

**METAHEURISTIC APPROACH TO ARTIFICIAL
NEURAL NETWORK OPTIMIZATION**

BY

ALHASSAN MOHAMMEDHANY ALDABBAGH

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

INDUSTRIAL & SYSTEMS ENGINEERING

MAY 2023

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Alhassan Mohammedhany Aldabbagh** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN INDUSTRIAL & SYSTEMS ENGINEERING.**



Dr. Syed Mujahid
(Advisor)



Dr. Anas AlGhazi
Department Chairman



Dr. Anas AlGhazi
(Member)



Dr. Suliman Saleh Al-Homidan
Dean of Graduate Studies



Dr. Waqar Khan
(Member)

Date



© Alhassan Mohammedhany Aldabbagh

2023

I dedicate this thesis to my beloved family for their support and encouragement

ACKNOWLEDGMENTS

I am grateful to Almighty Allah (the Powerful and Exalted in Might) for granting me the strength and resilience to accomplish this task. May the peace and blessings of Allah be upon his last prophet Muhammad (Peace and blessings be upon him) and all his companions (May Allah be pleased with them all).

I would like to express my deepest appreciation to my parents and all my family members for their love, support, and encouragement.

I extend special thanks and heartfelt gratitude to my thesis advisor, Dr. Syed Mujahid, for his invaluable assistance and advice. I am grateful for his efforts, encouragement, support, genuine concern, generous allocation of time, patience, and guidance, especially for his meticulous review of the thesis at each stage.

I would also like to convey my gratitude to my thesis committee members, Prof Salih Duffuaa, Dr. Anas AlGhazi, and Dr. Waqar Khan, for their interest, cooperation, and valuable feedback.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	III
TABLE OF CONTENTS.....	IV
LIST OF TABLES.....	VII
LIST OF FIGURES.....	VIII
LIST OF ABBREVIATIONS.....	IX
ABSTRACT.....	XI
ملخص الرسالة	XII
1. INTRODUCTION.....	1
1.1 Conventional Methods	1
1.2 Challenges	4
1.3 Alternative Methods.....	6
2. LITERATURE REVIEW	10
2.1 Artificial Neural Network Optimization	13

2.2	Why VNS for Optimizing ANN Structure	13
2.2.1	Notable Contributions	15
2.2.2	Challenges and Future Directions:	16
2.2.3	Gap in the literature	17
2.3	Summary of literature review	18
3.	METHODOLOGY.....	20
3.1	ANN Structure Optimization	26
3.2	Variable Neighborhood Search	29
4.	DATA	33
4.1	Iris Dataset.....	33
4.2	WBCD Dataset.....	34
4.3	MNIST Dataset	34
4.4	Pre-processing	35
4.4.1	Splitting	36
4.4.2	One hot encoding the y-target.....	37
4.4.3	Scaling	38
5.	RESULTS AND CONCLUSION	40

5.1	Iris:	41
5.2	WBCD:	43
5.3	MNIST.....	44
5.4	Findings	46
5.5	Win/loss/ties	48
5.6	Discussion.....	51
5.7	Threats to Validity	53
5.7.1	Internal Threats	54
5.7.2	External Threats	54
5.7.3	Construct Threats.....	54
5.7.4	Conclusion Threats.....	55
5.8	Future Work	55
5.9	Conclusion	56
APPENDIX.....		58
REFERENCES.....		61
VITAE.....		65

LIST OF TABLES

Table 1 SGD Variants Overview	2
Table 2 Strengths and Weaknesses of Notable Optimizers	6
Table 3 Datasets Description	35
Table 4 Iris Dataset Results	41
Table 5 WBCD Dataset Results.....	43
Table 6 MNIST Dataset Results	45
Table 7 Iris Dataset Win/Loss/Ties.....	50
Table 8 WBCD Dataset Win/Loss/Ties.....	50
Table 9 MNIST Dataset Win/Loss/Ties	50

LIST OF FIGURES

Figure 1 Categorization of a Sample of Metaheuristic Algorithms	8
Figure 2 Percent of Research on MH Algorithms for ANN Optimization	12
Figure 3 Model Version Comparison.	20
Figure 4 Variable Neighborhood Search Flowchart	22
Figure 5 Switch Layer BP - VNS Flowchart	25
Figure 6 Image Classification Flowchart.....	26
Figure 7 VNS Procedure	32
Figure 8 Gradient Descent Flowchart	32
Figure 9 Preprocessing Flowchart	37
Figure 10 Initialization Flowchart	39
Figure 11 Iris Dataset Pareto Chart.....	42
Figure 12 WBCD Dataset Pareto Chart	44
Figure 13 MNIST Dataset Pareto Chart.....	46
Figure 14 Sample Loss vs. Epoch Graph.....	47
Figure 15 Sample Loss vs. Epoch Graph 2.....	48

