A Hybrid Competitive Evolutionary Neural Network Optimization Algorithm for a Regression Problem in Chemical Engineering

Marius Gavrilescu 1,*, Sabina-Adriana Floria 1, Florin Leon 1 and Silvia Curteanu 2

1 Faculty of Automatic Control and Computer Engineering, “Gheorghe Asachi” Technical University of Iasi, 700050 Iasi, Romania
2 Faculty of Chemical Engineering and Environmental Protection, “Gheorghe Asachi” Technical University of Iasi, 700050 Iasi, Romania
* Correspondence: marius.gavrilescu@academic.tuiasi.ro

Abstract: Neural networks have demonstrated their usefulness for solving complex regression problems in circumstances where alternative methods do not provide satisfactory results. Finding a good neural network model is a time-consuming task that involves searching through a complex multidimensional hyperparameter and weight space in order to find the values that provide optimal convergence. We propose a novel neural network optimizer that leverages the advantages of both an improved evolutionary competitive algorithm and gradient-based backpropagation. The method consists of a modified, hybrid variant of the Imperialist Competitive Algorithm (ICA). We analyze multiple strategies for initialization, assimilation, revolution, and competition, in order to find the combination of ICA steps that provides optimal convergence and enhance the algorithm by incorporating a backpropagation step in the ICA loop, which, together with a self-adaptive hyperparameter adjustment strategy, significantly improves on the original algorithm. The resulting hybrid method is used to optimize a neural network to solve a complex problem in the field of chemical engineering: the synthesis and swelling behavior of the semi- and interpenetrated multicomponent crosslinked structures of hydrogels, with the goal of predicting the yield in a crosslinked polymer and the swelling degree based on several reaction-related input parameters. We show that our approach has better performance than other biologically inspired optimization algorithms and generates regression models capable of making predictions that are better correlated with the desired outputs.

Keywords: evolutionary algorithm; biologically inspired optimization; neural network optimization; imperialist competitive algorithm; regression model

MSC: 68W50

1. Introduction

Natural sciences and their related fields of research are essential for providing key answers to a wide range of real-world problems. Finding such answers nearly always involves an experimental phase, where data is collected by observing real-world phenomena or events, carrying out experiments within chemical and/or physical processes and noting the reaction outcomes, or using various sources and sensor arrays. Such methods generally produce large amounts of complex, multidimensional, and often unstructured data, which are difficult to interpret and make sense of.

In particular, chemical engineering processes present difficulties in modeling for several reasons. Often, the experiments are difficult to perform, being time-, material-, and energy-consuming. The mechanisms of complex processes are either unknown or not fully elucidated, which makes it difficult to apply material or energy balances specific to classic modeling. If mathematical models can be obtained, they are often based on approximations that affect their accuracy; they are complex models that are difficult to solve and, above all,
implement online. Thus, the use of “black box” models represents a beneficial approach for many situations in chemical engineering.

Consequently, a wide diversity of numerical processing methods has emerged over the years, which allow for structuring such data and deducing meaningful information from the underlying values. The end goal of such methods is to generate a mathematical or numerical model that describes the studied phenomenon/reaction/experiment as accurately as the experimental data will allow. It is very common for the task of interpreting experimental data to be reduced to a regression problem, where a relationship should be determined among one or multiple inputs and one or multiple observed outcomes. A frequent solution to such a problem is the development of a regression model that consists of a function that maps the inputs to the outputs, given that the outputs are real values from continuous domains. In instances where the data are of high dimensionality and complexity, a common approach is to search for a regression model by numeric optimization of a regression algorithm’s parameters. The choice in terms of an adequate algorithm is a difficult one, as many pitfalls exist given the problematic nature of experimentally gathered data. To this extent, neural networks have consistently proven their usefulness for solving complex problems such as those found in natural sciences. Sufficiently tuned and trained neural networks have demonstrated that they are capable of providing meaningful models of the relationships found within the inputs and outputs of experimental data. Such models allow for the generalization of the underlying phenomenon, given that enough effort was invested in searching for the right parameters that provide optimal convergence.

Neural networks can model complex relationships otherwise not possible with more basic alternatives. However, a neural network-based model requires searching through a complex parameter space in order to determine a set of values for the network hyperparameters and weights that provides the optimal solution for the problem at hand. Finding the right neural network architecture and the optimal weight values is often a tedious and time-consuming task, particularly for complex data sets, which themselves require a complex model to completely characterize them. Neural networks require extensive training, a comprehensive exploration of their hyperparameter space, and extensive validation before the right architecture and weights are found for a particular regression problem. Searching for the optimal neural network-based model is usually divided into two tasks: searching through the hyperparameter space of the network to find its optimal architecture and training the said architecture to find the optimal weight values. The former task, in particular, requires extensive computational resources and takes a lengthy amount of time before an optimal architecture is found, especially if an exhaustive exploration of the hyperparameter space is desired. Traditionally, searching through a neural network hyperspace is carried out using either a grid search or a random search. The former involves separately traversing through each hyperparameter domain by sampling each hyperparameter space axis into a finite number of parameter values and iterating through them in order. For each combination of parameter values, a new candidate model is trained and validated. In the latter case, random parameter values are generated from each corresponding hyperparameter domain, producing a candidate model for each random combination of hyperparameters to also be trained and validated. Ultimately, the model that minimizes the loss function or otherwise meets the desired convergence criterion is chosen. Grid searches are useful for systematically iterating through the hyperparameter space or a relevant portion of it, particularly in cases where an even sampling of the space is desired. A finer sampling of each axis yields more candidate models and a greater chance of finding the optimal one at the cost of more computational time and power required to validate a greater number of models. Random searches are useful when the hyperparameter space is too vast for a systematic grid search. The number of generated candidate models increases the probability of finding the optimal model; however, there is little guarantee that this will occur within a reasonable search time frame and a certain amount of “luck” is required for a good candidate model to be found. In both scenarios, a significant amount of time is required for a good neural network model to be identified.
For this reason, other search methods have been developed to more efficiently search through vast parameter spaces, of which the most notable are evolutionary algorithms. These are usually inspired by natural evolution, where animal and plant species sustain and propagate based on the "survival of the fittest" principle. Given a certain environment with certain characteristics, out of a population, only the individuals most suited to living in that environment survive and reproduce. Therefore, each generation statistically originates from the fittest individuals from the previous one. The evolution of organisms in the natural environment translates well to the evolution of candidate solutions to a given problem that can be mathematically modeled. A multitude of potential solutions that form a population of individuals is subjected within an evolutionary algorithm to mechanisms that mimic transformations occurring in the natural world, such as reproduction, gene crossover, and mutation. The fittest individuals are the ones who offer better solutions to the problem at hand and they are the ones who statistically propagate throughout the iterations of the evolutionary algorithm. For regression problems, finding the solution is often reduced to a minimization problem, where the optimal solution is the one that minimizes an error/loss/cost function. The fitness of a solution, therefore, translates to a lower error value, with the optimal solution being the one with the highest fitness, i.e., the lowest error value.

Evolutionary algorithms have consistently proven to be capable of solving difficult multidimensional problems due to their ability to efficiently explore vast complex solution spaces [1]. An evolutionary-based approach offers more room for experimenting with various random sampling, mutation, and crossover population combination functionalities, which means that they are highly configurable and customizable to specific problems of high complexity. This means that a carefully customized evolutionary algorithm often succeeds at locating the global minimum within a minimization problem and therefore finding the optimal solution, where alternative methods generally fail. Here we would also like to mention gradient descent-based solutions, which tend to get stuck in local minima and are difficult to use for problems with complex search spaces [2].

Considering the abovementioned context, this paper deals with a problem within the field of chemical engineering—the synthesis of polyacrylamide-based multicomponent hydrogels, with the goal of modeling the yield and swelling degree as functions of the reaction conditions. Consequently, the objective can be formulated as a multivariate regression problem, where we want to find a relationship/correlation between the inputs and the output variables. Despite the relatively small size of the available dataset, this regression problem has proven resilient to simple, commonly used regression methods. Therefore, we resort to finding a regression model based on optimizing a fully connected neural network that offers sufficient flexibility for exploring the complex solution space of our available data. In order to optimize the neural network, we propose an evolutionary algorithm that is based on the popular Imperialist Competitive Algorithm (ICA) [3], which we customize and modify so as to effectively explore the hyperparameter and weight space of the neural network and find the optimal architecture and weight values that should minimize the loss for our regression problem. Customization involves making modifications to the more essential steps of the traditional ICA to improve the convergence results, and, where possible, convergence speed. Furthermore, we combine the modified version of the ICA with a backpropagation-based approach by incorporating partial backpropagation training steps within the iterations of the modified ICA. As we describe in the subsequent sections, such an approach has proven to improve convergence effectiveness and speed and therefore lead to better neural network configurations. We use backpropagation as a boosting mechanism, which is meant to slightly “push” the solutions within an ICA iteration toward the optimum. The resulting hybrid algorithm generates neural network models that minimize the Root Mean Squared Error (RMSE) for our dataset to within an order of magnitude less than alternative algorithms while maintaining decent convergence times. We further improve the proposed approach by incorporating adaptive and self-adaptive strategies within the algorithm pipeline. These strategies form a two-part
parameter control mechanism: the parameters of the hybrid ICA are adapted along the
iterations of the algorithm using methods that are aimed at improving the convergence
results and speed; and along with the adapted phase, we employ a self-adaptive strategy,
where the parameters of the algorithm are incorporated into the chromosome of the candi-
date solutions, in which situation the algorithm searches through its own hyperparameter
space while also exploring the solution space of the neural network. This self-adaptive
strategy means that the hybrid ICA optimizes itself along with the neural network. Such
mechanisms added to the basic ICA have shown to improve the convergence results and
speed for our problem, as we detail in the subsequent sections.

Our contributions can be summarized as follows:

- We propose a variant of the ICA with alternative versions of some of its fundamental
  components. To this extent, we analyzed several versions of assimilation, revolution,
  and competition in order to find the ones that provide the best convergence for our
  problem.
- We test several initialization strategies in order to find the one that disperses the
  initial population as evenly as possible throughout the solution space. This allows
  us to generate initial populations that offer good coverage while having a relatively
  small individual count. Furthermore, we employ adaptive and self-adaptive strategies
to dynamically tune the hyperparameters of the evolutionary algorithm during its
  iterations.
- We incorporate a backpropagation-boosting mechanism into the iterations of the
  modified ICA, where each neural network candidate is slightly steered in the direction
  of the optimal solution using a gradient-based optimizer. This significantly contributes
to improving convergence and to minimizing the RMSE. The resulting combination
  of the modified evolutionary algorithm and the backpropagation-based optimization
  forms our hybrid method.

