Received February 21, 2022, accepted March 28, 2022, date of publication April 4, 2022, date of current version April 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3164669

Multi-Layer Perceptron Training Optimization Using Nature Inspired Computing

ALI AL BATAINEH[1], (Member, IEEE), DEVINDER KAUR[2], (Life Senior Member, IEEE), AND SEYED MOHAMMAD J. JALALI[3], (Member, IEEE)
[1]Department of Electrical and Computer Engineering, Norwich University, Northfield, VT 05663, USA
[2]Department of Electrical Engineering and Computer Science, The University of Toledo, Toledo, OH 43606, USA
[3]Institute for Intelligent Systems Research and Innovation, Deakin University, Burwood, VIC 3125, Australia

Corresponding author: Ali Al Bataineh (aalbatai@norwich.edu)

This work was supported in part by Norwich University, and in part by The University of Toledo.

ABSTRACT Although the multi-layer perceptron (MLP) neural networks provide a lot of flexibility and have proven useful and reliable in a wide range of classification and regression problems, they still have limitations. One of the most common is associated with the optimization algorithm used to train them. The most commonly used training method is stochastic gradient descent with backpropagation (or backpropagation for short) because it is mathematically tractable (given that the activation functions are differentiable). However, backpropagation is not guaranteed to find the globally optimal set of weights and biases. As a result, the MLP is often incapable of obtaining a desirable solution to the problem. Clonal selection algorithms (CSA) are optimization procedures that effectively explore a complex and large space to find values near the global optimum. Consequently, CSA can be used to solve the problem of training MLP networks. This paper presents a novel implementation of CSA for training MLP architectures to solve real-world problems such as breast cancer diagnosis, active sonar target classification, wheat classification, and flower classification. The CSA is used to find the optimal weights and biases that will significantly increase the classification accuracy of the MLP. The performance of our proposed approach is compared with other popular training methods: genetic algorithm (GA), ant colony optimization (ACO), particle swarm optimization (PSO), Harris hawks optimization (HHO), moth-flame optimization (MFO), flower pollination algorithm (FPA), and backpropagation (BP). The comparison is benchmarked using five classification datasets: Iris Flower, Sonar, Wheat Seeds, Breast Cancer Wisconsin, and Haberman's Survival. Comparative study results illustrate the improvements in MLP performance gained by using CSA over other training methods, and hence it can be considered a competitive approach to training MLP networks when solving real-world applications in various disciplines.

INDEX TERMS Breast cancer, clonal selection algorithm, multi-layer perceptron, nature inspired computing, neural networks, optimization, sonar target classification.

I. INTRODUCTION
The development of fast, reliable and inexpensive computing power has transformed the optimization field over the last decades [1]. Optimization refers to the process or methodology of making something (such as a system, design or decision) as ideal, functional, or efficient as possible. In mathematical terms, it refers to either maximization or minimization of an objective function relative to a given set, often representing a range of all possible choices in a particular situation [2]. The objective function allows a comparison of the different options to determine which might be the "best" [3]. Almost every problem in science and engineering can be formulated as an optimization problem. Some problems can be dealt with and solved by classical optimization techniques; however, most problems are either unmanageable or too costly in terms of time and other resources to be solved using traditional methods [4]. Fortunately, these hard problems can be solved by modeling inspiration from nature. Nature has been a great source of inspiration for the development of new computing technologies that are flexible, robust, and scalable

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Jiang.

by observing how naturally occurring phenomena solve complex problems in various environmental situations [5]. This produced groundbreaking research that led to the development of optimization techniques like CSA, GA, PSO, ACO, HHO, MFO, and FPA [6]–[11]. These are global optimization techniques that can generate near-optimum or even optimum solutions. They have numerous advantages which have made them enormously popular; these include:

- Better at handling very large spaces where there are a large number of parameters involved.
- Have great parallel abilities.
- Optimizes both discrete and continuous functions as well as multi-objective problems.
- Do not need any derived knowledge that might not be available for many real-world problems.
- Faster and more effective than classical approaches and provide not just one good solution but many of them.

Training neural network architectures such as MLP can be formulated as optimization problems; hence, they can be solved by some optimization methods. Stochastic gradient descent (SGD) is a popular optimization algorithm that is often adopted to train neural network models since it is flexible and mathematically tractable. SGD uses backpropagation to calculate the gradients for each parameter (i.e, weight, bias) so that new values for the variables can be computed [12]. However, the SGD learning algorithm might converge to a set of sub-optimal weights and biases from which it cannot escape. As a result, the MLP model is often incapable of finding a desirable solution to a problem at hand. Additionally, the algorithm requires the activation and loss functions to be differentiable in order to calculate the gradients. Generally speaking, it is not a simple task to find an optimum solution or even sub-optimum solutions. The good part is that it is possible to solve these optimization problems by drawing inspiration from nature to produce near-optimal solutions [13].

Several nature-inspired algorithms have been proposed in the literature to train MLP systems, with promising results [14]–[22]. GA methods have been widely used to solve a variety of optimization problems, including the training of the MLP. An early and successful approach is described in [23]. The authors proposed using GA to find a near-optimal set of weights of MLPs in a relatively short time. They ran a number of experiments using data from a sonar image classification problem. Their experimental results showed the GA is better than SGD for training MLP models. Similar works that used GA to train MLP models for various classification and regression tasks are reported in [24]–[28]. PSO algorithm has also been used successfully used to train neural networks. One of the first applications of the PSO algorithm was to train MLPs, which involved finding appropriate weights for MLPs [29]. Other remarkable works that use the PSO algorithm to train neural networks are described in [30]–[32]. ACO has also shown promising results for training MLP models [33], [34]. Recnetly, artificial immune system (AIS) methods have been applied to train MLP mdeols due to their ability to balance the search space exploration and exploitation [35]–[39]. HHO [40], MFO [41], and FPA [42] have also been used to find the weights and biases of MLP models and have proven to be very competitive in terms of achieving high classification accuracy.

Differential Evolution [43], Artificial Bee Colony [44], Grey Wolf optimizer [45], Lightning Search Algorithm [46], Multi-Verse Optimizer [47], and Whale Optimization Algorithm [48] are also other popular nature-inspired algorithms that have been used to train MLPs and have shown good performance. Hybrid systems, which combine two algorithms, have also been proposed to train MLP models effectively. In [49], the authors presented a hybrid model that combines GA and gradient descent backpropagation, where GA is utilized to initialize and optimize the weight parameters of the MLP model to classify medical data. In [50], the authors introduced a hybrid model that combines PSO and Gravitational Search Algorithm to train MLPs (or FFNNs) to investigate the effectiveness of these algorithms in avoiding local minima. Other proposed hybrid models for training MLPs are described in [51]–[53].

This research focuses on utilizing the CSA [54], to train MLP models, to improve their performance for solving classification tasks. CSA are metaheuristic optimization algorithms inspired by theoretical and experimental immunology to solve computational problems from optimization, mathematics, and engineering. It belongs to the board field of AIS. AIS encompass any system or computational tool that extracts ideas and metaphors from the biological immune system to solve problems [55], [56]. As opposed to swarm intelligence and evolutionary algorithms that emerged from one main central idea and developed into several branches and variations, AIS was proposed considering various features, processes, and immune system models [57]. The idea was first proposed by Farmer and team in 1986 [58], with some important work on Bersini and Varela's immune networks in 1990 [59]. Many variants, such as the CSA, negative selection algorithm, immune networks, and others, have been implemented over the last few decades. CSA is very encouraging, and it has demonstrated efficient performance in the optimization and pattern recognition domains. [60]. CSA can estimate known solutions and simulate the adaptive immunity behavior in complex systems. It searches the solution space by creating a population of candidate solutions called antibodies. It then evaluates each antibody's affinity (fitness) and matches it against the affinity of the potential solution called an antigen. Those antibodies with higher affinity than the antigen affinity will then be selected, and a number of clones are created from each antibody proportional to the respective affinity. The clones then undergo a hypermutation process, inversely proportional to corresponding affinities. This hypermutation process is called affinity maturation. Finally, depending on the mutate clones affinities, re-selection is performed to select the optimal mutate clone, which evolves and becomes the main antigen for the next generation, and the same process is repeated several times until the desired solution is obtained.

This paper proposes an effective CSA-based method for training MLP architectures to solve difficult real-world problems in a variety of domains, including breast cancer detection, active sonar target classification, wheat classification, and flower classification. The following are the contributions of the study:

1) We develop an encoding scheme for MLP weights and biases that enables efficient implementation of various mutation types.
2) We evaluate our proposed methodology on five real-world benchmark classification datasets with varying levels of difficulty.
3) We compared our method to six well-known nature-inspired methods and found that it produced better results.

The paper is structured as follows. Section II delves into the details of the clonal selection theory, focusing particularly on those elements abstracted and implemented in the CSA method. Section III provides an overview of the CSA method, explaining in detail the powerful features and main operators of the proposed algorithm with examples. Section IV discusses MLP models and how they learn using gradient descent backpropagation. The complete implementation of the proposed CSA-based method for training MLP systems is explained in Section V. Experimental design and a discussion of the presented results are presented in Sections VI and VII, respectively. Finally, conclusions and future work are presented in Sections VIII and IX, respectively.

II. CLONAL SELECTION THEORY

The key advances within the AIS centered on three main immunological theories: clonal selection, negative selection and immune networks. AIS researchers have focused mostly on the biological immune system's learning and memory processes inherent in the clonal selection and immune networks and the negative selection principle to generate detectors that can classify changes in self. This research's main focus is the CSA, a novel implementation of AIS inspired by the clonal selection theory. The clonal selection theory is an immunology theory that explains the basic features of the adaptive immune system, particularly a theory to describe the diversity of antibodies used to protect the organism against foreign attacks. [61].

