0.91	1.5583	0.99	1.4713	0.925	1.52536	0.965	1.497	0.93	1.5317
Two-fold	0.97125	1.4915	0.86	1.5918	0.97625	1.4869	0.905	1.5612	0.98	1.4836	0.955	1.509
Three-fold	0.98	1.4811	0.92	1.5463	0.9875	1.4737	0.915	1.5387	0.969	1.5109	0.925	1.5411
Four-fold	0.96	1.5019	0.895	1.5734	0.985	1.477	0.955	1.5065	0.9488	1.5122	0.92	1.5362
Five-fold	0.98375	1.4774	0.93	1.5289	0.95	1.5068	0.9	1.5719	0.9525	1.5098	0.94	1.5201
Average	**0.9773**	**1.4843**	**0.903**	**1.5597**	**0.9778**	**1.4831**	**0.92**	**1.5407**	**0.9631**	**1.5027**	**0.934**	**1.5276**

dataset. The results are compared to the model's performance.

(1) Rough Auto Encoder (RAE) hyperparameters calculation. In a similar way to hyperparameter extraction with LetNet5, the hyperparameter extraction test was conducted based on Rough Auto Encoder (RAE) and MNIST data-based, and finally, the average of the best three performances considers as the final hyperparameter. The RAE neural network's performance with different values of the learning rate hyperparameter, weight decay hyperparameter, and drop rate hyperparameter (p) are given in Tables 6 and 7, respectively. Table 6 shows the learning rate and weight decay hyperparameter of RAE neural network architecture (from 0.01 to 0.0001 with the step of 0.0001 and 0.0001 to 1.00E-07 with the step of 1.00E-05 and 1.00E-06) trained on the

Table 4: Average cross-validation based on five folds for the parameters average train accuracy (T_ACC), train loss (T_Loss), value accuracy (V_Acc), and train loss (T_Loss).

Cross validation	T_ACC	T_Loss	V_Acc	T_Loss
One-fold	0.98125	1.4815	0.92	1.535
Two-fold	0.96625	1.4977	0.94	1.514
Three-fold	0.95875	1.5047	0.955	1.5053
Four-fold	0.9675	1.4953	0.945	1.5086
Five-fold	0.98125	1.4796	0.955	1.5008
Average	**0.971**	**1.4918**	**0.943**	**1.5127**

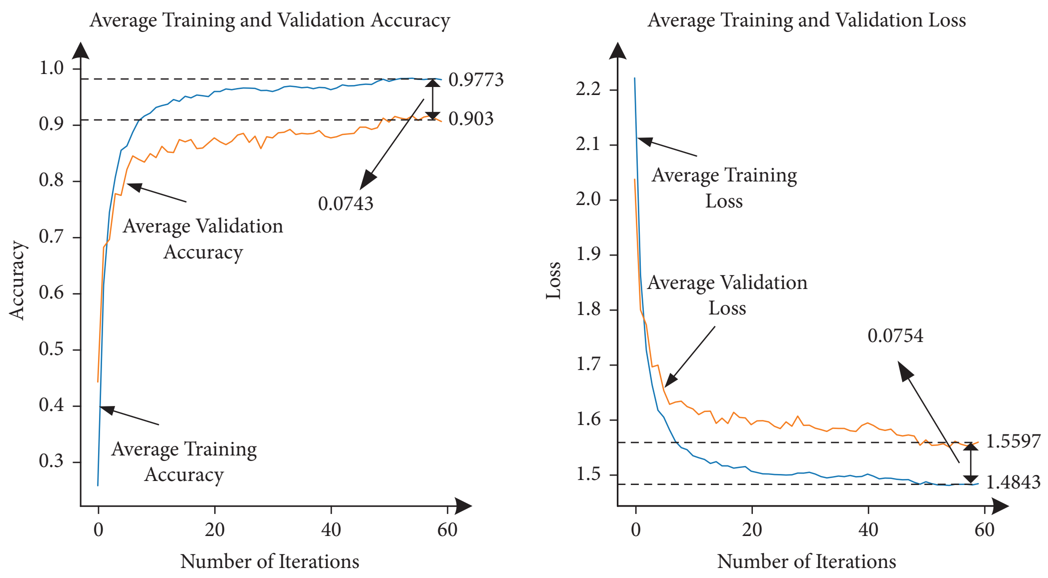

Figure 1: MNIST training results without regularization.

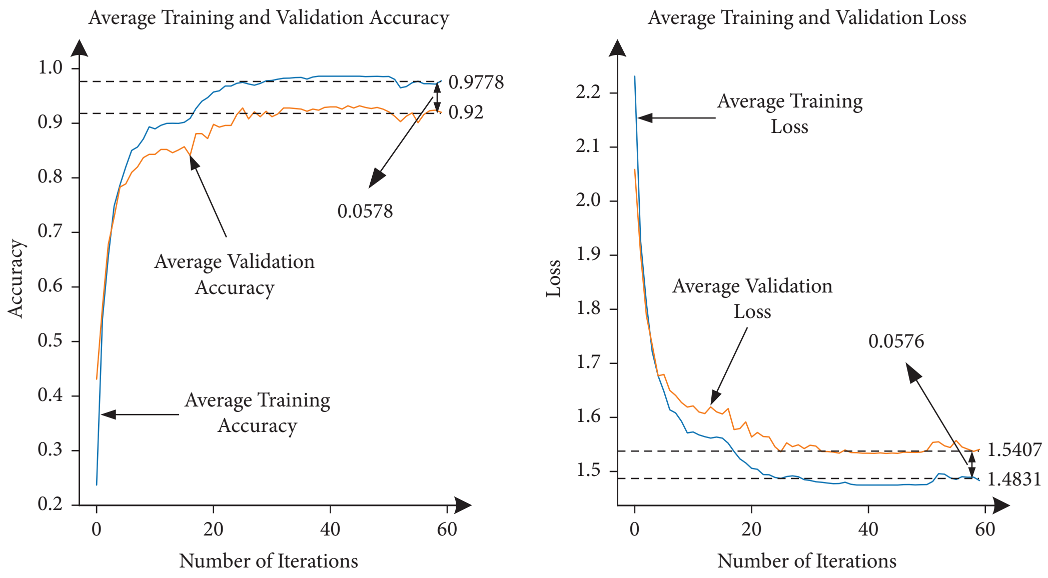

Figure 2: MNIST training results with L2 regularization.