The paper is structured as follows. After the Introduction, Section 2 presents the
chemical engineering problem and the underlying mechanisms from which our data set
originates, as well as the relevant experimentally determined parameters that constitute the
inputs and outputs of our regression model. Section 3 presents the most significant results
from related state-of-the-art models, in terms of using evolutionary algorithms for general-
purpose problems, optimizing neural networks, and achieving good convergence results
for difficult regression problems. In Section 4 we present the modified hybrid ICA and
provide detailed descriptions of the steps involved in the ICA iterations, the modifications
made to the basic algorithm, the particularities of the hybrid aspect of our method, as well
as the adaptive and self-adaptive strategies used for parameter control. In Section 5 we
present the results of our work, demonstrate the benefits of our customized version of ICA
considering all aspects involved, and provide a comparison to other similar approaches
from the related literature in terms of convergence effectiveness and speed. The paper ends
with the Conclusions, where we provide a closing discussion related to our method and
highlight the limitations and possible future improvements of the algorithm.

2. The Case Study

Multicomponent hydrogels are materials characterized by a high swelling capacity
and possess special properties (mechanical, diffusion, and absorption), which make their
use in various domains possible —food, cosmetics, the pharmaceutical industry, medicine,
tissue engineering, agriculture, electrotechnics, electronics, etc.

For many applications, e.g., controlled-release systems, agrochemical products, multi-
component hydrogels are required to present a high capacity of biodegradation under the
action of the biologic fluids or the microorganisms present in the soil. In addition, these
materials present selective biodegradability under the action of gastrointestinal juices so
they can be used as a covering agent for tablets to protect the active principle, conceal the
non-agreeable taste and smell, as well as control the release of the active principle. Three-
dimensional networks based on polyacrylamide are used in ophthalmology as mechanical
protectors for the iris, retina, and corneal endothelia. These are just a few examples that justify the choice of this process for study through modeling and simulation.

From the point of view of modeling action, a high-molecular-weight polymer system represents complex classes of materials and is very difficult to model. Besides being highly nonlinear, there are a large number of parameters that need to be accurately defined if such systems are to be properly characterized. The relationships between the parameters being modeled and the actual behavior of these variables in the real world must be correlated as precisely as possible. However, in most cases, this is not possible and several approximations and simplifications are often made at various stages. Considering the lack of complete knowledge about the phenomenology of the process, the main reasons why a phenomenological model for the addressed process could not be developed were highlighted.

In some previous papers, we reported the synthesis and swelling behavior of semi- and interpenetrated multicomponent crosslinked structures based on polyacrylamide [4–6]. For this process, the yield in the crosslinked polymer and the swelling degree were determined as a function of the monomer (acrylamide) concentration, initiator concentration, crosslinking agent (formaldehyde) concentration, amount of inclusion polymer (starch, poly(vinyl alcohol) (PVA), gelatin), temperature, and reaction time. The experimental results containing 177 data are given in Curteanu et al. [4].

The results obtained using the previous approaches were encouraging but susceptible to error. Consequently, in the current research, improved methods are proposed for modeling the main property of hydrogels, namely the degree of swelling, along with the efficiency of the process.

Seven input variables were considered: CM (monomer concentration), CI (initiator concentration), CA (crosslinking agent concentration), PI (amount of inclusion polymer), T (temperature), t (reaction time), and type of included polymer codified as 1—no polymer added, 2—starch, 3—PVA, and 4—gelatin. The two outputs were \( \eta \) (yield in the crosslinked polymer) and \( \alpha \) (swelling degree). Thus, the neural network modeling established the influence of the initial conditions on the reaction yield and swelling degree.

The predictions of the two outputs are useful in practice because they are related to process efficiency (yield) and they can replace experiments that necessitate a great number of materials and, especially, time (a determination of the swelling degree takes around 20 days).

3. Related Work

Neuroevolution (NE) is a procedure in which an evolutionary algorithm (EA) is used to optimize an artificial neural network and is an alternative to classical training (i.e., gradient-based algorithms). It can also evolve the hyperparameters of neural networks, such as the number of hidden layers, the number of neurons in each layer, biases, and the activation functions. Interestingly, neuroevolutionary methods achieve promising results using simple evolutionary algorithms [7], and proposing new strategies for these simple algorithms can significantly improve the convergence of training a neural network and its prediction performance. Although neuroevolutionary methods are generally computationally expensive, especially in deep learning [8], they are still preferred because they also offer flexibility in choosing the cost function (e.g., reward maximization [9]).

Various studies have been carried out in the literature on small-sized neural networks with fixed topologies in addition to studies that have addressed the evolution of neural network architecture.

Three neural network models, namely Multi-Layer Perceptron (MLP), Recurrent Neural Network (RNN), and Evolutionary Neural Networks (Neuroevolution: MLP-ABC), were used in [10] to predict the output of a photovoltaic panel. The authors used the Artificial Bee Colony (ABC) algorithm to optimize the neural network weights. The experimental results showed that the MLP-ABC model provided the best results.
In [11], four evolutionary algorithms, i.e., Multi-Verse Optimizer (MVO), Moth–flame optimization (MFO), Cuckoo Search (CS), and Particle Swarm Optimization (PSO), were used to train and optimize MLPs. The trained MLPs were then used to navigate an autonomous robot. The authors proposed two strategies for avoiding the local optimum, which were applied to each algorithm. In the first proposed strategy, 20% of the worst solutions were reinitialized at each iteration. The second strategy consisted of a mutation operator that randomly changed the genes of some solutions with a 20% chance. The experimental results showed that MFO had the lowest mean square error value and the fastest convergence speed. Also, MFO had a superior ability to avoid local optimum and good optimization efficiency. PSO, in particular, has seen extensive use for neural architecture search with applications for convolutional neural networks [12,13], deep belief networks [14,15], and autoencoders [16,17]. In most cases, the encoding of the particles only encompassed the hyperparameters of the neural networks, leaving the tuning of the optimal architecture to a more traditional backpropagation-based approach.

In [18], the authors solved an electrical load problem using a 1D convolutional neural network tweaked using a variant of Enhanced Grey Wolf Optimization (EGWO). Like many other evolutionary algorithms, this optimizer draws inspiration from the behavior of an animal in its natural habitat, specifically, the hunting behavior of wolves. The solution candidates were divided into hierarchical categories (such as alpha and beta individuals) and they interacted throughout the solution space via a mathematical model that simulated a pack-like behavior. Aside from the actual optimization, the authors noted the fast convergence of this meta-heuristic model and the low requirements in terms of computational resources.

The Cellular Genetic Algorithm (CGA) was used in [19] to find optimal weights of MLP to classify medical data accurately. The authors proposed a specially designed crossover operator called Damped Crossover (DX) that used information related to the best solution in the current iteration and the stage of the evolutionary process. The DX operator had a greater influence on the changing variables at the beginning of the evolutionary process and a reduced influence at the end of the iterations (i.e., the operator was more accurate at the end of the evolutionary process).

The Biogeography-based Optimization algorithm (BBO) was used in [20] to train the MLP to classify the sonar dataset, a high-dimensional problem. To improve the exploration ability of BBO, the authors used various mutation operators based on Gaussian mutation, Cauchy mutation, and exponential mutation and then proposed a novel mixed mutation strategy called Neighborhood Search Trainer (NST). The NST strategy consisted of a combination of Gaussian, Cauchy, and exponential mutations.

In [21], six neuroevolutionary classification techniques were used for the slope/failure stability assessment problem. The MLP was trained with Ant Colony Optimization (ACO), BBO, Evolutionary Strategy (ES), Genetic Algorithm (GA), Probability-based Incremental Learning (PBIL), and PSO to improve classification accuracy in the stability assessment. The experimental results showed that the MLP trained with the BBO algorithm obtained the best classification accuracy.

Two optimization techniques, i.e., GA and Binary Particle Swarm (BPS) optimization, were used in [22] to improve the predictive power of credit risk scorecards. GA and BPS were used to find the optimal architecture of an MLP along with activation functions, whereas the weights were optimized with the backpropagation algorithm. In terms of predictability, the two optimization techniques outperformed the logistic regression and a default neural network, but GA was more time-consuming than BPS.

In [23], a genetic algorithm was used with the ADAM optimizer to optimize the architecture and parameters of small neural networks. The problem addressed was the compact modeling of MOSFET devices using neural networks. Due to the requirements of requiring a compact model, the optimization problem consisted of finding a neural network of a small size that provided the most accurate answer. The authors used a genetic algorithm to find the optimal topology of the neural network, and ADAM was used to optimize the
weights and biases. The architecture of a neural network is defined using blocks of neurons, and how these blocks are connected can lead to partially connected neural networks. By limiting the number of connections between blocks of neurons, the authors minimized the genetic algorithm’s search space and improved the network training’s convergence speed.

Neuroevolution has also been successfully used in modeling PID controllers [24]. The authors used an MLP neural network with two hidden layers in a closed-loop feedback control to replace a PID controller. The chosen neural network, i.e., the neurocontroller, was subject to unsupervised learning because the optimal behavior of the developed controller was unknown. The training consisted of using a genetic algorithm to optimize the weights and biases of the neurocontroller so that specific closed-loop performance indices were minimized. The experimental results showed that the neurocontroller provided significantly better results than a linear PID controller.

A new algorithm, Evolutionary eXploration of Augmenting Memory Models (EXAMM), was proposed in [25] to evolve recurrent neural network (RNN) architectures with various cell types to perform the prediction of large-scale, real-world time-series data from the aviation and power industries. The EXAMM algorithm was further used in [26], where a novel speciation strategy based on extinction and repopulation events was proposed (specific strategy for island-based evolutionary algorithms).

Differential evolution (DE) is a popular approach used to optimize the hyperparameters of a wide variety of neural networks. In [27], the authors used a DE algorithm to optimize a pi-sigma neural network. This network drew from traditional MLPs and extended them by incorporating higher-order combinations of the inputs. The authors noted that, although such networks may lead to better solutions for certain complex problems, the added operations also increase the complexity of the network architecture and, consequently, the resulting hyperparameter space. A DE-based approach was, therefore, chosen to optimize the neural network to solve various forecasting problems, with accuracies reported as being higher than those obtained using simpler network structures. In [28], the authors optimized a neural network using differential evolution in order to tackle a lithology identification problem from a geological field. They reported significantly better accuracies compared to more traditional classification algorithms. At the same time, the DE-based approach proved to be able to explore the solution space much more effectively than using a more conventional exhaustive hyperparameter search.

Another category of problems where DE-based optimization is sometimes employed is medical image processing. The authors from [29] presented a technique for medical image fusion using a deep neural network derived from the Inception architecture. Unlike other applications where the objective is to search for the optimal hyperparameters of the network, in this case, a DE algorithm was used to search the feature map space for the best features to be used within the fusion pipeline. In [30], the authors developed a classification model for medical image-based diagnoses using an evolutionary algorithm to optimize the pruning strategy of Generative Adversarial Networks (GAN). Notably, the resulting model offered similar accuracies to the unpruned version, albeit using significantly reduced computational resources. Other significant results in the above-mentioned approaches focused on using DE and/or bio-inspired optimizers for handling global optimization problems [31–33].

Some notable results handled problems involving supply chain management and the related decision-making strategies using multiple exact optimization methods [34–39].