LIST OF ABBREVIATIONS

ANN	:	Artificial Neural Network
MH	:	Metaheuristic
VNS	:	Variable Neighborhood Search
GD	:	Gradient Descent
ML	:	Machine Learning
NN	:	Neural Network
BGD	:	Batch Gradient Descent
SGD	:	Stochastic Gradient Descent
MGD	:	Minibatch Gradient Descent
SGDM	:	Stochastic Gradient Descent with Momentum
GA	:	Genetic Algorithm
IGA	:	Improved Genetic Algorithm
PSO	:	Particle Swarm Optimization
CFO	:	Central Force Optimization
MBA	:	Modified Bat Algorithm
GWO	:	Grey Wolf Optimizer
HPSO	:	Hierarchy Particle Swarm Optimization

CNN	:	Convolutional Neural Network
SVM	:	Support Vector Machines
FPGA	:	Field Programmable Gate Array
DSP	:	Digital Signal Processor
WBCD	:	Wisconsin Breast Cancer Diagnosis
MNIST	:	Modified National Institute of Standards and Technology
RNN	:	Recurrent Neural Network
MLP	:	Multilayer Perceptron
BP	:	Backpropagation

|

Abstract

Name: Alhassan Mohammedhany Aldabbagh
Title: Metaheuristic Approach to Artificial Neural Network Optimization
Major Field: Industrial and Systems Engineering
Date of Degree: May 2023

The performance of Artificial Neural Networks (ANNs) depends on the tuning quality of the parameters, such as weights, learning rate, hyperparameters, number of layers and number of neurons. Typical tuning approaches include gradient descent based methods, which are applicable for smooth functions and perform at their best for convex problems. However, ANNs are non-convex and they may benefit from non-smooth functions. Thus, finding new innovative techniques to overcome these limitations is a priority for researchers in the field. Metaheuristic (MH) algorithms can be an alternate option to estimate the optimal parameters of ANNs. Since most MH algorithms have a high capability in exploration or exploitation of non-convex problems, developing customized MH algorithms will be a fruitful area for parameter search of ANNs. In this work, our focus is on developing a Variable Neighborhood Search (VNS) approach that overcomes the limitations of conventional methods in the parameter search of ANNs.

Keywords: metaheuristic, optimization, artificial neural network, variable neighborhood search.

MASTER OF SCIENCE DEGREE
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia.

May 2023

ملخص الرسالة

الاسم الكامل:

الحسن محمد هاني كاكل الدباغ

عنوان الرسالة:

نهج الأدلة العليا لتحسين الشبكة العصبية الاصطناعية

التخصص:

هندسة النظم والصناعة

تاريخ التخرج:

مايو 2023 م

يعتمد أداء الشبكات العصبية الاصطناعية (ANN) على جودة ضبط المعلمات، مثل الأوزان ومعدل التعلم والمعاملات الفائقة وعدد الطبقات وعدد الخلايا العصبية. تتضمن مناهج الضبط النموذجية طرقاً قائمة على النسب المتدرج، والتي يمكن تطبيقها على الوظائف السلسة وتؤدي أفضل ما لديها للمشاكل المحدبة. ومع ذلك، فإن الشبكات العصبية الاصطناعية غير محدبة وقد تستفيد من وظائف غير سلسة. وبالتالي، فإن إيجاد تقنيات مبتكرة جديدة للتغلب على هذه القيود هو أولوية للباحثين في هذا المجال. يمكن أن تكون خوارزميات الأدلة العليا خياراً بديلاً لتقدير المعلمات المثلى لشبكات الـ ANN. نظراً لأن معظم خوارزميات الأدلة العليا لديها قدرة عالية في استكشاف أو استغلال المشكلات غير المحدبة، فإن تطوير خوارزميات الأدلة العليا المخصصة سيكون مجاًلاً مثمراً للبحث عن المعلمات لشبكات الـ ANN. في هذا العمل، ينصب تركيزنا على تطوير نهج (VNS) Variable Neighborhood Search الذي يتغلب على قيود الأساليب التقليدية في البحث عن المعلمات لشبكات الـ ANN.

الكلمات المهمة : الأدلة العليا ، عمليات التحسين، الشبكات العصبية الاصطناعية.

درجة ماجستير في العلوم الهندسية

جامعة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

مايو 2023 م

CHAPTER 1

INTRODUCTION

As universal function approximators, ANNs are designed to learn (approximate) any continuous function for inputs, where the inputs are within a specific range. The multi-layered architecture of ANNs enables them to handle complex non-linearly separable functions. Still, in order to perform well, ANN architecture heavily relies on training factors such as weight initialization, hyperparameters, learning rate, number of layers and neurons at each layer, activation function selection, and regularization methods in training.

The tuning of the training factors is an integral part of a successful ANN application. Conventionally, gradient-descent based methods were employed to improve ANN performance metrics such as error, convergence rate, execution time, and robustness (the extent the algorithm relies on its initialization). And by and large, it has been highly successful in terms of real-world application, nevertheless, gradient-descent variants are limited in some capabilities, prompting computational scientists to find alternatives. One such alternative is the MH approach, which can come in many forms including: tabu search, simulated annealing, swarm intelligence, and evolutionary algorithms.