An antibody itself is a a compound produced by B lymphoid cells that has the ability to neutralise a specific molecule. Each B lymphocyte produces special or unique antibodies of a specific type. When first proposed, the theory was controversial and competed with another paradigm named template theory. Currently, the clonal selection theory is marked as fact, taking into account the vast amount of experimental evidence [62]. The clonal selection theory specifies that an organism has a a set of individually unique antibodies that can recognize all antigens with a specific degree of specificity. Whenever an antibody's primary antibody is recognised, a cell is stimulated to molecularly connect towards the proliferate, antigen, and start producing further molecules with much the same antibody [63]. During in the cell proliferation phase, genetic mutations take place inside the replica of cell lines that enhance the match as well as predilection only with antigen. The above means allowing the cells' potential to adhere towards the antigen to help to strengthen both with antigen exposure. This antigen-driven sampling of reproduction cell lines can be viewed as a Darwinian perfect example in which the fittest cell lines (best candidate with antigens) have been picked to stay alive, and gene mutation can provide cell diversity [64]. Figure 1 gives an overall picture of the clonal selection process.

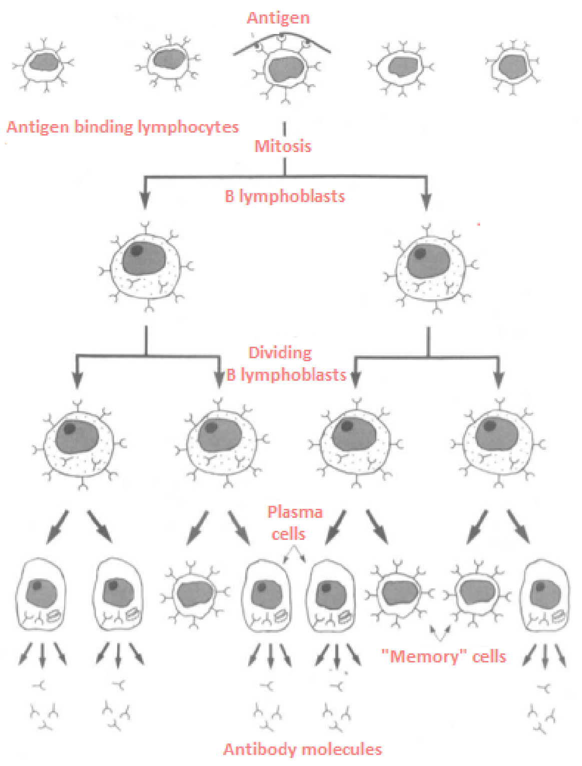


FIGURE 1. Clonal selection process, taken from [65].

It depicts B lymphocyte cells that specifically bind antigens only at upper edge. Following that, the B cell multiplies (mitoses) and generates a large number of B lymphoblasts, which differentiate into plasma cells producing memory cells or immunoglobulin.In an effective immune response, plasma cells produce a large number of antibodies to a specific antigen, which results in the removal of the antigen. Memory cells are known to remain within the original host and support a quick supplementary response if a specific immune resurfaces. This is the phenomenon of the acquired immunity. Another distinguishing feature of acquired immunity is how the system develops the ability to differentiate between self and non-self. This capacity is called tolerance and explains the failure of the system to launch an immune response against a specific antigen, such as self-antigens [66].This ability develops prior to the organism's birth, while the immune system itself originates. This ability to not produce antibodies against one's own cells can be acquired for foreign antigens given to the organism before birth. This tolerance, however, cannot be maintained if the antigen is not permanently present in the organism, which means that the system may forget unused knowledge [67].

III. CLONAL SELECTION ALGORITHM

A. OVERVIEW

The CSA is a global optimization method inspired by the principles of the clonal selection theory of acquired immunity. CSA has shown high performance in diverse, challenging optimization problems over traditional optimization techniques [68], [69]. The algorithm takes inspiration from the following elements of the clonal selection theory:

- Immune system produces multiple antibodies to combat invaded antigen/pathogen.
- Filter the antibodies produced based on affinity. Affinity is the degree of recognition between an antibody and an antigen. The higher the affinity, the better the recognition.
- Antibodies with higher affinity are cloned.
- The clone set of duplicate antibodies is then subjected to affinity maturation (mutation) process to match the antigen better.

In the CSA algorithm, an antigen represents the problem to optimize, and the algorithm generates a set of candidate solutions (antibodies) to solve the problem [70]. The quality of the candidate solutions is evaluated based on the affinity (better matched with antigen). The process of exploring feasible solutions is the means of immune cells identifying antigens and launching immune responses in the immune system [71]. CSA is comparable to GA; it typically develops solutions by repeating bio-operators such as cloning, mutation, and selection cycle to a population of candidate solutions and remains the best solutions. These operators help increase diversity and provide a means for potentially escaping local optima [72].

B. ALGORITHM

The following gives an overview of the steps of the CSA algorithm [64].

1) Initialize an antigen (potential solution) randomly.
2) In response to the initial antigen, a random population of candidate solutions (antibodies) with the same format is created to fight the antigen.
3) The affinity (fitness) score is measured for each antibody.
4) Select the antibodies that have affinity higher than the antigen.
5) Clone the selected antibodies to produce more like them. Depending on the designer's choice, number of clones can be fixed for all antibodies or proportion to their affinity (rank-based).
6) Mutate each antibody randomly to enhance diversity further and implement a means for potentially escaping local optima. In other words, an attempt to discover even more desirable candidates. Here, the mutation rate is inversely proportional to the affinity.
7) Affinity is measured for every mutated antibody; select the best and kills off the rest.
8) Replace the initial antigen with the best antibody and repeat the process until stopping criteria is reached, such as optimal solution has been found or the maximum number of iterations has been achieved.

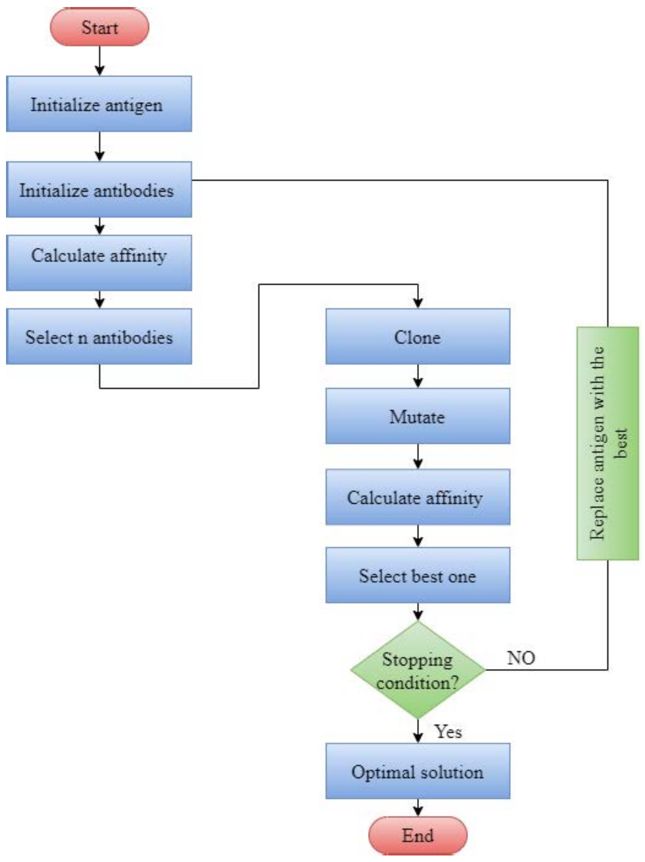
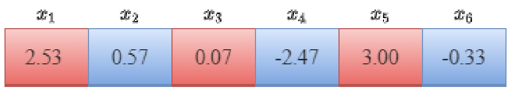


FIGURE 2. Flow chart diagram describing stages of CSA algorithm.

Figure 2 provides a complementing diagram of the workflow phases of the CSA method.

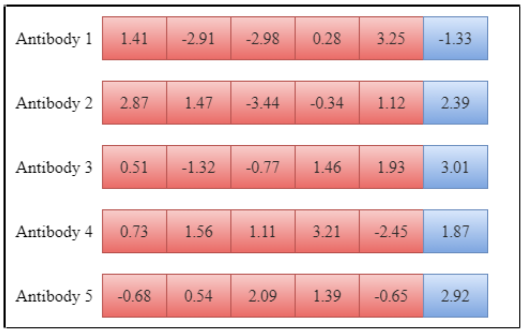
1) ANTIGEN INITIALIZATION

When solving an optimization problem, i.e., optimizing a function's variables, an antigen is a potential solution based on which the final optimum solution is evaluated. This potential solution will help select those antibodies in each generation whose affinity (fitness) are better than the antigen's affinity. First, a random antigen is created, representing the function's possible parameters over a specific range. There are as many antigen genes as the number of the function parameters, which their optimal value needs to be found to minimize or maximize the function. In other words, each gene represents a parameter value. For further explanation, let's assume that a function has six parameters, and the task is to find their optimal values, which will best minimize the objective function. In this case, the antigen can then be



x_1	x_2	x_3	x_4	x_5	x_6
2.53	0.57	0.07	-2.47	3.00	-0.33

FIGURE 3. A randomly generated antigen in decimal form, representing the weights and bias values of a neural network model.



Antibody 1	1.41	-2.91	-2.98	0.28	3.25	-1.33
Antibody 2	2.87	1.47	-3.44	-0.34	1.12	2.39
Antibody 3	0.51	-1.32	-0.77	1.46	1.93	3.01
Antibody 4	0.73	1.56	1.11	3.21	-2.45	1.87
Antibody 5	-0.68	0.54	2.09	1.39	-0.65	2.92

FIGURE 4. A random population of 5 candidate solutions (antibodies).

represented as a decimal vector containing six parameter values. These values are first initialized randomly in the predefined range (say in the range [-3.5, 3.5]), as shown in Figure 3.