Another form of neural network hyperparameter optimization is cooperative neuroevolution, which has been successfully used in training RNNs for chaotic time-series problems (e.g., signal processing, finance, weather forecasting, etc.) [40,41]. Cooperative neuroevolution is more effective in predicting time-series problems than backpropagation through time [40] or other standard evolutionary algorithms [41]. In this type of optimization, a neural network is decomposed into subcomponents divided into different subpopulations. Initially, the individuals in the created subpopulations do not have fitness because an individual represents only a subcomponent of a neural network. The evaluation
of an individual begins with sampling other individuals from the other subpopulations. Afterward, all the chosen individuals are concatenated to reconstruct a neural network and the fitness can be computed.

An increased interest in neuroevolution has also been seen in other fields, such as medicine [42–45], chemistry [46,47], education [48], and games [49,50].

4. Method Description

Our approach focused on finding the optimal neural network architecture and weight values that constitute the best model for describing the aforementioned regression problem. Specifically, the method was intended for searching the combined hyperparameter and parameter spaces of a fully connected neural network to find the best values resulting in the optimal model. For this purpose, we developed an evolutionary algorithm that searched through the network parameter space for the optimal parameter values as quickly and efficiently as possible.

Throughout the paper, we use well-known and understood terminology with regard to the key terms that define the architecture and functionality of a fully connected neural network. The number of hidden layers and the size of each hidden layer in terms of neuron count are hyperparameters, whereas the weights and biases of the network are simply its parameters. We differentiate the hyperparameters of the neural network from the hyperparameters of the evolutionary algorithm, which are the various variables that influence the functionality of the evolutionary method. Among these are the parameters of the various operators applied to the candidate solutions during the iterations of the algorithms, for example, the mutation probability or the assimilation distance.

Our methodology involved combining the hyperparameters and parameters of the neural network into a single network parameter space. Consequently, searching for the optimal neural network involved simultaneously optimizing the architecture of the neural network and its weights and biases, unlike traditional search-train-validate-test approaches, where designing the neural network is a separate task from training it. Therefore, a solution from the perspective of an evolutionary optimizer is a complete set of neural network parameters. The optimal solution, which is the goal of the search, is the set of neural network hyperparameters and parameters that fit the convergence criterion of the problem at hand. In our case, we aimed to find the neural network that minimizes the RMSE of the network predictions from the inputs of the dataset instances and the actual outputs from the same instances. In evolutionary optimization language, a potential solution is encoded as a chromosome containing the full set of neural network parameters, whereas the fitness of an individual with a specific chromosome is the inverse of the RMSE resulting from evaluating the corresponding neural network on the dataset corresponding to the regression problem. Consequently, in our implementation, we considered that the fitter individuals were the ones that generated lower RMSE values, or, in more general terms, that had a lower cost.

Our proposed neural network optimizer was based on the Imperialist Competitive Algorithm, to which we applied multiple modifications intended to improve the convergence results and, where possible, convergence speed. In order to achieve this goal, we implemented alternative versions of the fundamental ICA steps, incorporated a backpropagation boosting mechanism that significantly improved convergence, and added parameter control strategies meant to tweak and optimize the algorithm hyperparameters during execution. We refer to the resulting algorithm as a hybrid of an evolutionary algorithm and backpropagation-based optimization, which leverages the advantages of both approaches to generate a neural network solution that is as close as possible to the optimal one. As we shall illustrate in the subsequent sections, the behavior of this approach is twofold:

- The evolutionary algorithm has the advantage of thoroughly searching through the parameter space of the neural network for a solution that minimizes the RMSE. As we discovered after multiple attempts, the neural network space was sufficiently
complex and had a high-enough dimensionality to make it difficult for the evolutionary algorithm alone to explore it and settle in a global minimum.

- A backpropagation component was incorporated into the evolutionary algorithm’s pipeline and significantly contributed to proper convergence. With each iteration, its role was to “steer” the solutions in the direction of the optimum. Essentially, this component involved partially training each neural network solution in a backpropagation manner using gradient-based optimization.

4.1. Neural Network Encoding

As with most evolutionary algorithms, the solutions were represented via a 1D vector of genes forming the solution’s chromosome. For our purposes, we required that each neural network candidate be represented in the same manner. We achieved this by re-organizing the parameters and hyperparameters into a “flattened” version of the neural network, where the parameter values were arranged as a one-dimensional array. The resulting two-part chromosome had the following structure:

- A header containing the hyperparameters—the number of hidden layers followed by the hidden layer sizes in order from the input to the output layers.
- A larger body that contained the weights and biases in the same order as found in the neural network layers—first, the flattened weight matrix between the input and first hidden layer, followed by the bias vector of the input layer, followed by the next weight matrix and the corresponding bias vector, and so on.

Note that we used constant-size chromosomes; therefore, the size of each chromosome was the highest possible number of neural network parameters. For networks with fewer parameters than the maximum, the extra values were simply ignored. We found no advantage in using variable-size chromosomes, the only significant difference being a more complex implementation.

The encoding procedure of a neural network is illustrated in Figure 1. The general format of the chromosome shown in Figure 1a is presented for neural networks with a maximum of two hidden layers. The length of the chromosome was set to accommodate the largest possible network with the highest parameter count. For example, if we searched for a network with a maximum of 2 hidden layers and a maximum hidden layer size of 10, considering 7 inputs and 2 outputs, the size of the chromosome would be 3 + 7 \times 10 + 10 + 10 \times 10 + 10 + 10 \times 2 + 2 = 215, which accommodates a header size of 3 (the number of hidden layers and the size of each hidden layer) and the values of the weights and biases for the maximum-sized network. While searching for the best neural network configuration, we represented any potential network candidate via the maximum-sized chromosome. In cases where the optimal neural network had fewer parameters than the maximum, the “extra” values from the chromosome were simply ignored when reconstructing the corresponding neural network. For example, if the optimal neural network had a single hidden layer, the third value from the chromosome (which represents the size of the second hidden layer) was ignored, despite it being present in the chromosome structure. Likewise, a network with a single hidden layer would have fewer parameters than the maximum count. In this case, only the necessary values for rebuilding the neural network were used and the other trailing ones were ignored. This allowed for the use of constant-sized chromosomes within our evolutionary algorithm, which simplified the implementation to a certain extent. We also experimented with variable-size chromosomes, where each flattened neural network was sized according to the actual number of network parameters. In our experience, there was no noticeable benefit to this approach. Considering that our population was rather small (100 initial solutions), using fixed-size chromosomes did not significantly impact memory use and computational requirements. Conversely, using variable chromosome sizes complicated the implementation, particularly with regard to the adaptive sampling strategy, which is discussed later in this paper. Consequently, the population was formed from fixed-size solutions, where each corresponding chromosome was large enough to store the values of any neural network candidates.
Figure 1. Arranging (flattening) the parameters of a neural network into a 1D array forming a chromosome. The flattening method is illustrated for neural networks with a maximum of two hidden layers but is easily adaptable to larger architectures. (a) The general format, regardless of the actual neural network architecture; (b) An example of flattening a single hidden layer network into a chromosome.

4.2. The Evolutionary Competitive Algorithm

The evolutionary component of our method was an enhanced version of the Imperialist Competitive Algorithm (ICA), to which we made several modifications and additions in order to improve convergence in our dataset (Figure 2). The main task of the evolutionary algorithm was to explore the parameter space of a fully connected neural network and find a neural network architecture and weight values that come as close as possible to the optimum. Consequently, in the context of the evolutionary algorithm, a solution was a potential neural network configuration, represented by the configuration described in Section 4.1. The optimal solution was the network configuration that minimized the RMSE on our dataset, which could in more general terms be thought of as the cost of the solution. Therefore, the evolutionary algorithm searched through the solution space for the minimum-cost solution. The ICA was modeled on the interaction among countries and empires in the course of which the empires compete for dominance. The solutions were considered countries. Some of the solutions were imperialist countries, around which the other countries, known as colonies, formed groups known as empires. Thus, the population was formed from a few of the better solutions, the imperialist countries, which owned the other solutions, the colonies. Each imperialist country competed for the accumulation of as many colonies as possible until there was one dominant empire. Such interaction should ultimately lead to the optimal solution while avoiding the pitfalls of complex, difficult-to-explore solution spaces with multiple local minima. Although it is not the purpose of this paper to describe the ICA in detail, we loosely outline the most significant steps of the classic algorithm:

- As with any evolutionary algorithm, the population is initialized according to a specific sampling policy. Empires and their colonies are also initialized.
- The countries are subjected to an assimilation step, where each empire strengthens its hold on its colonies. Each colony moves closer to its imperialist owner, thereby becoming more similar to it.
Some of the colonies undergo a revolution step, which is equivalent to a mutation phase.

Some of the empires undergo a change in leadership. If an imperialist country has a higher cost than one of its colonies, the respective colony becomes the leader of the empire, i.e., the imperialist country and the better colony swap positions.

The empires compete for dominance. Empire competition can take several forms, but in the simplest case, each empire takes the weakest colony from the weakest empire. This step eventually leads to empires without colonies, which are eliminated (either removed from the population entirely or reverted to a regular colony).

Once the new empire grouping is established, a convergence criterion is tested. Commonly, the algorithm stops when there is only one empire left. Otherwise, a new iteration begins, involving assimilation, revolution, competition, and all smaller in-between operations.

Our main modifications to the standard ICA loop involved the following three components:

- Alternative initialization, assimilation, revolution, and competition steps. We analyzed several potential versions of these steps and ultimately used the ones that performed the best on our dataset.
- A backpropagation-based boosting step. This involved slightly guiding the solutions of each iteration toward the optimum using a mechanism similar to the backpropagation-based training of a neural network. As such, at each iteration, the weights of each solution were modified via backpropagation to ease convergence toward the optimal weight values.
- An algorithm hyperparameter optimization strategy based on a mixed adaptive/self-adaptive parameter control mechanism.

Figure 2. Diagram illustrating the steps of our algorithm: The country population was initialized using Maximin Latin Hypercube Sampling, an approach chosen out of several others. Then, at the start of each ICA iteration, backpropagation boosting was applied in order to steer the solutions toward the optimum. The main loop largely followed the same progression as the traditional ICA; however, the fundamental steps (assimilation, revolution, competition) were chosen out of several possible versions. The choices were made following an analysis to determine the steps that best suited our algorithm in terms of effectiveness, considering the problem described in Section 2.
4.3. Sampling Methods for Population Initialization

In order to improve upon the classic ICA, we analyzed several modified versions of some of the fundamental ICA steps to find the ones that best worked for our purposes. We mainly targeted initialization, revolution, assimilation, and competition methods as candidates for potential improvements.

Initialization is a frequently discussed and disputed aspect of evolutionary algorithms, which are notorious for being overly sensitive to the configuration and distribution of the initial population [51]. The main pitfall with regard to the initialization is the disproportionality between the size of the solution space and the size of the initial solution population. It is desired that the initial population be as representative as possible for its corresponding value-domain. Although, theoretically, a larger population would better cover this space, a practically large initial population would significantly impact performance, not necessarily guaranteeing a better convergence to the optimal solution. In almost all scenarios, the size of the initial population is orders of magnitude smaller than would be required to fully cover the problem space.