1.1 Conventional Methods

Gradient descent (GD) methods are 1st-order optimization techniques that are commonly used for training Machine Learning (ML) models and neural networks (NN). Many variants of GD exist, one of which is batch GD (BGD). BGD computes the gradient of the entire

training data set to perform a single update of weights or model parameters in one epoch. Therefore, it may take a long time to train on larger data sets. A technique called Stochastic Gradient Descent (SGD) or mini-batch gradient descent, which performs the update based on one or a subset of training samples, was developed to overcome this limitation. Nonetheless, SGD has its own set of challenges such as dealing with the sparsity of data, proper learning rate selection, and minimizing highly non-convex loss functions and avoiding suboptimal local minima. Numerous algorithms were proposed in the literature to tackle the challenges that arise in the usage of SGD. Table 1 [1] describes some of the most popular SGD variants and their differences.

Table 1 SGD Variants Overview

Parameter Update	About
Vanilla SGD: $\theta_{t+1} = \theta_t - \eta g_t$	Compared to batch gradient descent, SGD reduces parameter variance and resulting in a more stable convergence, but learning is slower.
SGD with Momentum: $\begin{aligned} \theta_{t+1} &= \theta_t - v_{t+1} \mid v_{t+1} \\ &= \gamma v_t + \eta g_t \end{aligned}$	Accelerates the learning process of SGD by accumulating the previous gradients and moving in their directions at speed γ .
Nesterov Momentum: $\begin{aligned} \theta_{t+1} &= \theta_t - v_{t+1} \mid v_{t+1} \\ &= \gamma v_t + \eta \tilde{g}_t \end{aligned}$	Corrects standard momentum by evaluating the current gradient after applying the velocity at that time step.
AdaGrad:	AdaGrad is based on adapting the learning rates of all the parameters by dividing the rate by the square root

$\theta_{t+1} = \theta_t - \left(\frac{\eta}{\sqrt{G_t + \delta}}\right) \odot g_t \mid G_t$ $= G_{t-1} + g_t \odot g_t$	of the sum of past and current squared gradients. The accumulation of gradients makes the learning rate shrink to infinitesimally small values for deep networks.
RMSprop: $\theta_{t+1} = \theta_t - \left(\frac{\eta}{\sqrt{\hat{G}_t + \delta}}\right) \odot g_t \mid \hat{G}_t$ $= \beta \hat{G}_{t-1} + (1 - \beta) g_t \odot g_t$	Solves the diminishing learning rate issue of AdaGrad by dividing the rate by an exponentially weighted average of squared gradients.
Adam: $\theta_{t+1} = \theta_t - \left(\frac{\eta \hat{s}_t}{\sqrt{\hat{r}_t + \delta}}\right) \mid \hat{s}_t = \frac{s_t}{1 - \beta_1^t}, \hat{r}_t = \frac{r_t}{1 - \beta_2^t} \mid s_t$ $= \beta_1 s_{t-1} + (1 - \beta_1) g_t r_t$ $= \beta_1 r_{t-1} + (1 - \beta_1) g_t \odot g_t$	It can be viewed as combining momentum and RMSprop to improve the falling learning rate problem in AdaGrad. Adam makes a significant improvement in the form of training speed. For some data sets, it has been demonstrated that there is a convergence problem. Several methods have been suggested to improve the Adam optimizer and address the convergence problem.
AdaMax: $\theta_{t+1} = \theta_t - \left(\frac{\eta \hat{s}}{R_t}\right) \mid R_t = \max(\beta_2 R_{t-1}, g_t)$	A variant of Adam that primarily utilized in settings with sparse parameter updates by using the exponential moving average of gradients and past p-norm of gradients.

Note the following notation: θ : Model Parameters, η : Learning Rate, $\odot$: Element-Wise Products, g_t : Gradient Estimate At Time-Step T, $\hat{g}_t$: Gradient Estimate At Interim Point, γ : Momentum Coefficient, δ : Small Constant, β, β_1, β_2 : Exponential Decay Rates, s : Biased First Moment Estimator For Adam & Adamax, r : Biased Second Moment Estimator For Adam, R_t : Second Moment Estimator For Adamax.

1.2 Challenges

According to Netrapalli [2], some notable challenges in SGD implementation are:

Step size schedules: How to determine the best step size schedule for steps $k=1, \dots, t$? While $1k/(\sqrt{1k})$ and $1/k$ step sizes are optimal at worst in convex optimization, they are suboptimal on a case-by-case basis. Polyak & Juditsky [3] show that in quadratic problems, a constant learning rate results in asymptotically optimal rates. But if a problem is non-quadratic, decaying step sizes such as cosine annealing are vital. Although many papers have been published on this topic [4], [5], and [6], the appropriate step size schedule for general functions is still an open problem.

Minibatching: Minibatch Gradient Descent (MGD) means that the gradient is calculated across the entire batch before updating weights. Parallelization is the principal basis for employing minibatching. Minibatching also results in a smaller variance leading to a better convergence rate. Setting `minibatch_size=1` leads to full SGD, whereas setting `minibatch_size=n` leads to BGD. Finding the optimal value for the minibatch size with the best convergence rate is the main objective. Dekel et al.[7] conducted research in this area but the performance of MGD had an upper bound, which does not yield conclusive answers. Jain et al. [8] studies the effect of MGD for the linear least squares setting without performance bounding. Further work can be done beyond the case of linear regression.