2) ANTIBODIES INITIALIZATION

Similarly to how the human body generates several antibodies to combat an antigen, antibodies are generated as candidate solutions to combat an antigen in the CSA algorithm [73]. Antibodies and antigens have the same format. Figure 4 depicts a randomly generated population of antibodies (candidate solutions) in response to the antigen.

3) AFFINITY CALCULATION

Following the generation of the initial population of antibodies, the affinity (fitness value) of each antibody is calculated using a fitness function. The antibody evaluation is a crucial part of the CSA because antibodies are selected for cloning and mutation based on their fitness. The fitness function changes based on the problem being solved, and it must quantitatively measure how fit a particular solution. For instance, the fitness function can be defined by the reciprocal of the error, where the error is the absolute difference between the predicted value and the true value. The higher the fitness value, the better the antibody. In CSA, affinity calculation is performed repeatedly, and thus it should be sufficiently fast to compute. A slow computation of the affinity can negatively affect a CSA and make it exceptionally slow. Table 1 below shows the fitness values of all candidate solutions (antibodies) and the potential solution (antigen). Antibody2 has the highest fitness value, whereas Antibody4 has the lowest one.

4) SELECTION

We select a number of antibodies n with a higher fitness value than the antigen based on the previously calculated fitness

TABLE 1. Fitness of the antigen and the antibodies.

Antigen/Antibodies	Fitness value
Antigen	0.13
Antibody1	0.11
Antibody2	0.25
Antibody3	0.16
Antibody4	0.08
Antibody5	0.18

TABLE 2. Number of clones created from each selected antibody.

Selected antibodies	Fitness value	#Clones
Antibody2	0.25	$N_c = \text{round}\left(\frac{1\cdot 50}{1}\right) = 50$
Antibody5	0.18	$N_c = \text{round}\left(\frac{1\cdot 50}{2}\right) = 25$
Antibody3	0.16	$N_c = \text{round}\left(\frac{1\cdot 50}{3}\right) = 17$

value. As shown in Table 1, antibodies 2, 3, and 5 are more fit than the antigen and will thus be used for cloning and mutation. Antibodies that are less fit than the antigen will be eliminated from the population.

5) CLONING

Cloning is the process of producing multiple copies of the n antibodies chosen in the preceding step. Using a rank-based measure, the number of clones generated for each of the selected antibodies is proportional to their affinity. This is accomplished by first sorting the chosen antibodies in ascending order of affinity. The ordered list is then iterated, and the number of clones generated for each antibody is calculated as shown below [54]:

$$N_c = \text{round}\left(\frac{\beta \cdot N}{i}\right) \tag{1}$$

In Eq. (1), N_c is number of clones generated for a provided antibody, β is a clonal factor, N is the population size, i is the antibody current rank where $i \in [1, n]$, and round (.) is an operator that rounds towards the nearest integer. Table 2 shows the number of clones generated from each of the selected antibodies, assuming the population size ($N = 50$) and $\beta = 1$. The highest affinity Antibody2 ($i = 1$) will produce 50 clones, while the second highest affinity Antibody5 ($i = 2$) produces 25 clones, and finally Antibody3 ($i = 3$) will produces 17.

In some CSA implementations, the affinity proportionate cloning is not implemented, meaning that the number of clones created from each of the selected antibodies is the same. This implies that each antibody will be viewed locally and have the same clone size regardless of its affinity (fitness).

6) MUTATION

By randomly altering genes in a given antibody, mutation introduces diversity into a population. Mutation is a key component of the CSA, allowing for global optimization while avoiding local optimization. In CSA, the mutation is inversely proportional to an antibody's affinity. To put it another way, the higher the affinity, the lower the mutation, and vice versa. This is referred to as affinity maturation. The following method can be used to implement the mutation rate [74]:

$$\alpha = \frac{1}{\rho}\exp(-f) \tag{2}$$

In Eq. (2), α is the rate of mutation, ρ is a decay controller, and f is the affinity of the antibody normalized in [0,1]. Given L as the length of the clone, then the number of genes to be mutated N_M can be calculated as follow [75]:

$$N_M = [L \times \alpha] \tag{3}$$

There are many mutation operators that can be used depending on how antibodies are encoded. Here we list widely used mutation operators for real-valued encoded antibodies (i.e., weights and/or bias). This is not an exhaustive listing, and the CSA designer may find a combination of these methods or a problem-specific mutation operator more useful.

- **Uniform mutation:** This mutation operator replaces the value of the selected gene from the given antibody with a uniform random value selected between the upper and lower bounds specified by the user for that gene.
- **Boundary mutation:** This operator replaces the value of the selected gene from the given antibody with either upper or lower bound randomly.
- **Gaussian mutation:** This mutation operator is more efficient in converging than the previously mentioned operators. It adds a random value from a Gaussian distribution to the selected gene of an antibody. If the new new gene value falls outside of the user-specified range for that gene, then this new value is clipped.

7) AFFINITY CALCULATION AND STOPPING CRITERION

Following mutation, each clone is evaluated based on its affinity, and the best one is chosen as the antigen for the next generation. The process is repeated several times until the algorithm reaches the maximum number of generations and/or the best fitness value in the population does not improve after a certain number of generations.

IV. MULTI-LAYER PERCEPTRON

A MLP is a type of feed-forward neural network (FFNN) containing one or more hidden layers, where each layer has one or more neurons [76]. It is an extension of the perceptron network and is perhaps the most widely used neural network model. MLP with a single hidden layer is termed a shallow neural network; with a sufficient number of hidden neurons, a single hidden layer MLP can provide a universal approximation for almost any problem with tabular data. When MLP contains more than one hidden layer, it is called a deep neural network. Adding more hidden layers may yield little benefit. However, it is computationally more expensive as the number of trainable parameters increases and is more prone to overfitting [77]. An MLP with one hidden layer is shown in Figure 5. It consists of three layers of nodes: an input layer, a hidden layer and an output layer. The data are only transferred in a forward direction from the input nodes, through the hidden nodes and to the output nodes. Except for the input nodes, each node is a neuron that includes a bias neuron and performs some computations using a non-linear activation function [78].

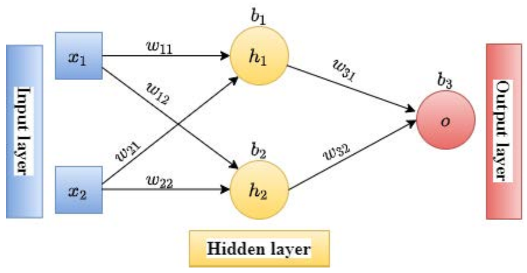


FIGURE 5. Multi-layer perceptron.

A. INPUT LAYER

The input layer is responsible for getting the inputs from an external source such as a CSV file. It requires one input node per variable (or feature). For example, the famous Iris Flower classification dataset contains four features (sepal-length, sepal-width, petal-length, and petal-width); hence, four nodes are in the input layer. In case two features out of four were only selected, then the input layer will include only two nodes.

B. HIDDEN LAYERS

The hidden layers are what making ANNs superior to other machine learning algorithms. The hidden layers are intermediate layers of neurons between the input and output layers. They detect and learn features by performing non-linear transformations of the inputs entered into the MLP. There is no precise general rule for selecting the number of neurons in a hidden layer; it is highly dependent on the problem. Therefore, programmers can use methodical experimentation to determine what works best for their particular dataset.

C. OUTPUT LAYER

The output layer is the final layer. It is responsible for producing the final output of the network. The number of neurons depends on the task format. For instance, in a binary classification problem (spam/ham), we use a single output neuron with sigmoid activation. However, in a multi-class classification problem, the number of neurons is usually equal to the number of classes/categories (e.g., three neurons for the three classes in the Iris Flower dataset). In this case, a softmax activation function is used to ensure the final probabilities sum to 1.

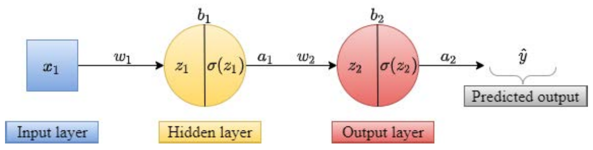


FIGURE 6. Simplified multi-layer perceptron.

D. BACKPROPAGATION

Backpropagation is a widely used algorithm to effectively train neural networks through a technique called chain rule. Backpropagation was first introduced in 1969 by Bryson and Ho [79] but was neglected due to its demanding computations. In 1986, the backpropagation learning method was rediscovered by Rumelhart *et al.* [80]. They described various neural networks where backpropagation works much faster than previous learning methods, making it desirable to apply neural networks to solve problems that had previously been unsolvable [81]. Currently, backpropagation algorithm is the workhorse of learning in ANNs, and a major component in modern deep learning models. [82].