Rather than having to rely on large initial populations for the initialization stage, we aimed for small populations where the solutions were spread as evenly as possible throughout the problem space. The most common initialization was carried out using a uniform distribution, which did not generate evenly spread solutions [52]. For this reason, along with uniform generation, we considered several sampling strategies that could potentially generate initial populations with much better coverage of the problem space. By “better coverage” we mean that the initialization relied on a sampling method where the generated samples were spread as evenly as possible throughout their domain so that the generated sample set was as representative as possible of the entire problem space. Such a property is commonly expressed as the discrepancy in the sampling method. A low-discrepancy approach has the property that any number of samples that fall into a certain hypercube is nearly proportional to the measure of the hypercube. Roughly, this means that a low-discrepancy sampling method produces a sample set of almost-even density, whereas a high-discrepancy method has a greater chance of generating samples that are locally grouped, as well as large regions void of samples. Low-discrepancy sampling is a widely discussed topic in related fields [53] and is beyond the scope of this paper. For our purposes, we tested a uniform initialization, as well as several other initialization approaches that are known to rely on low discrepancy sampling:

- Halton sampling, which generates a population from a Halton sequence by applying the radical inverse function to integers expressed in various bases. Halton sequences have been demonstrated to produce low-discrepancy pseudo-random point sets despite generating these points in a fully deterministic manner [54]. Due to the high dimensionality of our solutions, we used a scrambled Halton sequence as our generator, where the actual sequence was built using permutations of the coefficients corresponding to the standard sequence.

- Classic Latin Hypercube sampling (LHS), which spaces the solutions from one another so that no two solutions are found in the same axis-aligned hyperplane within the solution space [55]. In an equivalent 2D space, no two points generated via LHS would be found on the same row or column.

- Centered Latin Hypercube sampling (CLHS): for each generated point, a hypercube may be defined so that each point is centered within its respective hypercube.

- Maximin Latin Hypercube sampling (MLHS), which generates a point set so that, in addition to the standard LHS criterion, the points are further away from each other to maximize the minimum distance among them.

All four non-uniform sampling methods demonstrate low discrepancy and are reasonably capable of generating evenly distributed solutions within the solution space. Figure 3 illustrates this principle for a 2D space, where it can be seen that uniform sampling generally does the poorest job at “filling out” the problem space with the available points, resulting in regions of significantly varying densities (Figure 3a), whereas the other approaches
generate points that provide comparatively better coverage of their spaces (Figure 3b–e). In our experience, the low-discrepancy initialization methods were better suited to our approach than standard uniform initialization, although there was little difference among the methods themselves. We did however notice a slight improvement when using MLHS, which we ultimately chose as the default initialization method for our algorithm.

**Figure 3.** Tested sampling methods exemplified by generating 30 points in a 2D point space: (a) uniform sampling; (b) Halton sampling; (c) Classic Latin Hypercube sampling (LHS); (d) Centered Latin Hypercube (CLHS); (e) Maximin Latin Hypercube (MLHS). The images illustrate that the more advanced sampling methods (b–e) generated points that were much more evenly spread out compared to uniform sampling (a), i.e., the image plane was much better covered by a relatively small set of points. Thus, using a low-discrepancy sampling method allowed for the generation of sets of relatively few individuals, which were more representative of their corresponding space compared to using simpler sampling approaches.

### 4.4. Variations of the Fundamental ICA Steps

Although the classic ICA has been proven to perform well for a large number of optimization problems, we analyzed various modified versions of the essential ICA steps in order to find the best ones for our purposes.

**Assimilation** is a form of mutation in which the colonies are modified so that they are more similar to the imperialist countries. Considering that imperialist countries are constantly maintained to be the best in their respective empires, assimilation is equivalent to moving the solutions toward the fittest local or global one(s). Considering the variables $c_i$,
of any colony from the country population, which is assimilated by a destination imperialist $I_{\text{dest}}$, a new set of variables $c_i'$ is obtained by updating the current values $c_i$ using a parameter $a_{\text{dist}}$, which signifies the maximum distance toward the assimilating imperialist, as shown in Equation (1).

$$c_i' = c_i + a_{\text{dist}} \cdot U(0, 1) \cdot (I_{\text{dest}} - c_i)$$  

(1)

where $U(0, 1)$ is a random uniform number from the $(0, 1)$ interval.

In order to improve our algorithm, we considered the following types of assimilation:

- **Local assimilation:** Each colony moves toward its owner empire, equivalent to the solutions moving toward the current local minimum (Figure 4b).
- **Global assimilation:** Most countries move toward the best imperialist country, regardless of ownership. A smaller percentage maintains local assimilation to maintain the relevance of other imperialists. This form of assimilation is equivalent to most solutions moving toward the current global minimum. We find that a percentage of 75% of countries moving toward the best imperialist is suitable for our algorithm, whereas the remaining 25% move toward their owners in a local fashion (Figure 4c).
- **Free will assimilation:** Each country is free to choose which imperialist they move toward. The choice is made through a roulette wheel selection, where for each country, its destination empire is chosen using a distribution determined from the inverse costs of the empires. In this manner, each imperialist has a probability of being chosen, which is inversely proportional to its cost and equivalent to the colonies being more likely to choose better imperialists (Figure 4d).

![Figure 4](image-url)

**Figure 4.** Illustration of the tested assimilation types: (a) The imperialist–colony configuration, where the imperialists are the larger points; the size of the imperialist points indicates their fitness; (b) local assimilation, where each country moves towards its imperialist owner; (c) global assimilation, where most countries move toward the best imperialist; (d) free will assimilation, where countries choose which imperialist to move toward.

Our analysis showed that global assimilation proved to be the most useful approach for our algorithm, primarily due to an improved convergence curve.

**Revolution** is a step where the countries typically change independently from each other. We implemented this step as a mutation of the values of each respective country. Specifically, we tested two types of revolution:

- **Uniform revolution**, where countries change via a uniform mutation operator. Considering that a country $C$ is defined by variables $c_i$, $i \in \{1, \ldots, l_c\}$, where $c_i$ are equivalent to the genes of a chromosome from genetic algorithms and $l_c$ is the variable count (equivalent to the length of the chromosome), a uniform revolution involves applying
a uniform mutation operator to $C$ with probability $p_r$. This involves replacing variable $c_i$ with a randomly generated value from a uniform distribution within its respective domain $[c_{i,\min}, c_{i,\max}]$, where for each $c_i$ occurs with probability $p_{rv}$ (the probability of per-variable revolution). Let $\alpha \sim U(0,1)$, $\beta_i \sim U(0,1)$ and $\gamma_i \sim U(c_{i,\min}, c_{i,\max})$. Then, the $c_i$ values are updated as shown in Equation (2).

- **Normal revolution**: Countries change via a mutation operator that uses normal distribution. Considering the previously defined country $C$ with variables $c_i$, a normal revolution occurs with probability $p_{rv}$, where each variable $c_i$ is modified via a normally generated value centered in $c_i$ with a standard deviation of 1, scaled by step size $s_r$. Each variable is mutated with probability $p_{rv}$. The resulting value is then truncated within the respective domain $[c_{i,\min}, c_{i,\max}]$ of variable $c_i$. Let $\alpha \sim U(0,1)$, $\beta_i \sim U(0,1)$ and $\gamma_i \sim N(c_i, 1)$. Then, the mutated values $c_i'$ are updated from the original ones, $c_i$, as shown in Equation (3).

\[
c_i' = \begin{cases} 
\gamma_i & \text{if } p_r > \alpha \text{ and } p_{rv} > \beta_i \\
 c_i & \text{otherwise}
\end{cases}
\] (2)

\[
c_i' = \begin{cases} 
 s_r \cdot \gamma_i & \text{if } p_r > \alpha \text{ and } p_{rv} > \beta_i \\
 c_i & \text{otherwise}
\end{cases}
\] (3)

**Competition** is a step where imperialists attempt to gain additional colonies by taking them from other imperialists. The type of competition typically has a strong influence on convergence speed, as it is the main factor in controlling the rate at which imperialists are eliminated. As with the previously mentioned steps, we analyzed several types of competition, as follows:

- **Weakest competition**: At each iteration of the algorithm, an imperialist chosen by roulette-wheel selection conquers the weakest colony of the weakest imperialist. Eventually, the weakest empire is left without colonies and is removed. Statistically, stronger imperialists will accumulate the highest number of colonies. The total cost $t_{cost}$ of an imperialist $I$ with $n_{IC}$ colonies is determined via a combination of the imperialist’s own cost $cost_I$ and a fraction $f_{cost}$ of the average cost of its colonies $cost_{Cr}$, $i \in \{1, \ldots, n_{IC}\}$ (Equation (4)).

- **Strongest competition**: At each iteration, the strongest imperialist conquers the weakest colony of an imperialist chosen by roulette-wheel selection. When choosing a source imperialist, the distribution used for selection is generated so that imperialists with a higher cost have a higher probability of being selected. Therefore, weaker imperialists have a higher chance of yielding their weakest colony to the strongest imperialist.

- **Multiple competition**: This version is a combination of the selection methods from the weakest and strongest competition types. A conquering imperialist is chosen by roulette-wheel selection, where the related distribution favors lower-cost imperialists, and the conquered imperialist is also chosen by roulette-wheel selection but using a distribution that favors weaker imperialists. Therefore, a statistically stronger imperialist takes the weakest colony of a statistically weaker one.

- **Aggressive competition**: This approach is similar to the weakest competition, the difference being that the weakest imperialist is conquered multiple times during the same iteration. Each time, the weakest colony of the weakest imperialist is yielded to a stronger imperialist chosen by roulette-wheel selection. This approach means that weaker imperialists are depleted of their colonies at a faster rate, resulting in an overall significantly faster convergence. The number of colonies $n_{LC}$ lost by the weakest imperialist is decided by a hyperparameter of the algorithm, which we refer to as $comp_{AG}$ (competition aggression). $comp_{AG}$ is defined in the interval $[0,1]$ and its value is used to determine the actual number of lost colonies $n_{LC}$ according to the number of
initial countries $n_{\text{InitC}}$, the number of initial imperialists $n_{\text{InitI}}$, and the current iteration $it$ (Equation (5)).

\[
t\text{cost}_l = c\text{ost}_l + \frac{f\text{cost}}{n_{\text{InitI}}} \sum_{i=1}^{n_{\text{InitC}}} c\text{ost}_{c_i}
\]

\[
n_{\text{LC}} = \left[\ln(\text{comp}_{\text{Ag}} \cdot \frac{n_{\text{InitC}}}{n_{\text{InitI}}} \cdot it + 1)\right]
\]

4.5. Backpropagation Boosting

Additionally, from the evolutionary approach described in previous sections, we employed a backpropagation step that was incorporated into the modified ICA pipeline, which served as a complementary boost for each country. At the beginning of each iteration, we applied an additional partial optimization of each solution via gradient-based backpropagation, where each neural network was trained in varying amounts to get closer to the optimum. This operation served as a supplementary boosting mechanism that slightly steered each solution in the “right” direction, i.e., toward the optimal solution. This approach worked in tandem with the evolutionary pipeline, making the overall algorithm a hybrid of two optimization strategies:

- **The evolutionary approach** contributed mainly to exploring the solution space and ensuring the diversity of the solution population. To this extent, assimilation, revolution, and competition operators were applied to the solutions in order to regroup and transform them accordingly. As described in previous sections, we analyzed multiple versions of the fundamental ICA steps in order to find the ones that best suited our needs in terms of convergence speed and efficiency, considering the complexity and dimensionality of our data.