Acceleration: Lan G [9] showed that momentum techniques can improve SGD performance in the case where the stochastic gradient's variance is bounded. However, this is not common for ANNs. Jain et al. [10] has shown that it is possible to use SGD with

Momentum (SGDM) to improve the convergence rate. Further work can be done on non-strongly convex linear least squares regression.

Local Minima: Another obstacle in minimizing highly non-convex loss functions, which are usually the case for ANNs, is escaping suboptimal local minima. According to Dauphin et al.[11], the challenge is escaping areas surrounded by a plateau of the same error called saddle points. Since the gradient is nearly 0 in every direction, it is very difficult for SGD to avoid getting trapped in.

Generalization: According to Wilson et al. [12], adaptive approaches like Adagrad, RMSProp, AdaMax, and Adam are insufficient for generalizing in a way that is similar to SGD. These techniques typically perform well at the beginning of training, but SGD surpasses them during the later phases. Keskar & Socher [13] proposed training initially with Adam and then switching to SGD at a prespecified point in the training process. Their findings demonstrate that the suggested approach achieves outcomes comparable to SGD while preserving Adam's advantageous traits, such as hyperparameter insensitivity and quick initial progress.

Hyperparameter Sensitivity: Although multiple results in language modeling and computer vision show that SGDM outperforms adaptive approaches ([14], [15], [16]), SGDM requires thorough hyperparameter tuning before yielding sufficient results.

Table 2 [29] presents summary of the strengths and weakness of some of the common gradient-based algorithms that were mentioned above.

Table 2 Strengths and Weaknesses of Notable Optimizers

Optimizer	State Memory [bytes]	# of Tunable Parameters	Strengths	Weaknesses
SGD	0	1	Often best generalization (after extensive training)	Prone to saddle points or local minima. Sensitive to initialization and choice of learning rate.
SGD with Momentum	$4n$	2	Accelerates in directions of steady descent. Overcomes weaknesses of simple SGD.	Sensitive to initialization and choice of learning rate and momentum coefficient.
AdaGrad	$\sim 4n$	1	Works well on data with sparse features. Automatically decays learning rate.	Generalizes worse and converges to sharp minima. Gradients may vanish due to aggressive scaling.
RMSprop	$\sim 4n$	3	Works well on data with sparse features. Built in Momentum.	Generalizes worse and converges to sharp minima.
Adam	$\sim 8n$	3	Works well on data with sparse features. Good default settings. Automatically decays learning rate.	Generalizes worse and converges to sharp minima. Requires a lot of memory for the state.

1.3 Alternative Methods

Metaheuristic (MH) algorithms can estimate the optimal ANN parameters (such as weights, learning rate, hyperparameters and the number of layers and neurons).

ANN parameters' optimization is among the most challenging machine learning problems. Initially, ANNs were trained using gradient-based back propagation techniques. However, since ANN training is an NP-hard optimization problem, using MH algorithms to optimize their parameters and structure has been considerably raised.

Gradient-based learning methods manifest some notable limitations:

- For multiextremal error functions, local minima have been shown to be an obstacle during the learning phase. And in the case where multiple local minima exist in the search space, the learning algorithm may get stuck in the first local minimum.
- The training speed of gradient-based learning methods is slower since calculating gradient information with a large number of iterations is computationally expensive.
- The error function must be differentiable. But in some cases, the activation function may be non-differentiable. For instance, if a unit step function was selected, all backward calculations would be useless.

According to Ojha et al. [17], earlier, the only strategies used for optimizing ANNs were gradient-based linear and quadratic approximations. Because these conventional algorithms lack exploration capabilities, they typically locate local minima. They are local search algorithms that exploit the current solution to generate a new one. In contrast to conventional methods, metaheuristics excel in exploitation and exploration and are able to take into account simultaneous adjustments in every ANN parameter.

Figure 1 shows the categorization of a sample of metaheuristic algorithms. VNS is not specifically categorized as an evolutionary, swarm-based, physics-based, or human-based algorithm. Instead, it can be considered a local search metaheuristic. Local search algorithms, including VNS, focus on exploring the solution space by iteratively moving from one solution to another in the vicinity of the current solution, aiming to improve the objective function or find a better solution. VNS operates by systematically changing the neighborhood structure during the search process. It starts with an initial solution and then iteratively explores different neighborhoods around that solution, trying to find better solutions. When it reaches a local optimum in one neighborhood, it switches to another neighborhood structure to continue the search. While VNS can be used in combination with other algorithms (e.g., genetic algorithms or particle swarm optimization) as part of a hybrid approach, it is primarily a local search method, and its core idea is not tied to any of the specific algorithm categories mentioned (evolutionary, swarm-based, physics-based, human-based).