As a general rule, training neural networks using back-propagation involves two passes: forward and backward [83]. In the forward phase, we first initialize all the parameters with small random values. Then, at each iteration, we feed the model with a training instance, and each neuron in the hidden and output layers determines its output in a way similar to Rosenblatt's perceptron except for the activation function where a nonlinear function such as sigmoid is used instead of a linear function (e.g., step function). Finally, we check what the neural network predicts and compute the error (or loss) using a certain cost function. In the backward pass, backpropagation performs a backward pass to calculate the gradients of the cost function with respect to all model parameters. After finding the gradients, we use stochastic gradient descent (SGD) optimizer to recalculate the new values of the parameters. The same forward and backward passes with the new updated parameters values are repeated again for the second training instance and so on. This training process ends when a predefined stop condition is met, i.e., the maximum number of iterations has been reached. To derive the backpropagation algorithm, we will consider the MLP model shown in Figure 6 where sigmoid function (σ) is used for both hidden and output layers. For the sake of simplicity, we assumed that each training instance contains only one feature (x_1). Accordingly, we considered only one neuron for the hidden layer and one neuron at the output layer. First, we start with the forward pass which involves finding the predicted output ($\hat{y}$) of the network:

$$\begin{aligned} z_1 &= w_1 x + b_1 \\ a_1 &= \sigma(z_1) \\ z_2 &= w_1 a_1 + b_2 \\ \hat{y} = a_2 &= \sigma(z_2) \end{aligned} \tag{4}$$

Next, we define the cost (or loss) function, which potentially could be any function that measures the error, such as the squared error. One of the most common loss functions used in classification problems is the Cross Entropy Loss. In a binary classification problem, where number of classes $\hat{C} = 2$, the Cross Entropy Loss can be defined as:

$$\begin{aligned} \mathcal{L}(w_1, w_2, b_1, b_2) &= -\sum_{i=1}^{\hat{C}=2} y_i \log(\hat{y_i}) \\ &= -y_1 \log(\hat{y_1}) - (1 - y_1) \log(1 - \hat{y_1}) \end{aligned} \tag{5}$$

where it's assumed that there are two classes: $\hat{C}_1$ and $\hat{C}_2$. y_1 and $\hat{y}_1$ are the true and the predicted score for $\hat{C}_1$, and $y_2 = 1 - y_1$ and $\hat{y}_2 = 1 - \hat{y}_1$ are the true and the predicted score for $\hat{C}_2$. Following that, we employ backpropagation to compute the derivative of the network's cost function ($\mathcal{L}$) with respect to all network parameters. This is performed by using the chain rule recursively from the last to the first layer of the network.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial \mathcal{L}}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} \\ \frac{\partial \mathcal{L}}{\partial b_2} &= \frac{\partial \mathcal{L}}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} \\ \frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial \mathcal{L}}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} \\ \frac{\partial \mathcal{L}}{\partial b_1} &= \frac{\partial \mathcal{L}}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} \end{aligned} \tag{6}$$

Finally, we use SGD to update the network's parameters.

$$\begin{aligned} w_1 &= w_1 - \eta \frac{\partial \mathcal{L}}{\partial w_1} \\ b_1 &= b_1 - \eta \frac{\partial \mathcal{L}}{\partial b_1} \\ w_2 &= w_2 - \eta \frac{\partial \mathcal{L}}{\partial w_2} \\ b_2 &= b_2 - \eta \frac{\partial \mathcal{L}}{\partial b_2} \end{aligned} \tag{7}$$

For the rest of this paper, the term backpropagation will be used loosely to refer to the entire learning algorithm for the MLP, including how the gradient is used by algorithms such as SGD to perform learning.

V. IMPLEMENTATION OF THE PROPOSED APPROACH

This section describes MLP training for data classification using our proposed CSA. Two prerequisites must be met in order to use the CSA: 1) a solution representation or encoding of the antigen; and 2) an affinity measure function to evaluate the solutions produced during the process. Once an encoding has been determined and an appropriate affinity measure function has been chosen, the CSA will perform selection, cloning, hypermutation, and re-selection based on the affinity until stopping criteria are met. The overall procedure of MLP training using our CSA-based method is depicted in Figure 7.

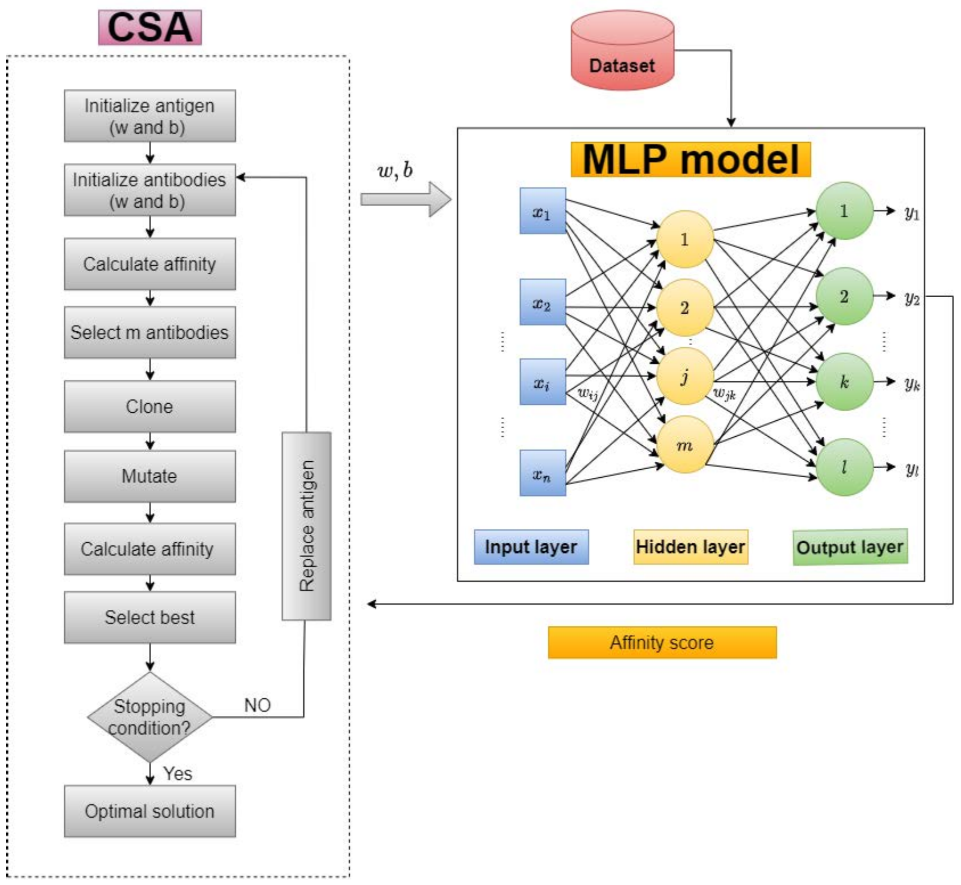


FIGURE 7. The procedure of the MLP training process with CSA based algorithm.

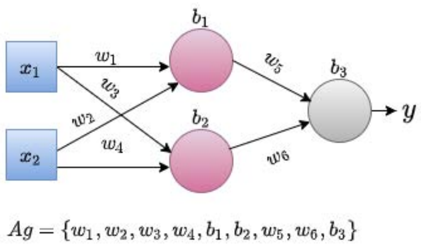
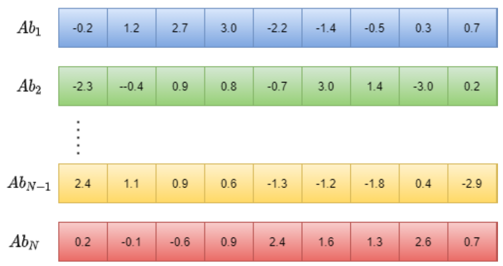


FIGURE 8. Encoding the weights and biases of MLP network in an antigen (Ag).



Ab_1	-0.2	1.2	2.7	3.0	-2.2	-1.4	-0.5	0.3	0.7
Ab_2	-2.3	--0.4	0.9	0.8	-0.7	3.0	1.4	-3.0	0.2
$\vdots$									
Ab_{N-1}	2.4	1.1	0.9	0.6	-1.3	-1.2	-1.8	0.4	-2.9
Ab_N	0.2	-0.1	-0.6	0.9	2.4	1.6	1.3	2.6	0.7

FIGURE 9. A random population of MLP networks with different weights and biases.

The following subsections V-A through V-H describe the complete procedure.

A. ANTIGEN ENCODING

To use the CSA to find the optimal set of weights and biases for MLP networks, we must first represent the problem domain as an antigen. Here, we want to find an optimal set of weights and biases for the MLP model shown in Figure 8. Initial weights and biases in the MLP network are chosen randomly within some range, i.e., [-3, 3]. The weights and biases can be represented by a 1D vector in which a decimal number corresponds to a particular weight or bias. In total, there are 6 weights and 3 biases in Figure 8. Since an antigen is a collection of genes, a set of weights and biases can be represented by a 9-gene antigen, where each gene corresponds to a single weight or bias.

B. ANTIBODIES

Antibodies (A_b) are created in response to the initial antigen initialized above. In simple terms, a random population of N antibodies (N number of MLP networks with different weights and biases) is created, as illustrated in Figure 9.

C. AFFINITY EVALUATION

The affinity of each generated antibody is evaluated according to the predetermined affinity function. This means training the MLP model with the training samples using the set of weights and biases determined by the antibody genes and then seeing how well it performs at classifying the test dataset. Since we are solving classification tasks, test classification accuracy can be used as the affinty function, which needs to be optimized or maximized. Test classification accuracy is defined as the ratio between the correctly classified samples and the total number of samples in the test dataset. The higher the accuracy, the higher the affinity.

D. SELECTION

Antibodies with higher affinity than the antigen are chosen to proceed to the cloning and hypermutation stages. Antibodies that have a lower affinity for the antigen will be eliminated from the population. This is a good approach for shortening the algorithm's execution time.

E. CLONING

In our proposed implementation, proportionate affinity cloning is not implemented. This means that, regardless of their affinity values, all selected antibodies will have the same clone size.

F. MUTATION

The Gaussian mutation is used on the cloned antibodies in this study. Gaussian mutation works well for real-value genes, as each weight and bias is encoded as a real value. Gaussian mutation makes small random changes in the antibodies in the population. It adds a random value from a Gaussian distribution to the chosen genes. For the CSA, the mutation is inversely proportional to the affinity of an antibody. Meaning, the higher the affinity, the fewer genes will be mutated or altered. This is known as affinity maturation. The goal is to preserve high-affinity antibodies without disturbing them while improving the affinity of low-affinity antibodies. When using Gaussian, there is a chance that some gene values will fall outside the specified range we set. Our implementation performs clipping on all genes after mutation to have all the gene values within the allowed range. Figure 10 depicts a mutation example.