- **Backpropagation boosting** further “pushes” the solutions toward the optimum. We found that, for our problem, this complementary approach had the most significant contribution to minimizing errors and eventually reaching a solution that was as close as possible to the optimum. As such, our approach leveraged the advantages of both an evolutionary algorithm and traditional gradient-based neural network training. Furthermore, the addition of backpropagation boosting added two hyperparameters to our algorithm: a parameter that controlled the number of epochs (backpropagation acceleration) and the learning rate used for backpropagation training.

4.6. Algorithm Parameters and Parameter Control Strategies

Considering the previously described steps and the versions of the said steps ultimately chosen for our approach (as seen in Figure 2), we defined several hyperparameters that impact the convergence of our hybrid evolutionary algorithm. For efficiency of discussion, we henceforth refer to these hyperparameters as “algorithm parameters” or simply “parameters” (Table 1). From the perspective of parameter control, we split the parameters into two categories: static parameters, which were globally set from the start of the algorithm, and variable parameters, whose values were steered during the iterations of the algorithm. The parameters in the second category were applied locally for each solution (they influenced countries at the individual level). These parameters were subjected to a mixed adaptive/self-adaptive parameter control strategy.
Table 1. The parameters of our algorithm, their values, and their role within the algorithm’s iterations. The value “self-adaptive” refers to the fact that the values of the corresponding parameters were optimized by the algorithm itself via the self-adaptive control mechanism.

| Parameter | Role | Default Value |
|-----------|------|---------------|
| \( n_{\text{init}} \) | Initial population count | 100 |
| \( n_{\text{initI}} \) | The initial imperialist count | 10 |
| \( \text{compAg} \) | Aggressiveness of the imperialist during the competition phase | 0.4 |
| \( f_{\text{cost}} \) | The fraction of the total imperialist cost contributed by its colonies | 0.3 |
| \( p_r \) | Probability of revolution occurring for a country | self-adaptive |
| \( p_{rv} \) | Probability of revolution occurring for each individual value of a country | self-adaptive |
| \( s_r \) | Step size used for normal revolution | self-adaptive |
| \( a_{\text{dist}} \) | Maximum distance to the destination imperialist during assimilation | self-adaptive |
| \( bp_a \) | Controls the number of epochs used for backpropagation boosting | self-adaptive |
| \( b_{\text{PLR}} \) | Learning rate used for backpropagation boosting | self-adaptive |

In order to aid in finding optimal values for some of the parameters in Table 1, we used a mixed adaptive/self-adaptive parameter control strategy. When using the term “adaptive”, we refer to the fact that during the iterations of the evolutionary algorithm, certain parameters were modified according to the state of the algorithm in each respective iteration in order to improve on the progress of the algorithm’s convergence. The topic of adaptive parameters in the evolutionary algorithm has been widely discussed in the related literature [56–58]. Consequently, for our approach we employed a few simple adaptive parameter control strategies, as follows:

- During the aggressive competition phase, the number of colonies lost by the weakest imperialist was adjusted according to the current iteration (Equation (5)). As the evolutionary algorithm advanced and progressed toward the optimum solution, the aggressiveness of the imperialists increased to speed up convergence.
- During the normal revolution phase, the actual step size used for normal mutation was adjusted according to the current state of convergence. In order to achieve this, we multiplied the base step size \( s_r \) by the square of the lowest current cost. Consequently, the step size decreased as the algorithm converged and lower-cost countries are found. The justification for this modification is that, as the algorithm converged toward a minimum, the mutation step size should be reduced so that the algorithm searches for other solutions in narrower neighborhoods [58].
- The number of epochs used during the backpropagation boosting phase was influenced by the current iteration. Specifically, the number of epochs was defined as round \( (bp_a \cdot \text{iter}) \), i.e., we multiplied the backpropagation acceleration by the current iteration and rounded the result. This means that the solutions were boosted further as the algorithm converged in order to encourage finding the optimal one.
- The learning rate used during the backpropagation boosting phase was multiplied by the current lowest cost. As the algorithm converged and lower-cost solutions were found, the learning rate also decreased to permit more fine-grained backpropagation.

Aside from the adaptive measures, for the country local parameters we employed a self-adaptive control strategy. This refers to the fact that the parameter values were initialized and subjected to the same transformations as the solutions themselves (with the notable exception of backpropagation boosting). In essence, the evolutionary algorithm optimized its own parameters as it searched for the optimal solution. It is worth noting
that self-adaptation in evolutionary algorithms is a well-known strategy that is frequently used for parameter control [57].

We implemented self-adaptive parameter control by extending the chromosome of each country with additional locations to represent the values of the self-adaptive parameters (Figure 5). Subsequently, we allowed the evolutionary algorithm to find both the optimal parameter values and the optimal solution (i.e., the optimal neural network). This means that, on the one hand, each country had its own set of self-adaptive parameters as opposed to using a global parameter set. On the other hand, as the algorithm converged, better countries with lower costs also contained better values of the self-adaptive parameters.

Figure 5. The extended variables of each country with added self-adaptive parameters.

5. Experimental Results

Our evaluation methodology first involved assessing the usefulness of the various alternative steps described in Section 4.4. To this extent, we tested several variations of some of the most important operators applied to the population during the main ICA loop. Secondly, we tested the contribution of the adaptive parameter control strategy as well as of the backpropagation boosting used in conjunction with the main algorithm. Our data set contained experimental data consisting of 177 instances with 7 input and 2 output values, as described in Section 2. For our test methodology, we aimed to select the strategies, steps, and parameter control mechanisms that produced a regression model that minimized the RMSE on our dataset. Although our main goal was to obtain the lowest possible error value, we also considered the convergence curve when evaluating the various employed strategies.

With regard to population initialization, we tested several initialization methods, as described in Section 4.3. The most commonly used sampling method used a uniform distribution to generate the values for the chromosomes. However, we explored a few other low-discrepancy alternatives. A low-discrepancy sampling method attempts to spread the generated values as evenly as possible throughout their domain. As such, lower-discrepancy sampling spaces out the initial population so that the density is relatively even throughout the problem space. To this extent, we compared uniform initialization with four other low-discrepancy initialization approaches. The resulting convergence curves are shown in Figure 6. Our tests showed that uniform initialization performed the worst, resulting in the highest RMSE, whereas the best initialization approach was Maximin Latin Hypercube sampling (MLHS), with Centered Latin Hypercube sampling (CLHS) coming close in terms of the results. Overall, using MLHS resulted in an improvement of about 15% compared to uniform initialization, which we consider to be a noticeable improvement, albeit not a dramatic one. The implementation overhead induced by one of the initialization strategies is worthy because (a) the initialization was performed only once at the beginning of the evolutionary algorithm, and (b) we consider a 15% improvement in RMSE to be satisfactory.

We used similar reasoning when trying the various alternative steps throughout the ICA loop. Of the three assimilation strategies tested, global assimilation demonstrated the best results and generated the fastest convergence with the steepest curve (Figure 7), whereas local assimilation performed the worst. Consequently, having most of the solutions migrate toward the strongest overall imperialist resulted in the lowest overall error value, whereas the classic approach where the solutions moved toward their owners provided the worst convergence. We observed a 15–20% improvement in RMSE by choosing an appropriate assimilation strategy.
In terms of competition strategies, surprisingly, the four tested approaches did not have substantial differences in terms of error minimization, resulting in an improvement of below 10%. One can see in Figure 8 that the aggressive competition strategy resulted in the fastest decreasing and overall lowest error value. These results were predictable since aggressive competition statistically leads to the fastest elimination of imperialists and therefore the imposed convergence criterion (one empire left) is reached the quickest. Consequently, the chosen competition strategy unintuitively had the least effect in terms of error minimization, though it ultimately proved to be an effective means of controlling the convergence speed and steepness.

Adopting an adaptive parameter control strategy and using backpropagation boosting to the population provided the biggest improvements to our method. Using a combination of adaptive and self-adaptive parameter controls, as described in Section 4.6, made a noticeable improvement to the resulting errors, compared to the non-adaptive approach.
Incorporating the algorithm parameters into the evolutionary optimization loop did not result in a noticeable penalty in terms of convergence speed, as the algorithm seemed to effectively optimize its own parameters. However, incorporating a backpropagation-based boosting led to the most significant improvement to our method. As presented in Section 4.5, for each iteration of the evolutionary algorithm, we steered the solutions toward the optimum via partial backpropagation-based training. We used the Adam optimizer during each backpropagation epoch [59], whereas the number of epochs and learning rate are part of the overall algorithm parameters. As such, the resulting algorithm was a hybrid that incorporated both an evolutionary and a gradient-based backpropagation approach. The convergence curves obtained using various improved and unimproved versions of the algorithm are shown in Figure 9, where it is clear that the hybrid-adaptive approach provided a significant improvement in terms of minimizing errors.

**Figure 8.** Comparison of convergence using various competition strategies: weakest—the highest-cost imperialist loses a colony to one of the other imperialists; strongest—the lowest-cost imperial gains a colony from one of the others; multiple—the conquering and conquered imperialists are both chosen based on their costs; aggressive—the highest-cost imperialist loses several colonies to the others.

**Figure 9.** Comparison between the non-adaptive, adaptive, and hybrid adaptive versions of our algorithm. In the adaptive version, adaptive and self-adaptive parameter control strategies were employed. In the hybrid version, the solutions were subjected to backpropagation boosting.
We also evaluated our algorithm by testing it against several evolutionary metaheuristics from the related literature:

- Football Game Algorithm (FGA) [60];
- Imperialist Competitive Algorithm (the classic version of ICA [3];
- Simple Human Learning Optimization (SHLO) [61];
- Social Learning Optimization (SLO) [62];
- Teaching Learning-Based Optimization (TLBO) [63];
- Viral System (VS) [64];
- Virulence Optimization Algorithm (VOA) [65];
- Volleyball Premier League (VPL) [66].

These algorithms have also been compared in a previous work [67] for the optimization of another industrial chemical process.

Our test methodology was as follows: first, we minimized the RMSE using the entire dataset. Then, we minimized the RMSE on 70% of the data (the training subset) and tested the resulting models on the 30% remaining instances (the test subset). We performed these tests by optimizing both outputs simultaneously via multivariate regression, and each output individually, therefore generating a separate regression model for each output separately. The results obtained from optimizing both outputs are presented in Table 2, whereas the results for the individual outputs are shown in Tables 3 and 4, respectively. In each Table, we list the RMSE and the correlation coefficients for the corresponding outputs presented in Section 2: \( \eta \) (yield in the crosslinked polymer) and \( \alpha \) (swelling degree). The output values were obtained using all instances from the dataset (the columns titled “all”), as well as using the training subset (“train”) and the test subset (“test”).