MNIST dataset. It should mention the lower value for the reconstruction error parameter selected as the best performance result.

As depicted in Table 6, the RAE neural network architecture learning rate hyperparameters with the three maximum values of 0.003, 0.004, and 0.005 as the best

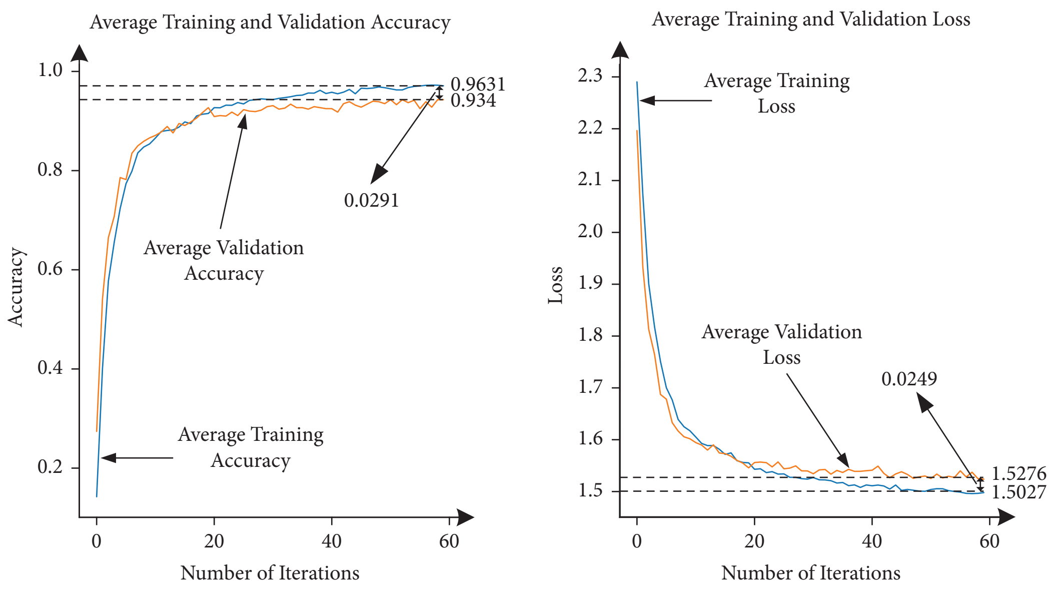

Figure 3: MNIST training results with dropout regularization.

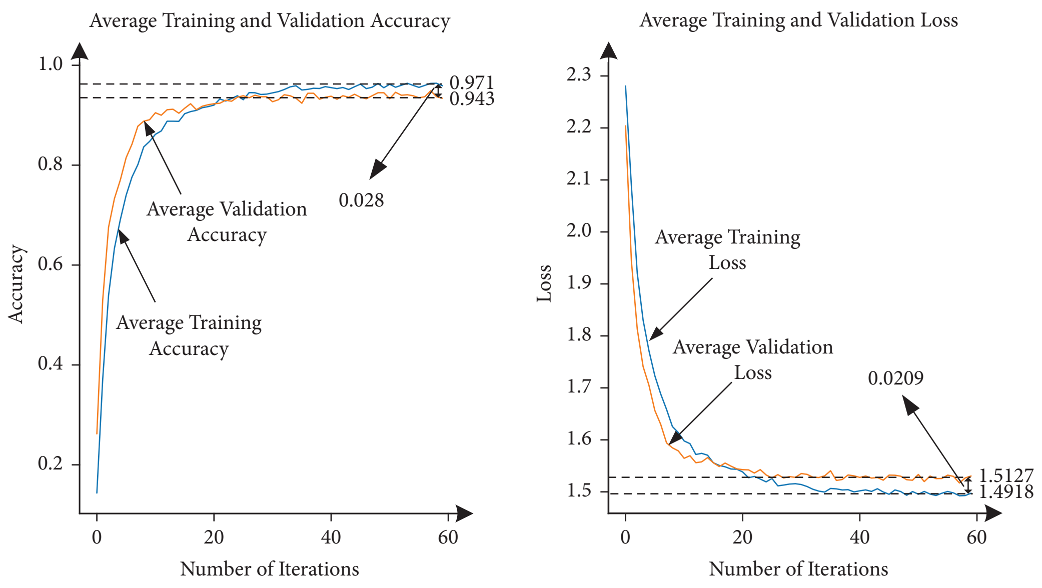

Figure 4: MNIST training results with hybrid algorithm.

performance are averaged as 0.004, and on the other side for the weight decay hyperparameter, the three values of 4.00E-07, 3.00E-07, and 2.00E-07 with the average value of 3.00E-07 with respect to the best performance of the reconstruction error on the MNIST dataset selected. The drop rate hyperparameter (p) and respected reconstruction error from 0.1 to 0.9 with the step of 0.1 trained on the MNIST dataset are expressed in Table 7.

Table 7 shows that as the rate of hyperparameter p increases and the reconstruction error decreases, the value of hyperparameter p as 0.1 performs best, and reconstruction error is defined as 0.038906.

(2) Rough Auto Encoder (RAE) performance vs. MNIST dataset. Since the input of the RAE fully connected network is one-dimensional (1D) data, it is necessary to process the image data into a 1D vector. Then, based on the *RAE*, the reconstruction errors of each algorithm on the MNIST dataset are compared. The smaller value of reconstruction error considers the better algorithm performance. Figure 5 and Table 8 show the compassion of the *RAE* reconstruction errors based on the MNIST dataset and reconstruction errors.

As Figure 5 states, using the proposed hybrid algorithm can more effectively reduce the impact of data noise and bias on model training under the unsupervised neural network and minimize the reconstruction error. As can be seen from Table 8, the reconstruction error on the MNIST dataset using the proposed hybrid algorithm has the least value (0.02438), and the maximum value belongs to dropout regularization (0.02836).

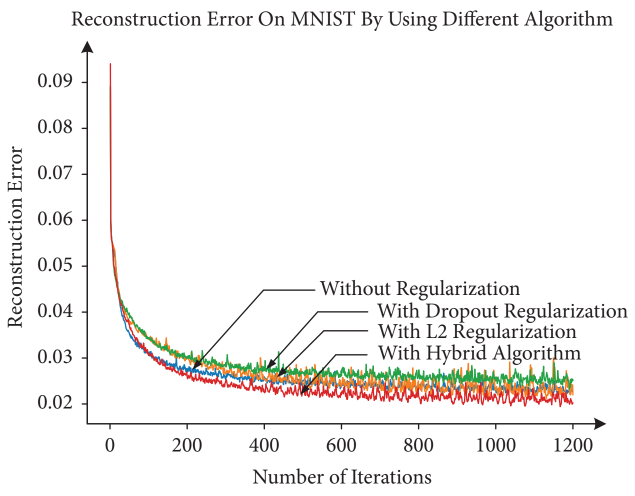

Figure 5: Reconstruction errors of different algorithms.