G. AFFINITY EVALUATION OF THE MUTATED CLONES

The affinity of each mutated clone is calculated. The mutated clone with the highest affinity is selected, and the rest are eliminated.

H. STOPPING CRITERION

The selected clone replaces the original antigen and becomes the antigen for the next generation. The process is repeated from V-B until a criterion, such as a certain number of generations, is met.

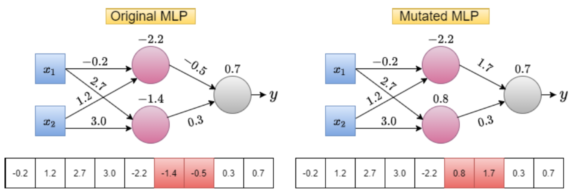


FIGURE 10. Mutation operation in CSA for MLP weights and biases optimization.

VI. EXPERIMENT DESIGN

A. DATASETS

1) IRIS FLOWER DATASET

The Iris Flower dataset is a multi-class (3-class) classification problem introduced by Ronald Aylmer Fisher [84]. It is considered the "hello world" dataset in machine learning and statistics. The dataset consists of 50 samples each for the three flower species, viz., Iris Setosa, Iris Versicolor, and Iris Virginica. It consists of 4 features measured in cm, namely, sepal length, sepal width, petal length, and petal width. The problem is to identify any iris flower category based on its four input characteristics of sepal length, sepal width, petal length, and petal width [85], [86].

2) SONAR DATASET

The Sonar dataset is a binary (2-class) classification problem developed by Terry Sejnowski in collaboration with R. Paul Gorman of the Allied-Signal Aerospace Technology Center [87]. The problem is to classify an object as a mine or rock. It contains 111 examples for the mine class obtained by bouncing sonar signals off a metal cylinder at various angles, and 97 examples for the rock class obtained from rocks under similar conditions. The dataset has 60 input features, with each feature representing the energy within a specific frequency band, combined during a certain duration.

3) WHEAT SEEDS DATASET

The Wheat Seeds dataset is a multi-class (3-class) classification problem that involves the classification of species given measurements of seeds belonging to three different varieties of wheat, namely Kama, Rosa, and Canadian. The number of samples for each class is balanced (70 for each), making 210 in total. It has 7 input variables (or features) that were constructed using a soft X-ray technique and the GRAINS package [88].

4) BREAST CANCER WISCONSIN DATASET

The Breast Cancer Wisconsin dataset is a binary (2-class) classification problem in which we attempt to predict one of two possible outcomes (benign or malignant). The dataset contains various measurements of breast tissue samples for cancer diagnosis. It contains measurements like the thickness of the clump, the marginal adhesion, the uniformity of cell size and shape, etc. The dataset was originally provided by Wolberg and Mangasarian [89] from the University of Wisconsin Hospitals in Madison. There are 569 cases of data where 357 cases belong to class 0 (benign), and the remaining 212 cases belong to class 1 (malignant). Therefore, the dataset is imbalanced and more challenging. The number of input variables or features to be used for this dataset is 30.

5) HABERMAN's SURVIVAL DATASET

The Haberman dataset is a binary (2-class) classification problem [90]. It includes cases from the University of Chicago's Billings Hospital's research on the survival status of patients who had undergone breast cancer surgery between 1958 and 1970. There are 306 data cases (or examples). 225 cases are classified as class 1 (the patient lived for 5 years or more), while the remaining 81 cases are classified as class 2. The dataset contains 3 input variables. The goal is to predict whether a patient will survive for 5 years or longer or die within 5 years after the surgery.

B. EXPERIMENTAL SETUP

The proposed CSA method for training MLP models and other algorithms are evaluated using the five classification datasets introduced above. Table 3 presents the specifications for each dataset. The number of input features, data examples, and classes are respectively presented for each dataset. Accordingly, the MLP architecture used by each dataset is defined prior to the training process, taking into account the dimensions of each dataset. Table 4 shows the MLP architectures for all datasets. The input layer for a dataset has one node per input feature. The output layer contains only one neuron with sigmoid as the activation function for the binary classification datasets. The domain of the sigmoid function is the set of all real numbers, $\mathbb{R}$, and is defined as follows:

$$\sigma(\vec{z}) = \frac{1}{1+e^{-z}} \tag{8}$$

Here, σ is the sigmoid and $\vec{z}$ is the input vector.

For multiclass classification datasets, the output layer contains one neuron per class with softmax as the activation function. In contrast to the sigmoid function, which takes a single input and assigns a number (the probability) ranging from 0 to 1 to whether it is a yes (positive), the softmax function can take multiple inputs and assign a probability to each one. The softmax activation function's equation is as follows.

$$S(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{c} e^{z_j}} \tag{9}$$

Here, S is the softmax, $\vec{z}$ is the input vector, e^{z_i} is the standard exponential function for the input vector, c is the number of classes in the dataset, and e^{z_j} is the standard exponential function for the output vector.

With respect to the hidden layer, it has little or nothing to do with data dimensions. To the best of the author's knowledge, there is no standard or accepted method for calculating the number of neurons in a hidden layer. As a result, we used systematic experimentation to determine the number of hidden neurons that work best for each dataset.

TABLE 3. Specification of the classification datasets.

Dataset	#Input features	#Data examples	#Classes
Iris Flower	4	150	3
Sonar	60	208	2
Wheat Seeds	7	210	3
Breast Cancer Wisconsin	30	569	2
Haberman's Survival	3	306	2

TABLE 4. MLP architecture for each dataset.

Dataset	Input layer	Hidden layer	Output layer
Iris Flower	4-nodes	11 neurons	3 neurons
Sonar	60-nodes	30 neurons	1 neuron
Wheat Seeds	7-nodes	25 neurons	3 neurons
Breast Cancer Wisconsin	30-nodes	25 neurons	1 neuron
Haberman's Survival	3-nodes	9 neurons	1 neuron

For all datasets, ReLU is used as the activation function in all hidden neurons. ReLU outputs the input directly if it is positive, otherwise it outputs zero. The ReLU formula is deceptively simple, as defined below.

$$R(z) = max(0, z) \tag{10}$$

Here, R is the ReLU and z is the input vector.

It should be noted that there is no computation involved in the input layer; thus, we have used the term nodes rather than neurons to represent the input layer. The input layer is simply a layer that receives input features.

Regrading the development enviroment, we used Jupyter Notebook with Python 3.9 to implement the CSA-based trainer and other trainer algorithms. The weights and biases of the MLP models are initialized randomly in the first iteration to small random values in the range of [-1,1]. Additionally, the input features of each dataset were scaled to lie between zero and one, as given in Eq. (11).

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{11}$$

where $x = (x_1, \ldots, x_n)$ and z_i is the i^{th} normalized input feature.

Furthermore, each algorithm includes a number of control parameters whose values must be carefully chosen. We experimented with various values for these parameters and chose the optimal values that produced the best performance. The goal is to find the best parameters for each algorithm in order to conduct a fair performance comparison. Table 5 shows each algorithm's control parameters. We can see that all nature-inspired algorithms have the same population size and run for 100 iterations. Other parameters are specific to each algorithm.

VII. RESULTS AND DISCUSSION

The algorithms' performance is evaluated using the 5-fold cross-validation method. As shown in Figure 11, each dataset

TABLE 5. Controlling parameters for the training algorithms.

Algorithm	Controlling parameter	Value
CSA	Population size	20
	Number of iterations	100
	Number of Clones	10
	Mutation rate	$\alpha = \frac{1}{\rho}\exp(-f)\ (\rho = 1.0)$
GA	Population size	20
	Number of iterations	100
	Crossover rate	0.9
	Mutation rate	0.3
	Selection method	Rank
ACO	Population size	20
	Number of iterations	100
	Pheromone update constant	20
	Pheromone constant	1
	Global pheromone decay rate	0.1
	Local pheromone decay rate	0.5
	Pheromone sensitivity	0.8
	Visibility sensitivity	2.5
PSO	Population size	20
	Number of iterations	100
	Local weight	0.5
	Global weight	0.3
	Inertia weight	0.9
HHO	Population size	20
	Number of iterations	100
	Initial energy	[-1, 1]
MFO	Population size	20
	Number of iterations	100
	Spiral factor	1
	Convergence constant	[-2, -1]
FPA	Population size	20
	Number of iterations	100
	Switch probability	0.8
BP	Number of iterations	1000
	Optimizer	SGD
	Learning rate	0.01
	Momentum	0.9

will be randomly split into 5-folds of approximately equal size. The first fold is used as a test set to compute a performance measure such as accuracy, and the remaining 4-folds are used as a training set to train the model. This approach generally results in a less biased estimate of the model and can be very useful in problems with a small number of data examples.

1) RESULTS FOR THE IRIS FLOWER DATASET

The MLP architecture for solving this dataset is $4 - 11 - 3$. Therefore, a fully-connected $4 - 11 - 3$ MLP will have $(4 \times 11) + (11 \times 3) + (11 + 3) = 91$ weights and biases. The goal of all algorithms is to find the optimal values for these weights and biases so that the Iris flowers can be correctly classified. Table 6 gives the average accuracy (AVG) and standard deviation (STD) for each algorithm obtained using the 5-fold cross-validation approach. As given in the table, CSA outperforms all other algorithms. The results of CSA follow by those of HHO, MFO, FPA, GA and BP. PSO and ACO show similar performance, but PSO has a lower standard deviation.