### Table 2. RMSE and \( r \) metrics for the two outputs (\( \eta, \alpha \)) resulting from applying the tested algorithms for optimizing the neural network architecture and weights using the entire dataset (“all”), the training split (“train”), and subsequently testing on the test split (“test”). ICAHY is our method.

| Method    | RMSE All | \( r \eta \) All | \( r \alpha \) All | RMSE Train | \( r \eta \) Train | \( r \alpha \) Train | RMSE Test | \( r \eta \) Test | \( r \alpha \) Test |
|-----------|----------|------------------|-------------------|------------|-------------------|-------------------|-----------|----------------|------------------|
| FGA       | 0.03223  | 0.26             | 0.036             | 0.03828    | 0.24              | 0.033             | 0.05966   | 0.139           | 0.009            |
| ICA       | 0.02157  | 0.662            | 0.411             | 0.02533    | 0.443             | 0.407             | 0.04087    | 0.296           | 0.226            |
| SHLO      | 0.02417  | 0.476            | 0.25              | 0.02834    | 0.414             | 0.227             | 0.04603    | 0.271           | 0.184            |
| SLO       | 0.02524  | 0.434            | 0.216             | 0.02998    | 0.472             | 0.216             | 0.04672    | 0.261           | 0.217            |
| TLBO      | 0.01964  | 0.841            | 0.481             | 0.02274    | 0.625             | 0.465             | 0.03838    | 0.387           | 0.433            |
| VOA       | 0.03322  | 0.43             | 0.132             | 0.04013    | 0.321             | 0.087             | 0.06513    | 0.164           | 0.104            |
| VPL       | 0.03366  | 0.404            | 0.085             | 0.0402     | 0.299             | 0.056             | 0.06153    | 0.191           | 0.13             |
| VS        | 0.03556  | 0.313            | 0.019             | 0.0408     | 0.304             | 0.023             | 0.06596    | 0.132           | 0.092            |
| ICAHY     | 0.0089   | 0.96             | 0.843             | 0.01061    | 0.934             | 0.732             | 0.01641    | 0.755           | 0.672            |

### Table 3. RMSE and \( r \) metrics for the yield in the crosslinked polymer \( \eta \), resulting from applying the tested algorithms for optimizing the neural network architecture and weights using the entire dataset (“all”), the training split (“train”), and subsequently testing on the test split (“test”). ICAHY is our method.

| Method | RMSE \( \eta \) All | \( r \) \( \eta \) All | RMSE \( \eta \) Train | \( r \) \( \eta \) Train | RMSE \( \eta \) Test | \( r \) \( \eta \) Test |
|--------|---------------------|--------------------------|----------------------|------------------------|----------------------|------------------------|
| FGA    | 0.02096             | 0.472                    | 0.02478              | 0.45                   | 0.03904              | 0.242                  |
| ICA    | 0.01067             | 0.875                    | 0.01243              | 0.802                  | 0.0205               | 0.532                  |
| SHLO   | 0.01572             | 0.715                    | 0.01866              | 0.695                  | 0.0291               | 0.342                  |
Table 3. Cont.

| Method | RMSE η All | r η All | RMSE η Train | r η Train | RMSE η Test | r η Test |
|--------|------------|---------|--------------|-----------|-------------|---------|
| SLO    | 0.01288    | 0.81    | 0.01512      | 0.725     | 0.02441     | 0.356   |
| TLBO   | 0.01497    | 0.729   | 0.01779      | 0.717     | 0.02757     | 0.338   |
| VOA    | 0.02172    | 0.432   | 0.0237       | 0.433     | 0.04675     | 0.265   |
| VPL    | 0.01964    | 0.686   | 0.02429      | 0.397     | 0.03277     | 0.308   |
| VS     | 0.0215     | 0.425   | 0.02625      | 0.347     | 0.03719     | 0.266   |
| ICAHY  | 0.00374    | 0.984   | 0.00474      | 0.963     | 0.0058      | 0.825   |

Table 4. RMSE and r metrics for the swelling degree α, resulting from applying the tested algorithms for optimizing the neural network architecture and weights using the entire dataset ("all"), the training split ("train"), and subsequently testing on the test split ("test"). ICAHY is our method.

| Method | RMSE α All | r α All | RMSE α Train | r α Train | RMSE α Test | r α Test |
|--------|------------|---------|--------------|-----------|-------------|---------|
| FGA    | 0.02947    | 0.196   | 0.03466      | 0.187     | 0.05578     | 0.185   |
| ICA    | 0.02221    | 0.587   | 0.02611      | 0.434     | 0.04205     | 0.343   |
| SHLO   | 0.02579    | 0.437   | 0.03001      | 0.407     | 0.04983     | 0.338   |
| SLO    | 0.02662    | 0.426   | 0.0309       | 0.409     | 0.05174     | 0.309   |
| TLBO   | 0.02627    | 0.419   | 0.03077      | 0.411     | 0.05008     | 0.301   |
| VOA    | 0.03172    | 0.214   | 0.0384       | 0.166     | 0.05614     | 0.103   |
| VPL    | 0.02882    | 0.231   | 0.0339       | 0.329     | 0.05448     | 0.126   |
| VS     | 0.03368    | 0.236   | 0.04058      | 0.106     | 0.06022     | 0.046   |
| ICAHY  | 0.01142    | 0.828   | 0.01304      | 0.802     | 0.02012     | 0.659   |

These results cover all the above-mentioned algorithms, as well as our algorithm, which is labeled ICAHY (ICA Hybrid). The results show that our algorithm outperformed the others in terms of minimizing errors on all variations and splits of the data and the outputs, often by as much as an order of magnitude. Furthermore, our approach allowed for the optimization of the regression models, which provided the strongest correlations between their predictions and the target values from the dataset. We, therefore, surmise that our hybrid-evolutionary approach constitutes a dependable method for finding and optimizing neural networks capable of generating reliable regression models.

Although in the previous articles that addressed the same database the accuracy was satisfactory, from the point of view of the methodology, the present article has clear advantages. Thus, in [5], the neural networks were designed by the trial-and-error method—an expensive procedure that does not guarantee obtaining the best model. In [6], the successive trials method was also used, adding neural network stacks as a type of model. The present evaluation methodology automatically determined an optimal neural network model from both points of view, structure, and parameters. In addition, the method showed flexibility and generality and could be easily adapted and applied to other complex processes in chemical engineering.

The simulation study of the synthesis process of multicomponent hydrogels benefitted from the results obtained through the predictions for parameters difficult to determine experimentally (swelling degree), which means significant savings in time, materials, and energy.

Based on the data presented in Tables 2–4, we created some derived figures that can help the reader to better visualize the results. They mainly show some relative values of the error metrics employed. However, they have a qualitative character rather than a quantitative one, because, e.g., the correlation coefficient r has a nonlinear definition and a
relative \( r \) cannot be defined by a direct ratio between two such values. Therefore, we use several quality indicators that can provide some insight into the results from a qualitative, possibly managerial, point of view. The greater the value of an indicator, the better.

Figure 10 displays, for each algorithm, the improvements using two separate networks to predict \( \eta \) and \( \alpha \) (case 1) rather than using a single neural network with two outputs (case 2). The results are considered for the testing set only. Thus, we use four quality indicators:

- QI1, defined as the ratio between the RMSE obtained in case 2 and the average of the two RMSEs obtained in case 1.
- QI2, defined as the ratio between the correlation coefficient \( r \) obtained in case 1 for \( \eta \) and the \( r \) obtained in case 2.
- QI3, defined as the ratio between the correlation coefficient \( r \) obtained in case 1 for \( \alpha \) and the \( r \) obtained in case 2.
- QI4, defined as the average of QI1, QI2, and QI3, as a global indicator.

![Figure 10. Improvements of two separate networks for each output over a single network with two outputs.](image)

In the case of FGA, QI3 and consequently QI4 are outliers with very high values (about 20 and 7, respectively), therefore their values are cropped in order to provide a better level of detail for the rest of the values. It can be seen that most, but not all, algorithms benefitted from the separation of the models. In the case of the ICAHY, the values are close to 1, and therefore the single network succeeded in providing results of approximately the same quality as the two individual networks.

Figure 11 shows how much worse the results obtained for the testing set are compared with those obtained for the training set. The purpose of this analysis is to estimate the generalization capability of the models, and it is made for the separate models, i.e., one network for the \( \eta \) output and another network for the \( \alpha \) output. If the training results are good and the testing results are bad, this is an indicator of overfitting. The closer the two results are, the better the generalization capability of the model. However, one can realistically expect the testing results to be (slightly) worse than the training results. Consequently, we use another five quality indicators:

- QI5, defined as the ratio between the RMSE for \( \eta \) obtained for the training set and the RMSE for \( \eta \) obtained for the testing set.
- QI6, defined as the ratio between the correlation coefficient \( r \) for \( \eta \) obtained for the testing set and the \( r \) for \( \eta \) obtained for the training set.
• QI7, defined as the ratio between the RMSE for $\alpha$ obtained for the training set and the RMSE for $\alpha$ obtained for the testing set.

• QI8, defined as the ratio between the correlation coefficient $r$ for $\alpha$ obtained for the testing set and the $r$ for $\alpha$ obtained for the training set.

• QI9, defined as the average of QI5, QI6, QI7, and QI8, as a global indicator.

Figure 11. A comparison of the results obtained for the training and testing sets.

It can be seen that the proposed algorithm ICAHY has the highest value for QI9 among all the algorithms, which demonstrates that it can generalize very well.

Finally, in Figure 12, we compare the results of ICAHY obtained for the testing set with those of the other eight algorithms, and we use:

• QI10, defined as the ratio between the RMSE for $\eta$ obtained with ICAHY and the RMSE for $\eta$ obtained with each algorithm.

• QI11, defined as the ratio between the RMSE for $\alpha$ obtained with ICAHY and the RMSE for $\alpha$ obtained with each algorithm.

• QI12, defined as the ratio between the correlation coefficient $r$ for $\eta$ obtained with each algorithm and $r$ for $\eta$ obtained with ICAHY.

• QI13, defined as the ratio between the correlation coefficient $r$ for $\alpha$ obtained with each algorithm and $r$ for $\alpha$ obtained with ICAHY.

• QI14, defined as the average of QI10, QI11, QI12, and QI13, as a global indicator.

We can see that ICAHY clearly shows better performance than the other algorithms for every error metric. From the qualitative point of view defined by our conventions, its average indicator value is about twice as good as the algorithm in second place, i.e., the standard ICA.