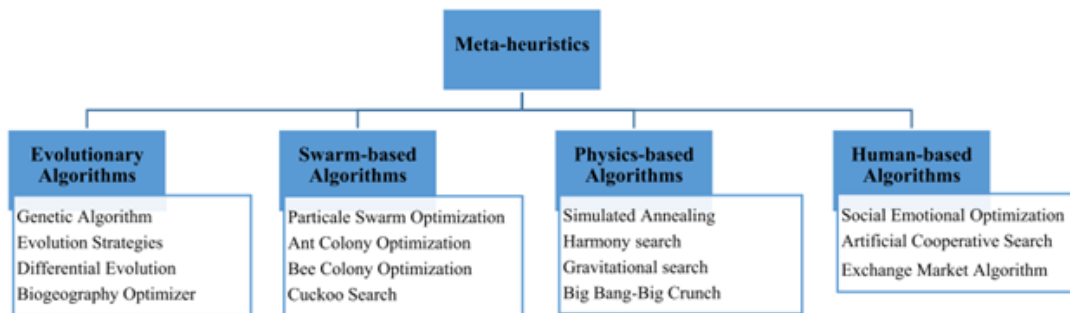


Figure 1 Categorization of a Sample of Metaheuristic Algorithms

The remainder of this study is as follows. Chapter 2 provides an overview of the literature on MH algorithms used to optimize ANN structure, and investigate the application of VNS

for ANN structure optimization, highlighting its benefits, challenges, and notable contributions. Chapter 3 describes the methodology for the proposed VNS approaches. Whereas the background information on the datasets used in this study, steps taken in data preprocessing, and the ANN initialization are described in chapter 4. Followed by chapter 5 where the results are displayed and interpreted in detail. Finally, we conclude the study in chapter 6. |

CHAPTER 2

LITERATURE REVIEW

Leung & Lam [18] improved the Genetic Algorithm (GA) to optimize ANN weights and structure. A major improvement to standard GA was adding multiple mutation stages. Improved Genetic Algorithms (IGA's) performance was tested on six benchmark test functions showing promising results. Then an associative memory was tuned by an ANN trained with the IGA. The proposed architecture (IGA-ANN) was shown to perform better when compared to other architectures using standard GA.

Das & Pattnaik [19] utilized the Particle Swarm Optimization (PSO) algorithm to tune the weights, learning rate, number of layers and neurons simultaneously. Extensive simulations indicated that their methodology led to better performance in all noise conditions compared to other ANN applications. This study is an example for future research implementing PSO or any MH algorithm for tuning multiple ANN parameters simultaneously.

Meissner & Schmuker [20] improved the PSO algorithm by utilizing swarms within swarms to optimize PSO hyperparameters. The Optimized PSO (OPSO) was used for ANN training. Five artificial fitness functions were utilized for the purpose of assessing the proposed algorithm's performance. Additionally, compared to two popular PSO implementations, training speed improved by a factor of four on average.

Zhao & Qian [21] used a modified PSO algorithm to determine the optimal weights and structure of an ANN. The number of nodes were gradually increased until a suitable learning ability was attained. The simulation experiments show that the suggested method can produce superior results compared to methods in recent literature.

Green et al. [22] used the Central Force Optimization (CFO) algorithm to optimize two different ANNs for data classification on a data set from the UCI ML repository (<https://archive-beta.ics.uci.edu/>). CFO-ANN was evaluated against a PSO application using the UCI ML repository data set. Although the study concluded that there is potential for improvement, the CFO algorithm's performance was shown to be the same if not superior to PSO.

Jaddi & Abullah [23] presented a Modified Bat Algorithm (MBA) that accurately selects the network weights or structure. The modifications are intended to enhance the original algorithm's capacity for exploration and exploitation. Different variations of the MBA are utilized to tune the parameters and structure of an ANN during training. MBA's parameters were then adjusted using the Taguchi method. The algorithm was tested on multiple classification and time-series problems. Statistical tests demonstrated that MBA generates superior results compared to similar techniques in the literature.

Hadavandi & Mostafayi [24] presented a Grey Wolf Optimizer (GWO) based ANN for regression application. The GWO algorithm was utilized to optimize the ANN's weights. The model resulted in superior prediction performance compared to a standard Back Propagation (BP) based ANN.

Wang et al. [25] proposed combining Hierarchy PSO (HPSO) with Cellular Automata (CA). The strategy utilizes a hybrid HPSO-CA algorithm to balance between global exploration vs. local exploitation, which then is combined with an ANN to help escape local minima. The proposed model overall outperformed several popular EA-based neural networks using 15 benchmark real-world data sets popular EA-based neural networks using 15 benchmark real-world data sets.

Figure 2 illustrate the distribution of research scope in the ANN optimization using MH algorithms literature.