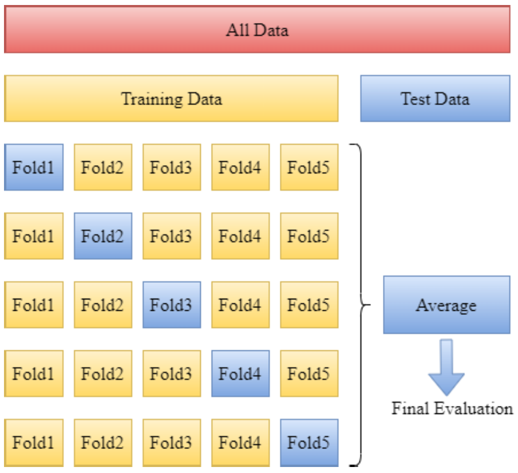


FIGURE 11. 5-fold cross-validation approach.

TABLE 6. 5-Fold cross-validation results for the iris flower dataset.

Algorithm	AVG (%)	STD (%)
CSA	98.89%	1.63%
GA	97.28%	2.94%
ACO	95.33%	4.98%
PSO	95.33%	2.66%
HHO	98.61%	2.36%
MFO	98.33%	2.39%
FPA	97.30%	2.49%
BP	96.66%	2.98%

2) RESULTS FOR THE SONAR DATASET

The MLP architecture for solving this dataset is $60 - 30 - 1$. Therefore, a fully-connected $60 - 30 - 1$ MLP will have $(60 \times 30) + (30 \times 1) + (30 + 1) = 1861$ weights and biases. Table 7 gives the average accuracy (AVG) and standard deviation (STD) for each algorithm obtained using the 5-fold cross-validation approach. As given in the table, CSA again outperforms all other algorithms. The results of CSA follow by those of HHO, MFO, FPA, GA, PSO, ACO and BP, respectively. Because the Sonar dataset is imbalanced and contains many more input features than the Iris dataset, all algorithms achieved lower results. Another reason is that the number of data examples for the Sonar dataset is very small, considering the high number of input features.

3) RESULTS FOR THE WHEAT SEEDS DATASET

The MLP architecture for solving this dataset is of $7 - 25 - 3$. Therefore, a fully-connected $7 - 25 - 3$ MLP will

TABLE 7. 5-Fold cross-validation results for the sonar dataset.

Algorithm	AVG (%)	STD (%)
CSA	82.85%	3.98%
GA	79.84%	5.27%
ACO	77.32%	6.30%
PSO	79.38%	3.68%
HHO	80.81%	4.66%
MFO	80.77%	4.87.66%
FPA	79.96%	5.15%
BP	76.46%	3.96%

TABLE 8. 5-Fold cross-validation results for the wheat seeds dataset.

Algorithm	AVG (%)	STD (%)
CSA	97.00%	2.9%
GA	95.44%	3.0%
ACO	94.90%	2.6%
PSO	95.20%	2.9%
HHO	96.80%	2.66%
MFO	96.20%	2.88%
FPA	95.87%	2.66%
BP	94.30%	2.9%

have $(7 \times 25) + (25 \times 3) + (25 + 3) = 278$ weights and biases. Table 8 gives the average accuracy (AVG) and standard deviation (STD) for each algorithm obtained using the 5-fold cross-validation approach. As given in the table, CSA again outperforms all other algorithms. The results of CSA follow by those of HHO, MFO, FPA, GA, PSO, ACO and BP, respectively.

4) RESULTS FOR THE BREAST CANCER WISCONSIN DATASET

The MLP architecture for solving this very challenging dataset is of $30 - 25 - 1$. Therefore, a fully-connected $30 - 25 - 1$ MLP will have $(30 \times 25) + (25 \times 1) + (25 + 1) = 801$ weights and biases. Table 9 gives the average accuracy (AVG) and standard deviation (STD) for each algorithm obtained using the 5-fold cross-validation approach. As given in the table, CSA outperforms all other algorithms slightly. The results of CSA follow by those of HHO, MFO, FPA, GA, PSO, ACO, and BP respectively.

5) RESULTS FOR THE HABERMAN's SURVIVAL DATASET

The MLP architecture for solving this very challenging dataset is of $3 - 9 - 1$. Therefore, a fully-connected $3 - 9 - 1$ MLP will have $(3 \times 9) + (9 \times 1) + (9 + 1) = 46$ weights and biases. Table 10 gives the average accuracy (AVG) and standard deviation (STD) for each algorithm obtained using the 5-fold cross-validation approach. As shown in the table, CSA outperforms all other algorithms. The results of CSA follow by those of HHO, MFO, FPA, GA, PSO, BP and ACO, respectively.

TABLE 9. 5-Fold cross-validation results for the breast cancer wisconsin dataset.

Algorithm	AVG (%)	STD (%)
CSA	98.24%	3.83%
GA	96.28%	3.84%
ACO	93.39%	3.86%
PSO	95.48%	4.46%
HHO	97.86%	3.84%
MFO	96.92%	3.88%
FPA	96.41%	3.84%
BP	92.08%	4.42%

TABLE 10. 5-Fold cross-validation results for the haberman's survival dataset.

Algorithm	AVG (%)	STD (%)
CSA	76.10%	1.60%
GA	74.66%	2.80%
ACO	72.20%	2.50%
PSO	74.56%	2.50%
HHO	75.90%	2.80%
MFO	75.50%	2.82%
FPA	74.76%	2.94%
BP	74.20%	3.6%

It is unlikely that any model developed using the Haberman's Survival dataset will generalize due to the small dataset's size and the fact that the data is based on diagnoses and operations for breast cancer that occurred many decades ago.

According to the experimental results on the five datasets, it is clear that CSA outperforms other algorithms, particularly the backpropagation algorithm. There are many motives why CSA should be considered rather than backpropagation to train MLP neural networks. Backpropagation relies on computing gradients and is highly sensitive to the initial values of the weights and biases, the learning rate, and the momentum. In some cases, a small change in any of these values has a significant impact on the predictive performance of the MLP network being trained. In contrast, CSA is a gradient-free optimization algorithm that can jump out of local minima or the weights and biases can be reinitialized to start looking in a new area of the search space, allowing it to find a good optimal solution if run long enough. The drawback, however, is that CSA and other nature-inspired algorithms often take longer than backpropagation to reach a solution. So, when the training time is reasonable, which can vary greatly depending on the problem, CSA is a better alternative MLP network training technique than backpropagation.

VIII. CONCLUSION

MLP networks are one of the most well-known and widely used machine learning algorithms. They have been widely applied in a variety of real-world applications, including medical diagnosis, electronic signal analysis, active and passive sonar target classification, seed and flower classification, and more. MLP network training is an optimization problem, and therefore, the optimization algorithm used is of primary importance. Backpropagation is the most widely used algorithm to train MLP networks. However, backpropagation is not ideal and often unable to find the global minimum. Optimization algorithms inspired by nature can be used to effectively train MLP networks. This research presents an efficient MLP training technique that employs CSA to improve the predictive accuracy of MLP when solving real-world problems. CSA involves selecting candidate solutions called antibodies based on affinity by matching against the primary antigen. The selected antibodies are cloned, and then undergo hypermutation inversely proportional to their affinity. Aside from the encoding paradigm used to represent the MLP's weights and biases as antibodies, our proposed methodology includes genetic operators for cloning and mutation to improve the MLP's ability to avoid getting stuck in local optima. The proposed CSA training method is tested on five real-world benchmark classification datasets of varying difficulty. Different MLP architectures are trained, taking into account the dimensions of each dataset. To validate the proposed CSA method's effectiveness in training MLPs, its performance is compared to that of BP and other six popular nature-inspired training algorithms, including GA, PSO, ACO, HHO, MFO, and FPA. We conduct systematic experimentation with a robust test harness in order to determine the optimal parameters of each algorithm that will provide the best possible performance for each algorithm. Experiment results show that MLP models provide better results in all five datasets when trained by CSA compared to other benchmark algorithms. This is due to the CSA's ability to avoid getting stuck in local optima. Therefore, it can be concluded that the proposed CSA method shows great promise for search and optimization problems, such as training neural networks.

IX. FUTURE WORK

The future direction of current research will focus on a thorough assessment of the CSA method for large benchmark datasets. Another potential future direction in this area is training two popular classes of deep learning neural networks: Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN). Both CNN and RNN are changing the way we communicate with the world. They are at the core of the deep learning revolution, powering a wide range of real-world applications such as self-driving cars, unmanned aerial vehicles, speech recognition, etc. Aside from the network architecture, which has been the primary focus of researchers' efforts to optimize, their performance is also heavily reliant on the training algorithm chosen to optimize the network weights and biases. Despite the fact that several optimizers have been proposed in the literature to address the shortcomings of traditional gradient descent approaches (e.g., SGD), there is still the possibility of getting trapped in local optima. Therefore, the future direction in this domain would be to propose efficient CSA methods for training deep neural networks in order to reduce the risk of falling into local optima. This can be achieved more easily than ever before due to the recent significant increase in processing power. A hybrid CSA and gradient descent (like Adam) model could also be developed in the future. The CSA can be used to find values for the initial weights and biases for the gradient descent algorithm, allowing it to avoid local optima and thus improve the predictive accuracy of the deep neural network being trained.

ACKNOWLEDGMENT

The authors are grateful to the anonymous peer reviewers for their insightful comments. Everyone's generosity and expertise have improved this manuscript in countless ways and saved them from numerous mistakes; those that inevitably remain are entirely their own responsibility.

REFERENCES

[1] M. Wahde, *Biologically Inspired Optimization Methods: An Introduction*. Southampton, U.K.: WIT Press, 2008.