Table 5: MNIST dataset and models average training time for LeNet-5.

Algorithms	Average time (second)
Without regularization	26.64
L2 regularization	27.26
Dropout regularization	26.80
Hybrid algorithm	26.27

Table 6: Learning rate and weight decay hyperparameter for rough autoencoder trained on the MNIST dataset.

Learning rate	Reconstruction error	Weight decay	Reconstruction error
0.01	0.042926	0.0001	0.065342
0.009	0.044397	0.00009	0.057028
0.008	0.041419	0.00008	0.056897
0.007	0.040493	0.00007	0.057183
0.006	0.039244	0.00006	0.055813
0.005	0.034996	0.00005	0.050411
0.004	0.035179	0.00004	0.051364
0.003	0.037584	0.00003	0.047023
0.002	0.046424	0.00002	0.047418
0.001	0.047845	0.00001	0.044193
0.0009	0.04841	9.00E-07	0.038387
0.0008	0.048602	8.00E-07	0.038776
0.0007	0.049599	7.00E-07	0.041042
0.0006	0.049872	6.00E-07	0.038804
0.0005	0.051237	5.00E-07	0.038249
0.0004	0.052534	4.00E-07	0.035552
0.0003	0.057329	3.00E-07	0.035109
0.0002	0.063413	2.00E-07	0.036192
0.0001	0.063393	1.00E-07	0.038259

(3) Time complexity. As the last stage of algorithm comparison, the algorithm performance time in terms of time complexity concerning the MNIST dataset based on the RAE neural network architecture is calculated and listed in Table 9.

As depicted in Table 9, the maximum time belongs to L2 regularization but compared with the dropout regularization, and without regularization, the hybrid algorithm has a higher value. It can be seen from these data that although the proposed algorithm complicates the loss function, the time performance does not deteriorate, and compared with L2 regularization, the performance time is improved by 64.57 seconds.

Table 7: Drop rate hyperparameter (p) for *RAE* trained on the MNIST dataset.

p	Reconstruction error
0.1	0.038906
0.2	0.04041
0.3	0.041018
0.4	0.043298
0.5	0.044949
0.6	0.044767
0.7	0.049857
0.8	0.049451
0.9	0.054446

Table 8: The reconstruction error of the autoencoder on the MNIST data set.

Algorithms	Reconstruction error
Without regularization	0.02697
L2 regularization	0.02612
Dropout regularization	0.02836
Hybrid algorithm	0.02438

3.2. Experiment on the CIFAR10 Dataset.

3.2. Experiment on the CIFAR10 Dataset. As the second data set, CIFAR10 is used to compare and check the affordance algorithm. Then, as in the previous data set, LeNet-5 and RAE neural network architectures are used to extract the hyperparameters, and then, the performance of each algorithm is checked as follows:

3.2.1. LeNet-5 Network Architecture vs. CIFAR10 Dataset. To use the CIFAR10 image data set, as the image and LeNet-5 neural network architecture have the same dimension (32×32), the data can be used directly to speed up the convergence of the gradient descent algorithm, a normalization operation is performed on the dataset [30].

(1) LeNet-5 hyperparameters calculation. Based on the LeNet-5 network architecture, experiments are carried out under the CIFAR10 data set to find the hyperparameters (learning rate hyperparameter, the weight decay hyperparameter, and the drop rate hyperparameter p) through the 5-fold cross-validation method, and finally, the average value is taken as the fitting hyperparameter. The obtained value as hyperparameters is represented in Tables 10 and 11. It should mention the first three top-performing hyperparameters selected as the final hyperparameter, the same as the previous dataset.

As displayed in Table 10, the LeNet-5 neural network architecture has the maximum three values for the learning rate of 0.001, 0.0009, and 0.0007, with the average of 0.00087 as the final value for the learning rate and the weight decay

Table 9: Model training time for *RAE*.

Algorithms	Average time (second)
Without regularization	655.91
With L2 regularization	969.61
Dropout regularization	661.05
Hybrid algorithm	905.04

Table 10: Hyperparameter, learning rate, and weight decay for LeNet-5. Hyperparameter values (H_v), average train accuracy (AT_ACC), average value accuracy (Av_Acc), average train loss (AT_Loss), and average val loss (AV_ Loss).

H_v	Learning rate (LeNet-5)				Weight decay (LeNet-5)			
	AT_ACC	AT_Loss	Av_Acc	AV_ loss	AT_ACC	AT_Loss	Av_Acc	AV_ loss
0.01	0.100665	2.325601	0.087	2.329935	0.103843	2.302525	0.07757	2.302969
0.009	0.11661	2.321201	0.1032	2.325257	0.147838	2.269972	0.11444	2.276568
0.008	0.104085	2.313793	0.08103	2.318763	0.199558	2.22705	0.16228	2.247321
0.007	0.118305	2.334549	0.10925	2.342245	0.210432	2.218645	0.16343	2.242279
0.006	0.111295	2.344276	0.10447	2.349096	0.3635	2.093841	0.29837	2.151691
0.005	0.107287	2.344719	0.10937	2.341619	0.41001	2.051818	0.33659	2.116615
0.004	0.115883	2.34176	0.11308	2.344981	0.468345	1.996177	0.339	2.113303
0.003	0.108578	2.352489	0.1048	2.356354	0.46432	1.998804	0.34399	2.108453
0.002	0.318593	2.141057	0.25226	2.206989	0.564043	1.901666	0.36199	2.091373
0.001	**0.641668**	**1.820811**	**0.37911**	**2.075958**	**0.629605**	**1.835522**	**0.38038**	**2.076359**
0.0009	**0.639165**	**1.823672**	**0.37641**	**2.07908**	0.60578	1.858498	0.35942	2.094168
0.0008	0.629515	1.833676	0.36403	2.090948	0.616883	1.847611	0.36528	2.08955
0.0007	**0.611138**	**1.85274**	**0.37436**	**2.07947**	0.605088	1.85905	0.36004	2.094749
0.0006	0.627852	1.836708	0.35691	2.097898	0.625547	1.838004	0.3672	2.086873
0.0005	0.607905	1.857226	0.37204	2.081586	**0.633242**	**1.830361**	**0.38206**	**2.072243**
0.0004	0.566653	1.89906	0.35047	2.103409	0.620425	1.842975	0.36759	2.086849
0.0003	0.53116	1.935363	0.35188	2.101477	0.651738	1.81153	0.36794	2.086562
0.0002	0.506135	1.962681	0.3551	2.098744	0.64045	1.822653	0.37564	2.078767
0.0001	0.437427	2.031425	0.3265	2.126424	**0.648648**	**1.814119**	**0.38642**	**2.068413**