This confirms that the proposed algorithm had very good potential for the optimization problem considered in the present paper and likely also for other optimization problems.
6. Conclusions

We developed a hybrid evolutionary algorithm that optimized a fully connected neural network, which successfully minimized the RMSE on a dataset from the field of chemical engineering. The data were collected from the synthesis of polyacrylamide-based multicomponent hydrogels; they consisted of 7 input parameters and 2 outputs and had as a goal the modeling of the yield and swelling degree as functions of the reaction conditions. In our experience, this dataset has proven difficult in terms of generating a regression model that properly characterizes the underlying phenomenon. Existing evolutionary metaheuristics have proven incapable of properly optimizing a neural network architecture, considering the high dimensionality and complexity of the corresponding neural network space. Consequently, we proposed a method that combined the advantages of two optimization approaches: an evolutionary optimization algorithm that was capable of thoroughly searching through the solution space and providing partial convergence and backpropagation-based optimization, which constituted a necessary boost toward minimizing errors further. For our evolutionary approach, we started with a loop inspired by the Imperialist Competitive Algorithm (ICA) and modified several key steps of the algorithm to better suit the requirements of our particular problem. To this end, we searched and tested various types of initializations, assimilations, and competition strategies. At the same time, we employed an adaptive/self-adaptive parameters control strategy so that the parameters of the algorithm were optimized by the algorithm itself, on the one hand, as well as altered during the algorithm’s iterations, on the other hand. The result of this analysis was an ICA-based evolutionary algorithm that used global assimilation, during which most of the colonies moved toward the strongest imperialist nation as opposed to their own imperialist; aggressive competition, where the imperialists repeatedly conquered the weaker countries from the weakest empire, thus depleting it of colonies much faster and contributing to the convergence speed; and Maximin Latin Hypercube initialization, which used a low-discrepancy sampling method to initialize the population so that it was relatively evenly dispersed throughout the solution space. All of the implemented steps contributed in varying degrees to improving the resulting evolutionary algorithm. Furthermore, the adaptive and self-adaptive strategies contributed in their own regard to improving the performance of the method, while at the same time alleviating the need for time-consuming parameter search and tuning. Ultimately, integrating a backpropagation boost proved to be the most significant enhancement in terms of optimizing a reliable neural
network. Over each iteration of the evolutionary algorithm, the neural network population was slightly optimized with partial backpropagation-based training. This resulted in a much-improved convergence and more reliable neural network candidates.

We tested our method on the dataset in multiple scenarios: for instance, we used both the entire dataset for optimization, and a train/test split; we optimized the errors on the two outputs simultaneously as well as on each of them individually. Our results showed that the neural networks optimized using our approach minimized the RMSE further than alternative evolutionary algorithms and demonstrated better correlations among the predictions and the desired values of the outputs. We, therefore, conclude that our algorithm constitutes a suitable method for finding optimal neural network models that are capable of accurately characterizing the data and making reliable predictions.

Naturally, our work has limitations and there are still many ways to improve upon the method, both in terms of the algorithms themselves and with regard to performance. In terms of the actual approach, there are many versions and modifications of the ICA in the related literature that may provide suitable candidates for a reliable evolutionary algorithm. Currently, backpropagation-based optimization is integrated rather bluntly, and we intend to search for more elegant ways to exploit it for faster convergence. There are other optimizers that can be combined with the evolutionary algorithm to form even mode-capable hybrid methods. Although our method is currently limited to fully connected neural networks, in future work we plan to extend the use of evolutionary optimization to other, potentially more useful architectures, such as sparse neural networks and/or convolutional and recurrent networks. An additional direction for improvement is performance. Hybrid algorithms combine the advantages of multiple methods but require most of the computational resources of those methods. As such, the current implementation of our method could be improved in terms of performance, especially considering runtimes. We intend to expand this aspect of our work by searching for ways to reduce computational times and parallelizing as much of the implementation as the algorithms will allow. This may require possibly migrating the implementation to a GPGPU language to exploit the advantages of GPU-based computation. Finally, neural networks may not be the best and easiest to work with as candidates for regression models. Other functions/parametric models may prove more reliable. We, therefore, intend to explore the related state-of-the-art results for better alternatives that may converge easier or offer higher accuracies/fewer errors. Fortunately, the literature in the related field of this paper provides many resources for gathering information and which are reliable sources of inspiration and learning, for the continuous improvement of our methods.

Author Contributions: Conceptualization, F.L. and S.C.; Investigation, M.G. and S.-A.F.; Methodology, M.G. and F.L.; Software, M.G.; Validation, F.L. and S.C.; Writing—original draft, M.G., S.-A.F., F.L. and S.C.; Writing—review and editing, M.G., S.-A.F., F.L. and S.C.; funding acquisition, S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by UEFISCDI Romania, Exploratory Research Project PN-III-P4-ID-PCE-2020-0551, no. 91/2021.

Acknowledgments: This work was supported by Exploratory Research Project PN-III-P4-ID-PCE-2020-0551, no. 91/2021, financed by UEFISCDI Romania.

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

| Notation | Description |
|----------|-------------|
| $c_i$    | The values of a country $i$ (equivalent to the genes of an individual in genetic algorithms) |
| $l_c$    | The number of variables of a country (equivalent to the size of a chromosome in genetic algorithms) |
| $I_{dest}$ | The destination imperial during the assimilation phase. |
Notation | Description
---|---
$\text{cost}_{ci}$ | The cost of country $c_i$ (countries with lower costs have higher “fitness”, in genetic algorithm terms)
$\text{cost}_i$ | The cost of imperialist $i$
$\text{total cost}_i$ | The total cost of imperialist $i$ (a combination of the imperialists’ own cost and the costs of its colonies)
$f_{\text{cost}}$ | The fraction of the total cost of an imperialist represented by the average cost of its colonies
$n_c$ | The number of colonies of imperialist $i$
$n_{\text{LC}}$ | The number of colonies lost during the competition phase
$it$ | Index of an iteration of the ICA loop
$n_{\text{init}}$ | Initial ICA population count
$c_{\text{init}}$ | Initial ICA imperialist count
$\text{comp}_{\text{Ag}}$ | Aggressiveness of the imperialist during the competition phase
$p_r$ | Probability of revolution occurring for a country
$p_{rv}$ | Probability of revolution occurring for each individual value of a country
$s_r$ | Step size used for normal revolution (the ICA equivalent of mutation)
$d_{\text{dist}}$ | Maximum distance to the destination imperialist during assimilation
$b_{\text{pa}}$ | Scaling factor used to determine the number of epochs used for backpropagation boosting
$b_{\text{PLR}}$ | Learning rate used for backpropagation boosting
$\alpha$ | The swelling degree of the semi- and interpenetrated multicomponent crosslinked structures
$\eta$ | The yield in the crosslinked polymer