[2] P. Adby, *Introduction to Optimization Methods*. Springer, 2013.

[3] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Philadelphia, PA, USA: SIAM, 2019.

[4] A. M. Hemeida, S. A. Hassan, A.-A.-A. Mohamed, S. Alkhalaf, M. M. Mahmoud, T. Senjyu, and A. B. El-Din, "Nature-inspired algorithms for feed-forward neural network classifiers: A survey of one decade of research," *Ain Shams Eng. J.*, vol. 11, no. 3, pp. 659–675, Sep. 2020.

[5] A. Mairaj, A. A. Bataineh, D. Kaur, and A. Javaid, "Identifying the optimal solutions of Bohachevsky test function using swarming algorithms," in *Proc. Int. Conf. Artif. Intell. (ICAI)*, 2019, pp. 109–115.

[6] X.-S. Yang, *Nature-Inspired Optimization Algorithms*. New York, NY, USA: Academic, 2020.

[7] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowl.-Based Syst.*, vol. 89, pp. 228–249, Nov. 2015.

[8] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, Aug. 2019.

[9] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Proc. Int. Conf. Unconventional Comput. Natural Comput.* Springer, 2012, pp. 240–249.

[10] A. A. Bataineh, A. Jarrah, and D. Kaur, "High-speed FPGA-based of the particle swarm optimization using HLS tool," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 5, pp. 1–7, 2019, doi: 10.14569/IJACSA.2019.0100502.

[11] S. A. G. Shirazi and M. B. Menhaj, "A new genetic based algorithm for channel assignment problems," in *Computational Intelligence, Theory and Applications*. Springer, 2006, pp. 85–91.

[12] A. A. Bataineh and D. Kaur, "A comparative study of different curve fitting algorithms in artificial neural network using housing dataset," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Jul. 2018, pp. 174–178.

[13] B. Zhang, Y. Wu, J. Lu, and K.-L. Du, "Evolutionary computation and its applications in neural and fuzzy systems," *Appl. Comput. Intell. Soft Comput.*, vol. 2011, Jan. 2011, Art. no. 938240.

[14] K. W. Chau, "Particle swarm optimization training algorithm for ANNS in stage prediction of Shing Mun river," *J. Hydrol.*, vol. 329, nos. 3–4, pp. 363–367, 2006.

[15] A. Jarrah, A. A. Bataineh, and A. Almomany, "The optimization of traveling salesman problem based on parallel ant colony algorithm," *Int. J. Comput. Appl. Technol.*, vol. 13, no. 1, pp. 1–19, 2022.

[16] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Let a biogeography-based optimizer train your multi-layer perceptron," *Inf. Sci.*, vol. 269, pp. 188–209, Jun. 2014.

[17] A. A. Bataineh and A. Jarrah, "High performance implementation of neural networks learning using swarm optimization algorithms for EEG classification based on brain wave data," *Int. J. Appl. Metaheuristic Comput.*, 2022.

[18] G. S. Shehu and N. Çetinkaya, "Flower pollination–feedforward neural network for load flow forecasting in smart distribution grid," *Neural Comput. Appl.*, vol. 31, no. 10, pp. 6001–6012, Oct. 2019.

[19] S. M. J. Jalali, S. Ahmadian, P. M. Kebria, A. Khosravi, C. P. Lim, and S. Nahavandi, "Evolving artificial neural networks using butterfly optimization algorithm for data classification," in *Proc. Int. Conf. Neural Inf. Process.* Springer, 2019, pp. 596–607.

[20] S. M. J. Jalali, M. Karimi, A. Khosravi, and S. Nahavandi, "An efficient neuroevolution approach for heart disease detection," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Oct. 2019, pp. 3771–3776.

[21] A. A. Bataineh and D. Kaur, "Immuno-computing-based neural learning for data classification," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 6, pp. 1–7, 2019, doi: 10.14569/IJACSA.2019.0100632.

[22] S. M. J. Jalali, S. Ahmadian, A. Khosravi, S. Mirjalili, M. R. Mahmoudi, and S. Nahavandi, "Neuroevolution-based autonomous robot navigation: A comparative study," *Cognit. Syst. Res.*, vol. 62, pp. 35–43, Aug. 2020.

[23] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. Int. Joint Conf. Artif. Intell.*, vol. 89, 1989, pp. 762–767.

[24] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, Aug. 1990.

[25] R. S. Sexton, R. E. Dorsey, and J. D. Johnson, "Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation," *Decis. Support Syst.*, vol. 22, no. 2, pp. 171–185, 1998.

[26] M. N. H. Siddique and M. O. Tokhi, "Training neural networks: Backpropagation vs. Genetic algorithms," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 4, Jul. 2001, pp. 2673–2678.

[27] Z.-G. Che, T.-A. Chiang, and Z.-H. Che, "Feed-forward neural networks training: A comparison between genetic algorithm and back-propagation learning algorithm," *Int. J. Innov. Comput., Inf. Control*, vol. 7, no. 10, pp. 5839–5850, 2011.

[28] K. Khan and A. Sahai, "A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context," *Int. J. Intell. Syst. Appl.*, vol. 4, no. 7, p. 23, 2012.

[29] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.

[30] R. Malviya and D. K. Pratihar, "Tuning of neural networks using particle swarm optimization to model MIG welding process," *Swarm Evol. Comput.*, vol. 1, no. 4, pp. 223–235, Dec. 2011.

[31] A. Roy, D. Dutta, and K. Choudhury, "Training artificial neural network using particle swarm optimization algorithm," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 3, 2013.

[32] R. C. Green, L. Wang, and M. Alam, "Training neural networks using central force optimization and particle swarm optimization: Insights and comparisons," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 555–563, Jan. 2012.

[33] C. Blum and K. Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification," in *Proc. 5th Int. Conf. Hybrid Intell. Syst. (HIS)*, Nov. 2005, pp. 1–6.

[34] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training," *Neural Comput. Appl.*, vol. 16, pp. 235–247, May 2007.

[35] A. Lanaridis, V. Karakasis, and A. Stafylopatis, "Clonal selection-based neural classifier," in *Proc. 8th Int. Conf. Hybrid Intell. Syst.*, Sep. 2008, pp. 655–660.

[36] B. H. Barbosa, L. T. Bui, H. A. Abbass, L. A. Aguirre, and A. P. Braga, "Evolving an ensemble of neural networks using artificial immune systems," in *Proc. Asia–Pacific Conf. Simulated Evol. Learn.* Springer, 2008, pp. 121–130.

[37] R. Pasti and L. N. de Castro, "Bio-inspired and gradient-based algorithms to train MLPs: The influence of diversity," *Inf. Sci.*, vol. 179, no. 10, pp. 1441–1453, Apr. 2009.

[38] H. Akbar, N. Suryana, and S. Sahib, "Training neural networks using clonal selection algorithm and particle swarm optimization: A comparisons for 3D object recognition," in *Proc. 11th Int. Conf. Hybrid Intell. Syst. (HIS)*, Dec. 2011, pp. 692–697.

[39] H. Chitsaz, N. Amjady, and H. Zareipour, "Wind power forecast using wavelet neural network trained by improved Clonal selection algorithm," *Energy Convers. Manage.*, vol. 89, pp. 588–598, Jan. 2015.

[40] S. S. Sammen, M. A. Ghorbani, A. Malik, Y. Tikhamarine, M. AmirRahmani, N. Al-Ansari, and K.-W. Chau, "Enhanced artificial neural network with Harris hawks optimization for predicting scour depth downstream of ski-jump spillway," *Appl. Sci.*, vol. 10, no. 15, p. 5160, Jul. 2020.

[41] W. Yamany, M. Fawzy, A. Tharwat, and A. E. Hassanien, "Moth-flame optimization for training multi-layer perceptrons," in *Proc. 11th Int. Comput. Eng. Conf. (ICENCO)*, Dec. 2015, pp. 267–272.

[42] Y. Ren, H. Li, and H.-C. Lin, "Optimization of feedforward neural networks using an improved flower pollination algorithm for short-term wind speed prediction," *Energies*, vol. 12, no. 21, p. 4126, Oct. 2019.

[43] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Process. Lett.*, vol. 17, no. 1, pp. 93–105, 2003.

[44] D. Karaboga, B. Akay, and C. Ozturk, "Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks," in *Proc. Int. Conf. Modeling Decisions Artif. Intell.* Springer, 2007, pp. 318–329.

[45] S. Mirjalili, "How effective is the Grey Wolf optimizer in training multi-layer perceptrons," *Appl. Intell.*, vol. 43, no. 1, pp. 150–161, 2015.

[46] H. Faris, I. Aljarah, N. Al-Madi, and S. Mirjalili, "Optimizing the learning process of feedforward neural networks using lightning search algorithm," *Int. J. Artif. Intell. Tools*, vol. 25, no. 6, Dec. 2016, Art. no. 1650033.

[47] H. Faris, I. Aljarah, and S. Mirjalili, "Training feedforward neural networks using multi-verse optimizer for binary classification problems," *Appl. Intell.*, vol. 45, no. 2, pp. 322–332, Sep. 2016.

[48] I. Aljarah, H. Faris, and S. Mirjalili, "Optimizing connection weights in neural networks using the whale optimization algorithm," *Soft Comput.*, vol. 22, no. 1, pp. 1–15, 2016.

[49] A. G. Karegowda, A. S. Manjunath, and M. A. Jayaram, "Application of genetic algorithm optimized neural network connection weights for medical diagnosis of PIMA Indians diabetes," *Int. J. Soft Comput.*, vol. 2, no. 2, pp. 15–23, 2011.

[50] S. Mirjalili, S. Z. M. Hashim, and H. M. Sardroudi, "Training feed-forward neural networks using hybrid particle swarm optimization and gravitational search algorithm," *Appl. Math. Comput.*, vol. 218, no. 22, pp. 11125–11137, Jul. 2012.