Table 11: Hyperparameter p for LeNet-5, average train accuracy (T_ACC), average train loss (T_Loss), average value accuracy (V_Acc), and average train loss (T_Loss).

p	Average train Acc	Average train loss	Average val Acc	Average val loss
0.1	0.622085	1.840741	0.37342	2.081888
0.2	0.602748	1.859754	0.37194	2.082475
0.3	0.543943	1.917456	0.37764	2.07829
0.4	0.49937	1.960975	0.3778	2.077712
0.5	0.440283	2.018242	0.3784	2.076063
0.6	0.404923	2.052566	0.34546	2.107493
0.7	0.35408	2.100192	0.34361	2.110094
0.8	0.272338	2.178325	0.28199	2.164043
0.9	0.18234	2.260305	0.16645	2.265217

hyperparameter the three maximum values 0.001, 0.0005, and 0.0001 performance averaged to 0.00053 and considered as weight decay hyperparameter value. The same process was followed for the hyperparameter drop loss (p), and the results are shown in Table 11.

As demonstrated in Table 11, the values of 0.3, 0.4, and 0.5 are the maximum three, with the best performance averaged to be 0.4 and defined as the hyperparameter p for the next steps.

(2) LeNet-5 performance and cross-validation five-fold. In order to verify the proposed hybrid algorithm's effectiveness, each model's performance is measured by cross-validation and the second data set (CIFAR10) according to the LeNet-5 architecture tests [22]. The average train accuracy, average validation accuracy, average train loss, and average validation loss parameters are calculated and compared. The cross-validation results for all affordance algorithms, along with the train accuracy, train loss, accuracy value, and loss value, are presented in Tables 12 and 13.

Based on the obtained result in cross-validation tables (Tables 12 and 13), the result of all four steps is illustrated in Figures 6–9.

Table 12: Average cross-validation for steps 1–3, train accuracy (Train Acc), train loss (Train Loss), and value accuracy (Val Acc).

Cross validation	Without regularization				L2 regularization				Dropout regularization			
	Train Acc	Train loss	Val Acc	Val loss	Train Acc	Train loss	Val Acc	Val loss	Train Acc	Train loss	Val Acc	Val loss
One-fold	0.7932	1.6674	0.38525	2.0737	0.8475	1.6152	0.38215	2.0739	0.7958	1.6684	0.369	2.0886
Two-fold	0.8222	1.6386	0.3712	2.0872	0.8685	1.594	0.37	2.0837	0.8039	1.6802	0.3621	2.0942
Three-fold	0.8269	1.6339	0.3758	2.0826	0.83	1.6322	0.353	2.1013	0.823	1.642	0.3825	2.0731
Four-fold	0.8244	1.6364	0.373	2.0849	0.8682	1.5946	0.3899	2.0666	0.8135	1.661	0.3775	2.0787
Five-fold	0.8088	1.6519	0.353	2.1057	0.8434	1.6192	0.4004	2.0565	0.7986	1.666	0.3851	2.0707
Average	0.815	1.6456	0.3716	2.0868	0.8515	1.611	0.3791	2.0764	0.807	1.6635	0.3752	2.081

Table 13: Average cross-validation for hybrid algorithm train accuracy (Train Acc), train loss (Train Loss), and value accuracy (Val Acc).

Cross validation	Train Acc	Train loss	Val Acc	Val loss
One-fold	0.7522	1.708	0.3902	2.0679
Two-fold	0.7427	1.7183	0.39215	2.0657
Three-fold	0.74585	1.7154	0.37185	2.0545
Four-fold	0.7738	1.6869	0.3996	2.0582
Five-fold	0.7385	1.722	0.3878	2.0697
Average	0.751	1.710	0.3883	2.0692

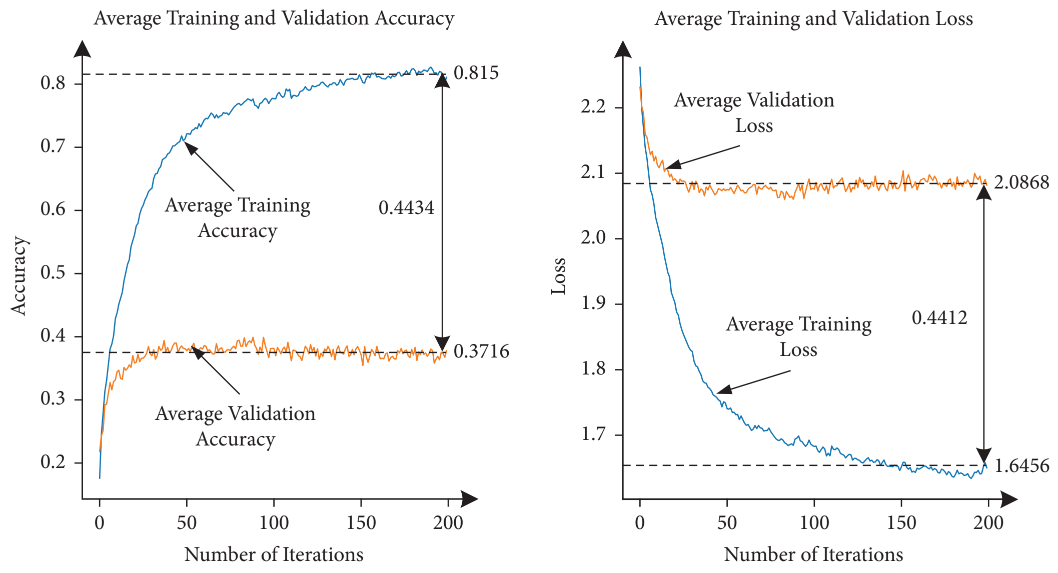

Figure 6: CIFAR10 dataset training results without regularization.