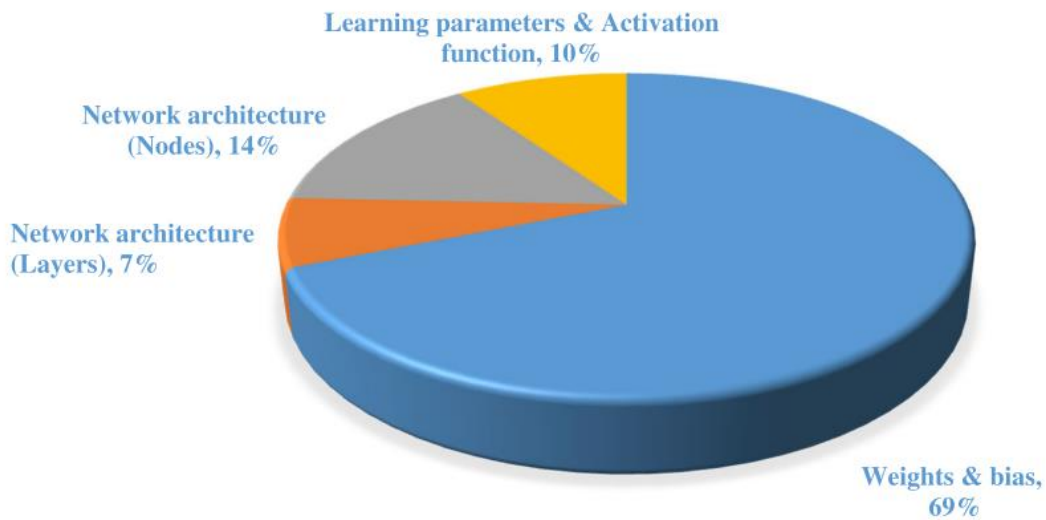


Figure 2 Percent of Research on MH Algorithms for ANN Optimization

2.1 Artificial Neural Network Optimization

In recent years, Variable Neighborhood Search (VNS), a metaheuristic optimization algorithm, has gained attention for its effectiveness in ANN optimization. This literature review provides an overview of the application of VNS for ANN structure optimization, highlighting its benefits, challenges, and notable contributions.

2.2 Why VNS for Optimizing ANN Structure

Variable Neighborhood Search (VNS) can be a suitable choice for optimizing the number of layers and nodes in a neural network due to the following reasons:

Flexibility: VNS allows for flexible exploration of different configurations of layers and nodes in a neural network. It can systematically vary the number of layers and nodes within different neighborhoods, enabling the algorithm to search a wide range of architectural possibilities.

Local Search Capability: VNS incorporates local search algorithms within each neighborhood, which helps in fine-tuning the architecture of the neural network. By iteratively exploring and improving solutions within different neighborhoods, VNS can converge towards locally optimal or even globally optimal configurations.

Escaping Local Optima: Neural network architecture optimization is a highly complex and non-convex optimization problem. VNS's ability to change neighborhoods and explore different regions of the search space helps to escape local optima. By continuously

perturbing the architecture, VNS increases the chances of finding better solutions that may be missed by traditional optimization methods.

Adaptability: VNS can adapt its search strategy based on the characteristics of the optimization problem. The neighborhoods can be designed to capture problem-specific features and constraints. For example, neighborhoods can be defined to control the number of layers, limit the size of the neural network, or enforce certain architectural constraints.

Efficiency: VNS is known for its efficiency in exploring and exploiting different solutions. By using a combination of local search and neighborhood change, VNS can quickly assess the quality of different architectural configurations. It allows for a systematic exploration of the search space, reducing the computational effort required to find good neural network architectures.

Trade-off Exploration: VNS can assist in finding a suitable trade-off between model complexity and performance. By varying the number of layers and nodes, it can help identify architectures that balance model capacity and generalization. VNS can explore a range of configurations and provide insights into the relationship between architecture and performance.

Customizability: VNS can be customized and extended to incorporate problem-specific heuristics or constraints. This flexibility allows researchers and practitioners to adapt VNS to their specific requirements and constraints when optimizing neural network architectures.

Overall, VNS provides a powerful framework for optimizing the number of layers and nodes in a neural network. Its ability to adapt, explore different configurations, and escape

local optima makes it a valuable tool for architectural optimization, leading to improved performance and efficiency of neural network models.

2.2.1 Notable Contributions

In this subsection, we present those works that have significant contributions to the usage of VNS in ANN parameter optimization.

Alba et al. [26] introduced an unsupervised approach for designing Convolutional Neural Network (CNN) architectures. Their method utilized a VNS-based technique to discover CNN architectures and hyperparameter values that enhance classification performance. By applying k-Means clustering to this representation space, they partitioned it based on the relative distances between neighbors. The researchers evaluated their algorithm using the CIFAR-10 image dataset and observed significant improvements in CNN loss (over 15%) and accuracy (5%). Furthermore, the optimized network demonstrated increased predictive power and robustness, affirming the effectiveness of their approach for CNN design optimization.

Uzun et al. [27] demonstrated the efficacy of VNS in improving the performance of a hardware-friendly version of the Support Vector Machine (SVM). The authors aimed to implement the feed-forward phase of SVM on resource-limited hardware devices like Field Programmable Gate Arrays (FPGAs) and Digital Signal Processors (DSPs). They tested their proposal using a machine-vision benchmark dataset for embedded automotive applications and observed significant performance enhancements compared to previously utilized techniques.

Araújo et al. [28] introduced a novel method for extracting accurate and intelligible rules from Trained ANNs using VNS. The VNS method operated independently from the ANN, ensuring that the results of the ANN were not modified. They evaluated their method using real-time Wisconsin Breast Cancer (WBC) data to assess its feasibility. The VNS method was employed to obtain optimal fitness values for input attributes. The proposed method underwent computational evaluation using the investigated data sets, and the results indicated its potential to generate accurate and concise rules. Specifically, the proposed method achieved an accuracy value of 98.97% for the WBC dataset.