References

1. Antonio, L.M.; Coello Coello, C.A. Coevolutionary Multiobjective Evolutionary Algorithms: Survey of the State-of-the-Art. *IEEE Trans. Evol. Comput.* 2022, 22, 851–865. [CrossRef]
2. Sebastian, R. An overview of gradient descent optimization algorithms. arXiv 2016, arXiv:1609.04747.
3. Atashpaz-Gargari, E.; Lucas, C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; pp. 4661–4667. [CrossRef]
4. Curteanu, S.; Dumitrescu, A.; Mihaiescu, C.; Simionescu, B. Neural Network Modeling Applied to Polyacrylamide based Hydrogels Synthesised by Single Step Process. *Polym. Prot. Technol. Eng.* 2008, 47, 1061. [CrossRef]
5. Curteanu, S.; Dumitrescu, A.; Mihaiescu, C.; Simionescu, B.C. Stacked Neural Network Modeling Applied to the Synthesis of Polyacrylamide Based Multicomponent Hydrogels. *J. Macromol. Sci. Part A Pure Appl. Chem.* 2008, A46, 368.
6. Leon, F.; Piuleac, C.G.; Curteanu, S. Stacked Neural Network Modeling Applied to the Synthesis of Polyacrylamide-Based Multicomponent Hydrogels. *Macromol. React. Eng.* 2010, 4, 591–598. [CrossRef]
7. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* 2019, 1, 24–35. [CrossRef]
8. Galvan, E.; Mooney, P. Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges. *IEEE Trans. Artif. Intell. Intell.* 2021, 2, 476–493. [CrossRef]
9. Koppejan, R.; Whiteson, S. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evol. Intell.* 2011, 4, 219–241. [CrossRef]
10. Aryan, H.; Khademi, M.; Pedram, M. Prediction of Photovoltaic Panels Output Power by using MLP, RNN and Neuroevolution Models. *Adv. Nat. Appl. Sci.* 2014, 8, 74–81.
11. Jalali, S.M.J.; Ahmadian, S.; Khosravi, A.; Mirmajili, S.; Mahmoudi, M.R.; Nahavandi, S. Neuroevolution-based autonomous robot navigation: A comparative study. *Cogn. Syst. Res.* 2020, 62, 35–43. [CrossRef]
12. Junior, F.F.F.; Yen, G.G. Particle swarm optimization of deep neural networks architectures for image classification. *Swarm Evol. Comput.* 2019, 49, 62–74. [CrossRef]
13. Wang, B.; Sun, Y.; Xue, B.; Zhang, M. A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In Proceedings of the Australasian Joint Conference on Artificial Intelligence, Wellington, New Zealand, 11–14 December 2018; pp. 237–250.
14. Kim, J.K.; Han, Y.S.; Lee, J.S. Particle swarm optimization–deep belief network–based rare class prediction model for highly class imbalance problem. *Concurrency Comput. Pract. Exp.* 2017, 29, e4128. [CrossRef]
15. Qiang, N.; Ge, B.; Dong, Q.; Ge, F.; Liu, T. Neural architecture search for optimizing deep belief network models of fmri data. In Proceedings of the International Workshop on Multiscale Multimodal Medical Imaging, Shenzhen, China, 13 October 2019; pp. 26–34.
16. Sun, Y.; Xue, B.; Zhang, M.; Yen, G.G. A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE Trans. Neural Netw. Learn. Syst.* 2018, 30, 2295–2309. [CrossRef]
17. Sun, Y.; Xue, B.; Zhang, M.; Yen, G.G. An evolutionary study on hyper-parameter optimization for stacked auto-encoders. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.
18. Jalali, S.M.J.; Ahmadian, S.; Khosravi, A.; Shafie-khah, M.; Nahavandi, S.; Catalo, J.P.S. A Novel Evolutionary-Based Deep Convolutional Neural Network Model for Intelligent Load Forecasting. *IEEE Trans. Ind. Inform.* 2021, 17, 8243–8253. [CrossRef]
19. Rojas, M.G.; Olivera, A.C.; Vidal, P.J. Optimising Multilayer Perceptron weights and biases through a Cellular Genetic Algorithm for medical data classification. *Array* 2022, 14, 100173. [CrossRef]
20. Kaveh, M.; Khishe, M.; Mosavi, M.R. Design and implementation of a neighborhood search biogeography-based optimization trainer for classifying sonar dataset using multi-layer perceptron neural network. *Analog. Integr. Circuits Signal Process.* 2018, 100, 405–428. [CrossRef]
21. Yuan, C.; Moayedi, H. The performance of six neural-evolutionary classification techniques combined with multi-layer perception in two-layered cohesive slope stability analysis and failure recognition. *Eng. Comput.* 2019, 36, 1705–1714. [CrossRef]
22. Bahnse, A.C.; Gonzalez, A.M. Evolutionary Algorithms for Selecting the Architecture of a MLP Neural Network: A Credit Scoring Case. In Proceedings of the 11th International Conference on Data Mining Workshops, Vancouver, BC, Canada, 11 December 2011. [CrossRef]
23. Ho, Y-W.; Rawat, T.S.; Yang, Z.-K.; Pratik, S.; Lai, G.-W.; Tu, Y.-L.; Lin, A. Neuroevolution-based efficient field effect transistor compact device models. *IEEE Access* 2021, 9, 159048–159058. [CrossRef]
24. Sekaj, I.; Kenicky, I.; Zubeck, F. Neuro-Evolution of Continuous-Time Dynamic Process Controllers. *MENDEL* 2021, 27, 7–11. [CrossRef]
25. Ororbia, A.; ElSaid, A.; Desell, T. Investigating Recurrent Neural Network Memory Structures using Neuro-Evolution. In Proceedings of the Genetic and Evolutionary Computation Conference 2019 (GECCO’19), Prague, Czech Republic, 13–17 July 2019. [CrossRef]
26. Lyu, Z.; Karns, J.; ElSaid, A.; Desell, T. Improving Neuroevolution Using Island Extinction and Repopulation. arXiv 2020, arXiv:2005.07376v1.
27. Yilmaz, O.; Bas, E.; Egrioglu, E. The Training of Pi-Sigma Artificial Neural Networks with Differential Evolution Algorithm for Forecasting. *Comput. Econ.* 2022, 59, 1699–1711. [CrossRef]
28. Saporetti, C.M.; Goliatt, L.; Pereira, E. Neural network boosted with differential evolution for lithology identification based on well logs information. *Earth Sci. Inform.* 2021, 14, 133–140. [CrossRef]
29. Kaur, M.; Singh, D. Multi-modality medical image fusion technique using multi-objective differential evolution based deep neural networks. *J. Ambient. Intell. Humaniz. Comput.* 2021, 12, 2483–2493. [CrossRef]
30. Fernandes, F.E.; Yen, G.G. Pruning of generative adversarial neural networks for medical imaging diagnostics with evolution strategy. *Inf. Sci.* 2021, 558, 91–102. [CrossRef]
31. Zamani, H.; Nadimi-Shahraki, M.H.; Gandomi, A.H. Starling murmuration optimizer: A novel bio-inspired algorithm for global and engineering optimization. *Comput. Methods Appl. Mech. Eng.* 2022, 392, 114616. [CrossRef]
32. Nadimi-Shahraki, M.H.; Zamani, H.; Mirjalili, S. Enhanced whale optimization algorithm for medical feature selection: A COVID-19 case study. *Comput. Biol. Med.* 2022, 148, 105858. [CrossRef]
33. Nadimi-Shahraki, M.H.; Zamani, H. DMDE: Diversity-maintained multi-trial vector differential evolution algorithm for non-decomposition large-scale global optimization. *Expert Syst. Appl.* 2022, 198, 116895. [CrossRef]
34. Askari, R.; Sebt, M.V.; Amjadian, A. A Multi-product EPQ Model for Defective Production and Inspection with Single Machine, and Operational Constraints: Stochastic Programming Approach. In Proceedings of the International Conference on Logistics and Supply Chain Management LSCM 2020: Logistics and Supply Chain Management, Tehran, Iran, 23–24 December 2020; pp. 161–193. [CrossRef]
35. Gharaei, A.; Amjadian, A.; Amjadian, A.; Shavandi, A.; Hashemi, A.; Taher, M.; Mohamadi, N. An integrated lot-sizing policy for the inventory management of constrained multi-level supply chains: Null-space method. *Int. J. Syst. Sci. Oper. Logist.* 2022, 1–14. [CrossRef]
36. Taleizadeh, A.A.; Safaei, A.Z.; Bhattacharya, A.; Amjadian, A. Online peer-to-peer lending platform and supply chain finance decisions and strategies. *Ann. Oper. Res.* 2022, 315, 397–427. [CrossRef]
37. Gharaei, A.; Amjadian, A.; Shavandi, A. An integrated reliable four-level supply chain with multi-stage products under shortage and stochastic constraints. *Int. J. Syst. Sci. Oper. Logist.* 2021, 1–22. [CrossRef]
38. Amjadian, A.; Gharaei, A. An integrated reliable five-level closed-loop supply chain with multi-stage products under quality control and green policies: Generalised outer approximation with exact penalty. *Int. J. Syst. Sci. Oper. Logist.* 2022, 9, 429–449. [CrossRef]
39. Gharaei, A.; Shekarabi, S.A.H.; Karimi, M. Optimal lot-sizing of an integrated EPQ model with partial backorders and re-workable products: An outer approximation. *Int. J. Syst. Sci. Oper. Logist.* 2021, 1–17. [CrossRef]
40. Hussein, S.; Chandra, R.; Sharma, A. Multi-step-ahead chaotic time series prediction using coevolutionary recurrent neural networks. In Proceedings of the Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 24–29 July 2016. [CrossRef]
41. Chandra, R.; Zhang, M. Cooperative coevolution of Elman recurrent neural networks for chaotic time series prediction. *Neurocomputing* 2012, 86, 116–123. [CrossRef]
42. Si, T.; Bagchi, J.; Miranda, P.B.C. Artificial neural network training using metaheuristics for medical data classification: An experimental study. *Expert Syst. Appl.* 2022, 193, 116423. [CrossRef]
43. Salman, I.; Ucan, O.N.; Bayat, O.; Shaker, K. Impact of metaheuristic iteration on artificial neural network structure in medical data. *Processes* 2018, 6, 57. [CrossRef]
44. Kumar, N.; Kumar, D. An improved grey wolf optimization-based learning of artificial neural network for medical data classification. *J. Inf. Commun. Technol.* **2021**, *20*, 213–248. [CrossRef]

45. Das, S.; Mishra, S.; Senapati, M.R. New approaches in metaheuristic to classify medical data using artificial neural network. *Arab. J. Sci. Eng.* **2019**, *45*, 2459–2471. [CrossRef]

46. Cartwright, H.; Curteanu, S. Neural Networks Applied in Chemistry. II. Neuro-Evolutionary Techniques in Process Modeling and Optimization. *Ind. Eng. Chem. Res.* **2013**, *52*, 12673–12688. [CrossRef]

47. Cerecedo-Cordoba, J.A.; Barbosa, J.J.G.; Terán-Villanueva, J.D.; Frausto-Solis, J.; Martínez Flores, J.A. Use of Neuroevolution to Estimate the Melting Point of Ionic Liquids. *Int. J. Comb. Opt. Probl. Inform.* **2017**, *8*, 2–9.

48. Yin, W. Personalized Hybrid Education Framework Based on Neuroevolution Methodologies. *Comput. Intell. Neurosci.* **2022**, *2022*, 6925686. [CrossRef]

49. Risi, S.; Togelius, J. Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Trans. Comput. Intell. Al Games* **2017**, *9*, 25–41. [CrossRef]

50. Waris, F.; Reynolds, R. Neuro-Evolution Using Game-Driven Cultural Algorithms. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, Cancun, Mexico, 8–12 July 2020.

51. Tharwat, A.; Schenck, W. Population initialization techniques for evolutionary algorithms for single-objective constrained optimization problems: Deterministic vs. stochastic techniques. *Swarm Evol. Comput.* **2022**, *67*, 100952. [CrossRef]

52. Kazimipour, B.; Li, X.; Qin, A.K. A review of population initialization techniques for evolutionary algorithms. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC)*, Beijing, China, 6–11 July 2014; pp. 2585–2592. [CrossRef]

53. Bangyal, W.H.; Nisar, K.; Ag Ibrahim, A.A.B.; Haque, M.R.; Rodrigues, J.J.P.C.; Rawat, D.B. Comparative Analysis of Low Discrepancy Sequence-Based Initialization Approaches Using Population-Based Algorithms for Solving the Global Optimization Problems. *Appl. Sci.* **2021**, *11*, 7591. [CrossRef]

54. Jassova, A.; Lertochoosakul, P.; Nair, R. On variants of the Halton sequence. *Mon. Math.* **2015**, *180*, 743–764. [CrossRef]

55. Gnewuch, M.; Hebbinghaus, N. Discrepancy bounds for a class of negatively dependent random points including Latin hypercube samples. *Ann. Appl. Probab.* **2021**, *31*, 1944–1965. [CrossRef]

56. Aleti, A.; Moser, I. A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms. *ACM Comput. Surv.* **2017**, *49*, 1–35. [CrossRef]

57. Cerecedo-Cordoba, J.A.; Barbosa, J.J.G.; Terán-Villanueva, J.D.; Frausto-Solis, J.; Martínez Flores, J.A. Use of Neuroevolution to Estimate the Melting Point of Ionic Liquids. *Int. J. Comb. Opt. Probl. Inform.* **2017**, *8*, 2–9.

58. Meyernieberg, S.; Beyer, H.G. Self-Adaptation in Evolutionary Algorithms. In *Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence*; Lobo, F.G., Lima, C.F., Michalewicz, Z., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 54. [CrossRef]

59. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.

60. Djunaidi, A.V.; Juwono, C.P. Football game algorithm implementation on the capacitated vehicle routing problems. *Ind. Eng. Chem. Res.* **2013**, *52*, 12673–12688. [CrossRef]

61. Wang, L.; Ni, H.; Yang, R.; Fei, M.; Ye, W.A. Simple Human Learning Optimization Algorithm. In *Computations in Computer and Information Science*; Book Series CCIS; Springer: Berlin/Heidelberg, Germany, 2014; Volume 462, pp. 56–65.

62. Liu, Z.-Z.; Chu, D.H.; Song, C.; Xue, X.; Lu, B.Y. Social learning optimization (SLO) algorithm paradigm and its application in QoS-aware cloud service composition. *Inf. Sci.* **2016**, *326*, 315–333. [CrossRef]

63. Rao, R.V.; Savsani, V.J.; Vakharia, D.P. Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Comput. Aided Des.* **2011**, *43*, 303–315. [CrossRef]

64. Anton, C.; García, J.M.; Muñuzuri, J.; Onieva, L. Viral systems: A new bio-inspired optimisation approach. *Comput. Oper. Res.* **2008**, *35*, 2840–2860. [CrossRef]

65. Jadheryan, M.; Khotanlou, H. Virulence optimization algorithm. *Appl. Soft Comput.* **2016**, *43*, 596–618. [CrossRef]

66. Moghdani, R.; Salimifar, K. Volleyball premier league algorithm. *Appl. Soft Comput.* **2018**, *64*, 161–185. [CrossRef]

67. Anton, C.; Leon, F.; Gavrilescu, M.; Drăgoi, E.-N.; Floria, S.-A.; Curteanu, S.; Lisa, C. Obtaining Bricks Using Silicon-Based Materials: Experiments, Modeling and Optimization with Artificial Intelligence Tools. *Mathematics* **2022**, *10*, 1891. [CrossRef]