As Figure 6 shows, in Step 1 (without using any overfitting prevention method and regularization), as the number of iterations increases, the difference between the training set's loss (value 1.6456) and the validation set's loss (value 2.0868) is large; the error between the training set accuracy (value 0.815) and the validation set accuracy (value 0.3716) is about 0.4434. The loss value on the training set differs from the loss value on the validation set by about 0.4412, which also reflects that overfitting is serious from the side.

Figure 7 (Step 2: L2 regularization) illustrates that the difference between the training set's loss (value 1.611) and the validation set's loss (value 2.0764) is also large. The training set's loss value differs from the value on the validation set by about 0.4724. From the perspective of average accuracy, the error between the training set accuracy (value 0.8515) and the validation set accuracy (value 0.3791) is about 0.4654. The overfitting problem is also serious.

As can be seen from Figure 8 (Step 3: dropout regularization method), as the number of iterations increases, the difference between the training set's loss (value 1.6635) and the validation set's loss (value 2.081) becomes larger and larger; the error between the training set accuracy (value 0.807) and the validation set accuracy (value 0.3752) is about 0.4318, and from the perspective of average accuracy, the loss value on the training set differs from the loss value on the

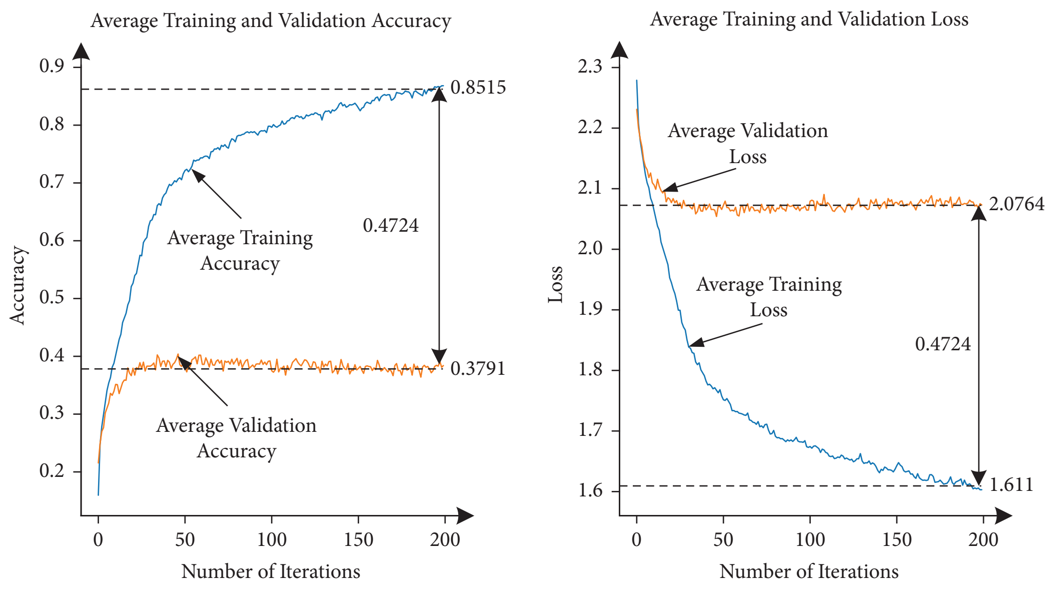

Figure 7: CIFAR10 training results with L2 regularization.

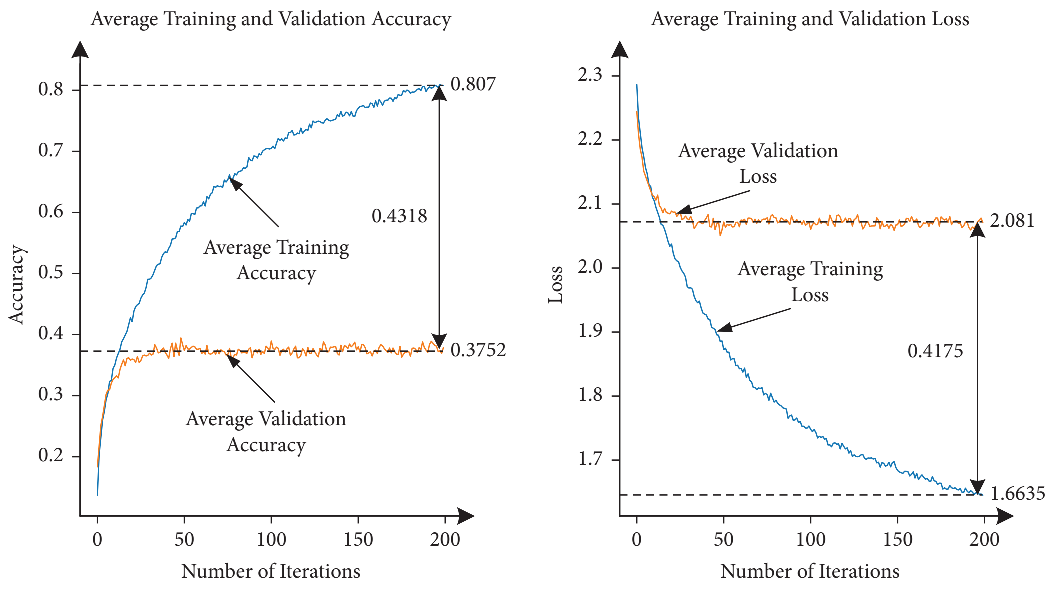

Figure 8: CIFAR10 training results with dropout regularization.

validation set by about 0.4175, which also reflects that overfitting is becoming more and more serious from the side.

The step 4 (hybrid algorithm) results shown in Figure 9 and Table 14 are the best compared to the previous steps and regarding average accuracy values. The error between the training set accuracy (value 0.751) and the validation set accuracy (value 0.3883) is about 03627. From the perspective of loss value, the stability during training is greatly improved compared to other steps; the loss value on the training set (value 1.710) differs from the loss value on the validation set (value 2.0692) by about 0.3592, which also shows that using the proposed hybrid algorithm is more effective than using other regularization methods.

(3) Time complexity. To compare the performance of each model in terms of time complexity, the cross-validation method is used, and the average time is calculated based on the LeNet-5 neural network architecture and CIFAR10 dataset, as indicated in Table 14.