[51] C. Ozturk and D. Karaboga, "Hybrid artificial bee colony algorithm for neural network training," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2011, pp. 84–88.

[52] J.-F. Chen, Q. Do, and H.-N. Hsieh, "Training artificial neural networks by a hybrid PSO-CS algorithm," *Algorithms*, vol. 8, no. 2, pp. 292–308, Jun. 2015.

[53] B. P. Doppala, D. Bhattacharyya, M. Chakkravarthy, and T.-H. Kim, "A hybrid machine learning approach to identify coronary diseases using feature selection mechanism on heart disease dataset," *Distrib. Parallel Databases*, pp. 1–20, Mar. 2021.

[54] L. N. de Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Trans. Evol. Comput.*, vol. 6, no. 3, pp. 239–251, Jun. 2002.

[55] D. Dasgupta, "Artificial neural networks and artificial immune systems: Similarities and differences," in *Proc. IEEE Int. Conf. Syst., Man, Cybern., Comput. Cybern. Simulation*, vol. 1, Oct. 1997, pp. 873–878.

[56] L. N. Castro, L. N. De Castro, and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, 2002.

[57] L. N. De Castro, *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*. Boca Raton, FL, USA: CRC Press, 2006.

[58] J. D. Farmer, N. H. Packard, and A. S. Perelson, "The immune system, adaptation, and machine learning," *Phys. D, Nonlinear Phenomena*, vol. 22, nos. 1–3, pp. 187–204, Oct./Nov. 1986.

[59] H. Bersini and F. J. Varela, "Hints for adaptive problem solving gleaned from immune networks," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Springer, 1990, pp. 343–354.

[60] L. N. De Castro and F. J. Von Zuben, "The clonal selection algorithm with engineering applications," in *Proc. GECCO*, 2000, pp. 36–39.

[61] S. F. M. Burnet, *The Clonal Selection Theory of Acquired Immunity*, vol. 3. Nashville, TN, USA: Vanderbilt Univ. Press, 1959.

[62] J. Brownlee, "IIDLE: An immunological inspired distributed learning environment for multiple objective and hybrid optimisation," in *Proc. IEEE Int. Conf. Evol. Comput.*, Jul. 2006, pp. 507–513.

[63] G. L. Ada and S. G. Nossal, "The clonal-selection theory," *Sci. Amer.*, vol. 257, no. 2, pp. 62–69, Aug. 1987.
[64] J. Brownlee, "Clonal selection theory & CLONALG-the clonal selection classification algorithm (CSCA)," Swinburne Univ. Technol., Hawthorn, VI, Australia, Tech. Rep., 2005, p. 38.
[65] J. W. Kimball, *Introduction to Immunology*. New York, NY, USA: Macmillan, 1983.
[66] J. Brownlee, "Antigen-antibody interaction," Melbourne, Australia: Complex Intell. Syst. Lab., Centre Inf. Technol. Res., Fac. Inf. Commun. Technol., Swinburne Univ. Technol., Hawthorn, VI, Australia, Tech. Rep., 2007.
[67] J. Brownlee, "A review of the clonal selection theory of acquired immunity," Complex Intell. Syst. Lab., Swinburne Univ. Technol., Melbourne, VI, Australia, Tech. Rep., 2007.
[68] V. Cutello, G. Nicosia, M. Pavone, and J. Timmis, "An immune algorithm for protein structure prediction on lattice models," *IEEE Trans. Evol. Comput.*, vol. 11, no. 1, pp. 101–117, Feb. 2007.
[69] J. Kelsey and J. Timmis, "Immune inspired somatic contiguous hypermutation for function optimisation," in *Proc. Genetic Evol. Comput. Conf.* Springer, 2003, pp. 207–218.
[70] A. A. Bataineh and D. Kaur, "Optimal convolutional neural network architecture design using clonal selection algorithm," *Int. J. Mach. Learn. Comput.*, vol. 9, no. 6, pp. 788–794, Dec. 2019.
[71] J. Brownlee, *Clever Algorithms: Nature-Inspired Programming Recipes*. Jason Brownlee, 2011.
[72] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
[73] A. A. Bataineh and D. Kaur, "Immunocomputing-based approach for optimizing the topologies of LSTM networks," *IEEE Access*, vol. 9, pp. 78993–79004, 2021.
[74] V. Cutello, G. Narzisi, G. Nicosia, and M. Pavone, "Clonal selection algorithms: A comparative case study using effective mutation potentials," in *Proc. Int. Conf. Artif. Immune Syst.* Springer, 2005, pp. 13–28.
[75] S. W. Tan, S. C. Lee, and C. L. Chan, "Clonal-selection-based minimum-interference channel assignment algorithms for multiradio wireless mesh networks," in *Bio-Inspired Computation in Telecommunications*. Amsterdam, The Netherlands: Elsevier, 2015, pp. 287–321.
[76] A. A. Bataineh, "A comparative analysis of nonlinear machine learning algorithms for breast cancer detection," *Int. J. Mach. Learn. Comput.*, vol. 9, no. 3, pp. 248–254, Jun. 2019.
[77] A. A. Bataineh, A. Mairaj, and D. Kaur, "Autoencoder based semi-supervised anomaly detection in turbofan engines," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 11, pp. 1–7, 2020, doi: 10.14569/IJACSA.2020.0111105.
[78] A. S. A. Bataineh, "A gradient boosting regression based approach for energy consumption prediction in buildings," *Adv. Energy Res.*, vol. 6, no. 2, pp. 91–101, 2019.
[79] A. E. Bryson and Y.-C. Ho, *Optimization, Estimation and Control*. Peachtree City, GA, USA: Ginn Company, 1969.
[80] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.
[81] A. A. Bataineh, D. Kaur, and A. Jarrah, "Enhancing the parallelization of backpropagation neural network algorithm for implementation on FPGA platform," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Jul. 2018, pp. 192–196.
[82] M. A. Nielsen, *Neural Networks and Deep Learning*. San Francisco, CA, USA: Determination Press, 2015, 2018.
[83] M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*. London, U.K.: Pearson, 2005.
[84] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.
[85] E. Anderson, "The species problem in iris," *Ann. Missouri Botanical Garden*, vol. 23, no. 3, pp. 457–509, 1936.
[86] R. A. Fisher and M. Marshall, "Iris data set," *UC Irvine Mach. Learn. Repository*, vol. 440, p. 87, 1936.
[87] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Netw.*, vol. 1, no. 1, pp. 75–89, 1988.
[88] M. Charytanowicz, J. Niewczas, P. Kulczycki, P. A. Kowalski, S. Łukasik, and S. Żak, "Complete gradient clustering algorithm for features analysis of X-ray images," in *Information Technologies in Biomedicine*. Springer, 2010, pp. 15–24.
[89] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proc. Nat. Acad. Sci. USA*, vol. 87, no. 23, pp. 9193–9196, 1990.
[90] S. J. Haberman, "Generalized residuals for log-linear models," in *Proc. 9th Int. Biometrics Conf.*, 1976, pp. 104–122.

ALI AL BATAINEH (Member, IEEE) received the B.Sc. degree in computer engineering from Yarmouk University, Jordan, in 2010, the M.Sc. degree in computer engineering from the University of Bridgeport, CT, USA, in 2016, and the Ph.D. degree in electrical and computer engineering from The University of Toledo, OH, USA, in 2021. He is currently an Assistant Professor with the Electrical and Computer Engineering Department, David Crawford School of Engineering, Norwich University, VT, USA. His current research interests include the areas of artificial intelligence, machine learning, deep learning, computer vision, metaheuristic optimization, fuzzy logic, embedded systems, and FPGAs.

DEVINDER KAUR (Life Senior Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) in physics, majoring in electronics from Panjab University, in 1969 and 1970, respectively, the M.Sc. degree in medical physics from the University of Aberdeen, U.K., in 1976, and the M.Sc. and Ph.D. degrees in computer engineering from Wayne State University, USA, in 1985 and 1989, respectively. She was a Scientist with the Central Scientific Instruments Organization, a National Laboratory, Ministry of Science and Technology, Chandigarh, India, from 1971 to 1981. In 1989, she joined The University of Toledo as a Faculty, where she is currently a Full Professor with the Department of EECS. She has published upwards of 100 articles in refereed journals and proceedings of the international conferences. She has worked on projects funded by NSF, AFRL, Daimler Chrysler, and ROMAN Engineering. Her research interests include develop intelligent applications based on hybrid computational models using biologically inspired computing and fuzzy systems. She was a recipient of the IIT Delhi Fellowship, from 1970 to 1971, and the Fulbright Senior Specialist Award, in 2004. She visited the Nippon Institute of Technology Japan in that capacity. She was awarded the University Medal for obtaining the first rank in her B.Sc. and M.Sc. degrees from Panjab University. She was also received the Commonwealth Scholarship Award for her M.Sc. degree from the University of Aberdeen.

SEYED MOHAMMAD J. JALALI (Member, IEEE) received the M.S. degree in information technology major in advanced information systems from Allameh Tabataba'i University, Tehran, Iran, in 2016, and the Ph.D. degree from the Institute for Intelligent Systems Research and Innovation (IISRI), Deakin University, VIC, Australia, in 2021. He was a Research Assistant at the University of Massachusetts, MA, USA, conducting research in the field of artificial intelligence. He is currently a Research Fellow at Deakin University. His primary research interests include machine learning, deep neural architecture search, deep learning, and optimization. He has received the prestigious Deakin University Postgraduate Research Scholarship (DUPRS), in 2018. Besides, he is the Winner of the prestigious Alfred Deakin Postdoctoral Research Fellowship Award, in 2021.