2.2.2 Challenges and Future Directions:

Computational Complexity: As the number of layers and nodes in ANNs increases, the computational complexity of VNS optimization grows significantly. Efficient strategies, such as parallelization or intelligent search space exploration, need to be explored to address this challenge.

Scalability: VNS may face scalability issues when optimizing large-scale ANNs due to the vast search space. Researchers should investigate techniques such as problem decomposition or surrogate modeling to handle high-dimensional optimization problems effectively.

Hybridization with Other Techniques: Combining VNS with other optimization techniques, such as evolutionary algorithms or gradient-based methods, may offer improved performance and faster convergence. Hybrid approaches could exploit the

strengths of different algorithms to enhance the efficiency and effectiveness of ANN optimization.

2.2.3 Gap in the literature

The following gaps in the literature on the usage of VNS for ANN can be seen:

Limited Focus: While there have been studies exploring the application of VNS for optimizing ANN structure, the literature still lacks comprehensive and extensive research in this area. Most studies focus on specific types of ANNs or optimization problems, such as MLPs or CNNs for image classification. There is a need for broader investigations that cover a wider range of ANN architectures and problem domains.

Lack of Comparative Studies: Although VNS has shown promising results in optimizing ANN structures, there is a lack of comprehensive comparative studies that assess its performance against other optimization techniques. A comprehensive analysis would involve comparing VNS with other metaheuristics, evolutionary algorithms, or gradient-based methods commonly used for ANN optimization. Such comparative studies would provide a better understanding of the strengths and weaknesses of VNS in relation to other approaches.

Scalability Issues: ANN optimization becomes more challenging as the size and complexity of the network increases. The scalability of VNS for large-scale ANNs, especially with high-dimensional architectures, remains an open question. Research addressing scalability issues and proposing techniques to handle the computational complexity associated with larger networks is lacking.

Hybridization Potential: VNS can potentially benefit from hybrid approaches that combine it with other optimization techniques. However, there is a gap in the literature regarding the exploration of hybridization strategies. Investigating the synergistic effects of combining VNS with other algorithms, such as evolutionary algorithms or gradient-based methods, could lead to enhanced optimization performance and faster convergence.

Real-World Applications: The existing literature on VNS for ANN optimization predominantly focuses on benchmark problems or toy examples. There is a lack of research that applies VNS to real-world, complex problems with diverse data domains. Bridging this gap would require exploring the efficacy of VNS in optimizing ANNs for various applications, such as finance, healthcare, or natural language processing.

Closing this gap in the literature would involve conducting comprehensive comparative studies, addressing scalability challenges, exploring hybridization techniques, and applying VNS to real-world applications. Such advancements would contribute to a deeper understanding of the capabilities and limitations of VNS for ANN optimization and provide valuable insights for practitioners and researchers working in the field.

2.3 Summary of literature review

VNS has emerged as a promising approach for optimizing the structure of ANNs. Its flexibility, ability to escape local optima, and adaptability make it suitable for various ANN optimization problems. Notable studies have demonstrated the efficacy of VNS in optimizing MLPs, (Recurrent Neural Networks) RNNs, and CNNs. However, challenges

related to computational complexity and scalability need to be addressed. Future research should focus on developing hybrid approaches and exploring efficient techniques to tackle these challenges. Overall, VNS offers a valuable framework for ANN parameter optimization.

In the following chapter, the methodology used in this study is presented in detail.

CHAPTER 3

METHODOLOGY

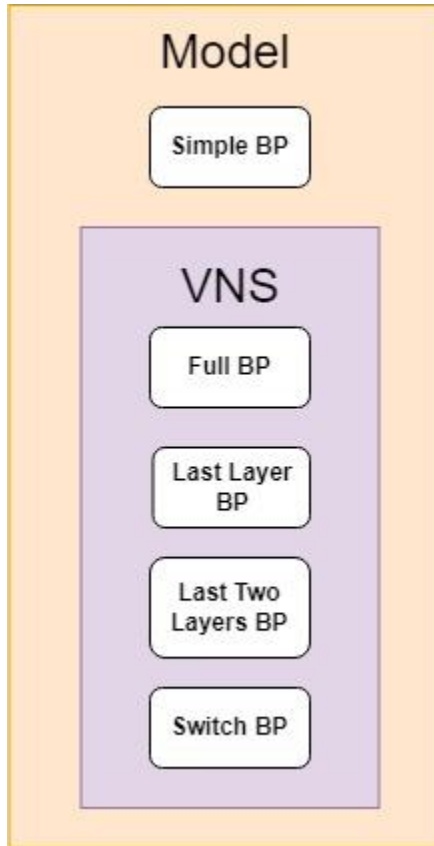


Figure 3 Model Version.