As can be seen from Table 14, the average time difference of each model in the CIFAR10 dataset training is very small, and without regularization with the 227.27 value, the least value, L2 regularization is the maximum (235.45), and proposed hybrid algorithm has the second least value (228.09) among other algorithms. The result shows that although the hybrid algorithm complicates the loss function, the time performance does not deteriorate. Compared with L2 regularization, the time performance is improved by 7.36

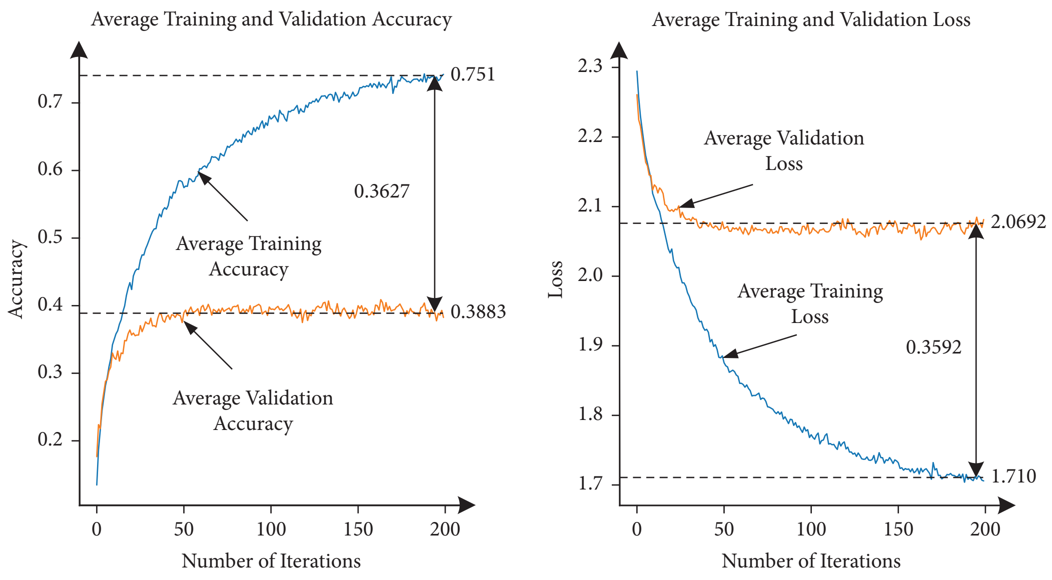

Figure 9: CIFAR10 training results with the hybrid algorithm.

Table 14: Average model training time for LeNet-5.

Algorithms	Average time (second)
Without regularization	227.27
L2 regularization	235.45
Dropout regularization	233.47
Hybrid algorithm	228.09

seconds, and the difference with the without regularization is 0.82 seconds.

3.2.2. Rough Auto Encoder (RAE) vs. CIFAR10 Dataset. Similar to the method used in the previous data set, the *RAE* -based neural network architecture was also used for the *CIFAR10 dataset.* To conduct the following test, the fully connected layers are used to build the RAE's encoder and decoder parts, and the hyperparameters are determined. Then, all mentioned algorithms are compared based on the reconstruction error to measure the model's performance.

(1) Rough Auto Encoder (RAE) hyperparameters calculation. Based on the *Autoencoder's RAE network architecture, several* experiments are carried out under the CIFAR10 data set. The hyperparameters are determined from the results of many experiments, and finally, the average value is taken as the appropriate hyperparameter. The neural network's performance with different values of the hyperparameter learning rate, weight decay hyperparameter, and drop rate hyperparameter p are displayed in Tables 15 and 16, respectively. In order to select more suitable hyperparameters, this paper selects the first three hyperparameters with the best performance and then, takes the average value as the final hyperparameter.

Table 15 indicates that the RAE neural network architecture is trained data based on the CIFAR10 dataset. The top three perform best when the learning rate

Table 15: Learning rate and weight decay hyperparameter for autoencoder trained on the CIFAR10 dataset.

Learning rate	Reconstruction error	Weight decay	Reconstruction error
0.01	0.062009	1.00E-05	0.033044
0.009	0.062047	9.00E-06	0.037125
0.008	0.061958	8.00E-06	0.032976
0.007	0.061975	7.00E-06	0.033005
0.006	0.061991	6.00E-06	0.037
0.005	0.062031	5.00E-06	0.032751
0.004	0.061894	4.00E-06	0.032763
0.003	0.048493	3.00E-06	0.032802
0.002	0.046619	2.00E-06	0.032881
0.001	0.036068	1.00E-06	0.033129
0.0009	0.034174	**9.00E-07**	**0.032664**
0.0008	0.034174	8.00E-07	0.032751
0.0007	0.033736	7.00E-07	0.03283
0.0006	0.035271	6.00E-07	0.032748
0.0005	**0.03735**	5.00E-07	0.032959
0.0004	**0.033001**	4.00E-07	0.032808
0.0003	**0.034589**	**3.00E-07**	**0.032485**
0.0002	0.036847	2.00E-07	0.032824
0.0001	0.037022	**1.00E-07**	**0.032701**

Table 16: Hyperparameter drop rate (p) for RAE Vs CIFAR10.

p	Reconstruction error
0.1	0.032797
0.2	0.033228
0.3	0.033777
0.4	0.037264
0.5	0.037555
0.6	0.037791
0.7	0.044
0.8	0.044079
0.9	0.049096

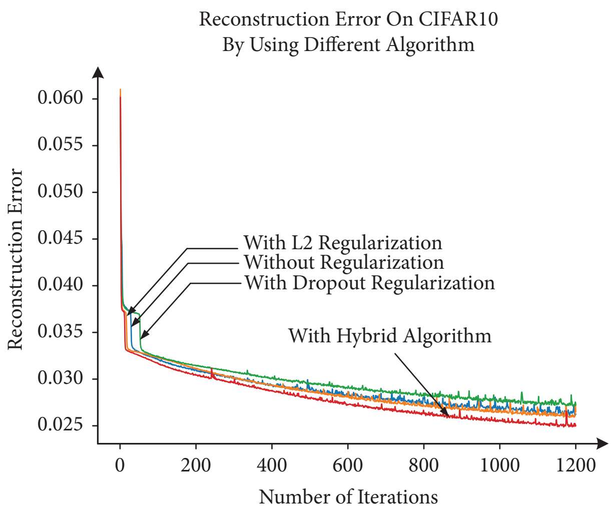

Figure 10: Reconstruction errors of different algorithms.

hyperparameters are equal to 0.0003, 0.0004, and 0.0005 averaged as 0.0004, and for the hyperparameter weight decay as 9.00E-07, 3.00E-07, and 1.00E-07, and the top three with the best performance averaged as 4.33E-07. The drop rate (p) hyperparameter from 0.1 to 0.9 with the step of 0.1 is given in Table 16.

As Table 16 shows, the reconstruction error's loss value on the CIFAR10 dataset increases as the drop rate hyperparameter p increases, and the best drop rate hyperparameter belongs to the value 0.1.

(2) Rough Auto Encoder (RAE) performance vs. CIFAR10 dataset. Since the input of the fully connected network is one-dimensional data, and the CIFAR10 dataset contains $32 * 32 * 3$ RGB images, then the image data should convert into a one-dimensional vector. Then, based on the *RAE*, the reconstruction errors of each algorithm on the CIFAR10 dataset are compared. The smaller value of reconstruction error considers the better algorithm performance. Figure 10 and Table 17 offer the compassion of the *RAE* reconstruction errors based on the CIFAR10 dataset and reconstruction errors.

As Figure 10 indicates, the proposed hybrid algorithm can more effectively reduce the impact of data noise and bias on model training under the RAE unsupervised neural network and minimize the reconstruction error. The reconstruction errors of each algorithm are presented in Table 17. The table shows that the reconstruction error on the CIFAR10 dataset using the proposed hybrid algorithm is the least (0.02808), and the dropout is the most compared to the algorithms; the reconstruction error value is reduced by 0.00174.

(3) Time complexity. To compare the algorithm's performance in terms of time complexity based on the CIFAR10 and RAE neural network architecture, the obtained results are displayed in Table 18.

As Table 18 demonstrates, the average time by the proposed hybrid algorithm is 4560 seconds as the second maximum value, but it can be seen that although the proposed algorithm complicates the loss function, the time performance does not deteriorate. Compared with L2 regularization, the time performance is improved by 945.74 seconds.

Table 17: The reconstruction error of RAE on the CIFAR10 data set.

Algorithms	Reconstruction error
Without regularization	0.02907
L2	0.02886
Dropout	0.02982
Hybrid algorithm	0.02808

Table 18: Model training time for autoencoder.

Algorithms	Average time (second)
Without regularization	2521.93
L2 regularization	5506.71
Dropout regularization	2431.45
Hybrid algorithm	4560.97

4. Result

The trained models were applied to the test set of the MNIST and CIFAR10 datasets. Concerning the different network architectures, cross-validation was adopted to evaluate the performance of the without regularization, L2, dropout, and hybrid algorithm models to calculate the accuracy and training time. The results using different algorithms are specified in Table 19.

As Table 19 explains, the highest accuracy belongs to the proposed hybrid algorithm, and the lowest one fits without regularization; the highest reconstruction errors parameter goes for the dropout regularization and the least performed by the hybrid algorithm for both the data sets MNIST and CIFAR10. The accuracy result for the MNIST data shows that using the hybrid algorithm causes an improvement of

Table 19: The accuracy and training time of different models VS. both the data set.

Algorithms	Data set MNIST				Data set CIFAR10			
	Accuracy (%)	Reconstruction errors	Training time (s) LeNet-5	Training time (s) RAE	Accuracy (%)	Reconstruction errors	Training time (s) LeNet-5	Training time (s) RAE
Without regularization	90.3	0.02697	26.64	655.91	37.16	0.02907	227.27	2521.93
L2 regularization	92.0	0.02612	27.26	969.61	37.91	0.02886	235.45	5506.71
Dropout regularization	93.4	0.02836	26.80	661.05	37.52	0.02982	233.47	2431.45
Hybrid algorithm	94.3	**0.02438**	26.27	905.04	38.83	**0.02808**	228.09	4560.97

4.0%, 2.3%, and 0.9%; on the other side, for the CIFAR10, the accuracy improved by 1.67%, 0.92%, and 1.31%, in comparison with without regularization, L, and dropout model respectively. The training time for LeNet-5 is less for the hybrid algorithm and second high for RAE neural network in both data sets. But the results show that its performance improves by 0.99 seconds in MNIST and 7.36 seconds in CIFAR10, compared to L2 regularization for LeNet-5. Also, RAE trains the neural network on the MNIST dataset, the hybrid algorithm improves the time performance by 64.57 seconds and 945.74 seconds on the CIFAR10 compared to L2 regularization.

5. Conclusion

This study introduces the hybrid algorithm as an improved algorithm for the neural networks model training process based on L2 and dropout regularizations. The proposed algorithm combines the advantages of the L2 regularization term with the loss function so that the network model attenuates the weight parameters during the training process and prevents overfitting; on the other side, the dropout benefits from optimizing the network structure in the training process and some neurons are randomly suppressed to obtain different network structures; finally, a model-averaging strategy is used in the testing phase to prevent overfitting. Under the cross-validation method, comparative experiments are conducted under different datasets by designing different neural network architectures. Based on the supervised (LeNet-5) and unsupervised (RAE) neural network architectures and verified under the MNIST (grayscale— $28 \times 28 \times 1$) and CIFAR10 (RGB—$32 \times 32 \times 3$) datasets. The obtained results show that hybrid algorithms can effectively improve the model's prediction performance and performance without much increment in the training time, and even compared to L2 regularization, the results are improved.

In addition, the proposed algorithm can reduce the reconstruction error to a certain extent. The experimental results on small samples show that although the loss function of the hybrid algorithm proposed in this paper becomes complicated, the algorithm can effectively improve the model's prediction performance and reduce the reconstruction error without reducing the time performance. However, this also has a small drawback. Due to the introduction of the hybrid algorithm, the loss function becomes more complex. In each iteration process, the forward and backward propagation calculations require additional time, leading to the network model training taking longer compared with no regularization and dropout regularization. Given this potential shortcoming, we will suggest that the algorithm can be optimized in future work to reduce the algorithm's time complexity and speed up the convergence of the network model.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Xiaoyun Xie, Ming Xie, Ata Jahangir Moshayedi Methodology Xiaoyun Xie, Ming Xie, and Mohammad Hadi Noori Skandari conceptualized this study. Xiaoyun Xie, Ata Jahangir Moshayedi, Mohammad Hadi Noori Skandari investigated this study. Xiaoyun Xie found the resources of this study. Xiaoyun Xie and Ming Xie' Ata Jahangir Moshayedi wrote the original draft. Ata Jahangir Moshayedi and Mohammad Hadi Noori Skandari reviewed and edited the study. All authors have read and agreed to the published version of the manuscript.

Acknowledgments

This work was supported by 03 Special Project and 5G Project of Jiangxi Province (Grant no. 20204ABC03A18), The National Key Research and Development Program of China (Grant no. 2020YFB1713700).

References

[1] E. G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, https://arxiv.org/abs/1207.0580.

[2] A. Zhou and K. Luo, “Sparse dropout regularization method for convolutional neural networks,” *Journal of Chinese Computer Systems*, vol. 39, pp. 1674–1679, 2018.

[3] X. Zhong, E. Chen, R. Luo, and Y. Lu, “Multi-scale fusion dropout optimization algorithm,” *Journal of Huazhong University of Science and Technology*, vol. 46, pp. 35–39, 2018.

[4] G. Ghiasi, T.-Y. Lin, and V. Q. Le, “Drop block: a regularization method for convolutional networks,” *Advances in Neural Information Processing Systems*, vol. 31, no. NIPS 2018, 2018.

[5] J. Cheng, G. Zeng, D. Lu, and B. Huang, “Improved convolution neural network model averaging method based on Dropout,” *Journal of Computer Applications*, vol. 34, pp. 1601–1606, 2019.

[6] H. Pham and V. Q. Le, “Autodropout: learning dropout patterns to regularize deep networks,” in *Proceedings of the THIRTY-FIFTH AAAI Conf. Artif. Intell. THIRTY-THIRD Conf. Innov. Appl. Artif. Intell. Elev. Symp. Educ. Adv. Artif. Intell*, pp. 9351–9359, CA, USA, February 2021.

[7] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning (still) requires rethinking generalization,” *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[8] N. A. Gomez, I. Zhang, K. Swersky, Y. Gal, and E. G. Hinton, “Learning Sparse Networks Using Targeted Dropout,” 2019, https://arxiv.org/abs/1905.13678.

[9] Q. Xu, M. Zhang, Z. Gu, and G. Pan, “Overfitting remedy by sparsifying regularization on fully-connected layers of cnns,” *Neurocomputing*, vol. 328, pp. 69–74, 2019.

[10] R. Keshari, S. Ghosh, S. Chhabra, M. Vatsa, and R. Singh, “Unravelling small sample size problems in the deep learning world,” in *Proceedings of the 2020 IEEE SIXTH Int. Conf. Multimed. BIG DATA (BIGMM 2020)*, pp. 134–143, New Delhi, India, September 2020.

[11] N. Golsanami, M. N. Jayasuriya, W. Yan et al., “Characterizing clay textures and their impact on the reservoir using deep learning and lattice-Boltzmann simulation applied to sem images,” *Energy*, vol. 240, Article ID 122599, 2022.

[12] O. Sysoev and O. Burdakov, “A smoothed monotonic regression via l2 regularization,” *Knowledge and Information Systems*, vol. 59, pp. 197–218, 2019.

[13] S. Hahn and H. Choi, “Understanding dropout as an optimization trick,” *Neurocomputing*, vol. 398, pp. 64–70, 2020.

[14] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, ICLR, Louisiana, USA, 2019.

[15] M. A. Rafi, J. Wu, and K. M. Hasan, *L2-constrained Remnet for Camera Model Identification and Image Manipulation Detection*, pp. 267–282, ECCV Workshops, Israel, 2020.

[16] N. Srivastava, E. G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[17] W. Sun, J. Tang, and C. Bai, “Evaluation of university project based on partial least squares and dynamic back propagation neural network group,” *IEEE Access*, vol. 7, pp. 69 494–69503, 2019.

[18] W. Feng, J. Tang, and T. X. Liu, “Understanding dropouts in moocs,” *AAAI*, vol. 33, pp. 517–524, 2019.

[19] M. Kamp, L. Adilova, J. Sicking et al., “Efficient decentralized deep learning by dynamic model averaging,” *MACHINE LEARNING AND KNOWLEDGE DISCOVERY IN DATABASES*, pp. 393–409, ECML PKDD, Spain, 2019.

[20] V. Nair and E. G. Hinton, *Rectified Linear Units Improve Restricted Boltzmann Machines*, pp. 807–814, ICML, Delhi, India, 2010.

[21] X. Wang, S. Zhang, S. Wang, T. Fu, H. Shi, and T. Mei, “Misclassified vector guided softmax loss for face recognition,” *AAAI*, vol. 34, no. 7, pp. 12241–12248, 2020.

[22] V. Suppakitpaisarn, A. Ariyarit, and S. Chaidee, “A voronoi-based method for land-use optimization using semidefinite programming and gradient descent algorithm,” *International Journal of Geographical Information Science*, vol. 35, no. 5, pp. 999–1031, 2021.

[23] Q. N. Islam, *Mastering PyCharm*, Packt Publishing Ltd, 2015.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do imagenet classifiers generalize to imagenet?” in *Proceedings of the International Conference on Machine Learning*, pp. 5389–5400, CA, USA, June 2019.

[26] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Technical Report, BibSonomy, Germany, 2009.

[27] P. D. Kingma and L. J. Ba, “Adam: a method for stochastic optimization,” in *Proceedings of the International Conference on Learning Representations*, San Diego. CA, USA, May 2015.

[28] Z. Hu, H. Zhao, and J. Peng, “Low-rank reconstruction-based autoencoder for robust fault detection,” *Control Engineering Practice*, vol. 123105 156 pages, 2022.

[29] S. Baik, M. Choi, J. Choi, H. Kim, and K. M. Lee, “Meta-learning with adaptive hyperparameters,” vol. 33, pp. 20 755–820 765, 2020.

[30] N. D. Bokde, Z. M. Yaseen, and G. B. Andersen, “ForecastTB-an R package as a test-bench for time series forecasting-application of wind speed and solar radiation modeling,” *Energies*, vol. 13, p. 2578, 2020.

[31] W. Li, L. Zhu, Y. Shi, K. Guo, and E. Cambria, “User reviews: sentiment analysis using lexicon integrated two-channel cnn-lstm family models,” *Applied Soft Computing*, vol. 94, Article ID 106435, 2020.

[32] C. G. Vázquez, A. Breuss, O. Gnarra, J. Portmann, A. Madaffari, and G. D. Poian, “Label noise and self-learning label correction in cardiac abnormalities classification,” *Physiological Measurement*, vol. 43, 2022.