In this study, there are five versions of the VNS approach that are developed to tune the tradeoff between speed and quality. Simple BP is the least sophisticated version where the model simply executed backpropagation on the initial node structure and then attempts to classify the images. Full BP is the most computationally expensive version where the candidate space is explored by running full backpropagation on each candidate in an attempt to find the optimal structure. Last layer BP is the version that attempts to balance the tradeoff in favor of speed over quality in comparison to the Full BP. This is done by executing full backpropagation only once, and then

only running backpropagation for the last hidden layer of each new candidate, and leaving the weights and biases of the expanded/shrunk connections untouched as they were generated randomly according to the normal distribution. Last two layers BP is the version that attempts to balance the tradeoff in favor of quality over speed in comparison to last layer BP. This is done similarly to the last layer BP approach with the exception of backpropagation the error to the last two layers instead of only the last layer. Finally, the

switch BP is the version that attempts to lessen the tradeoff between speed and quality by switching between executing backpropagation on the last layer and last two layers depending on whether an improvement has been found recently. Figure 3 illustrates the five model versions utilized in this study.

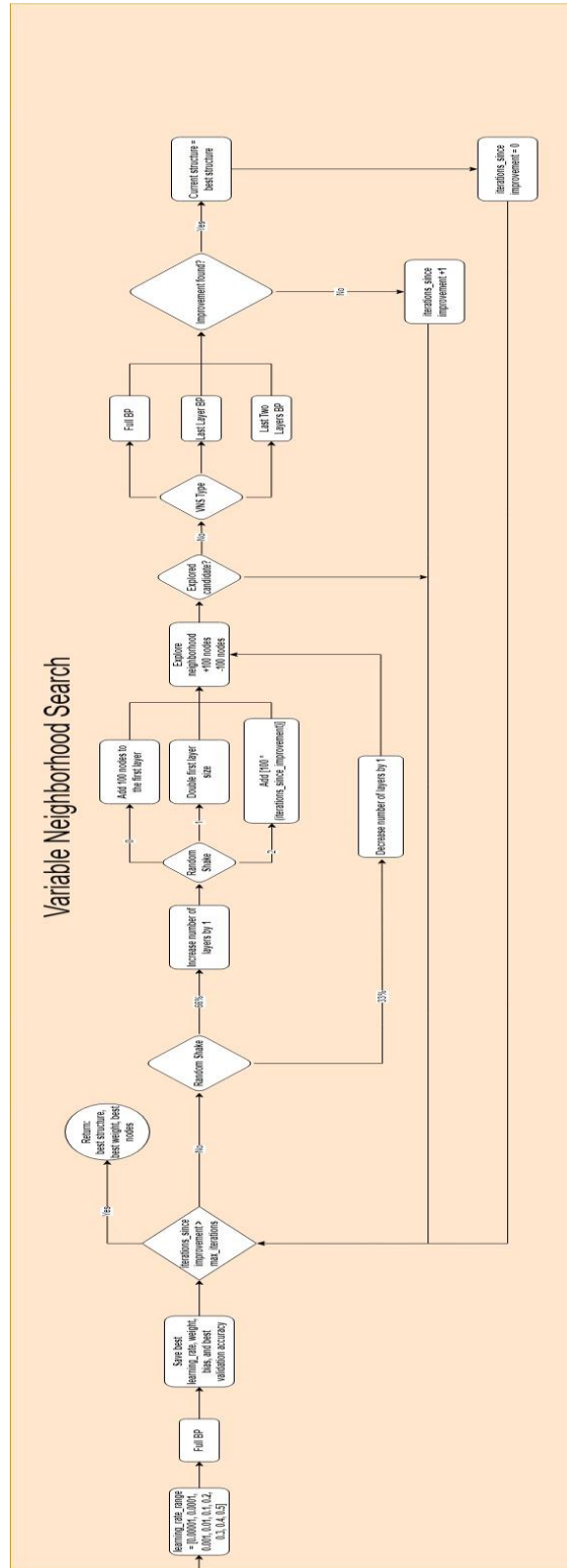


Figure 4 Variable Neighborhood Search Flowchart

During the VNS phase, the process starts by executing full backpropagation on a range of select learning rates. After which the run with the best validation loss is selected and the learning rate, weight, and bias of that run are used to begin the VNS process. A random shake takes place to determine whether to expand the neural network or to shrink it. In two third of the cases, it expands the neural network by duplicating the first hidden layer and either double the newest layer's size, add 100 nodes to the newest layer, or adding $(100 * \text{iterations_since_improvement})$ in an attempt to find candidates that are far away when candidates that are nearby aren't yielding an improvement. However, in one third of the cases, the random shake shrinks the neural network by discarding the furthest hidden layer in an attempt to incentivize finding good solutions that are not too computationally expensive. After which, the neighborhood of the current structure is explored by adding and removing 100 nodes to the first hidden layer resulting in two new candidates. Additionally, there is a redundancy prevention mechanism in place that skips exploiting a candidate that has been already explored. At this point, the one or two new candidates are searched using full backpropagation, last layer backpropagation, or last two layers backpropagation depending on the approach. Once the backpropagation process has been completed, the `iterations_since_improvement` variable is increased by 1 in case no improvement has been found. Otherwise, if the validation loss for any of the new candidates yields a marginal improvement of 1% of the best candidate the `iterations_since_improvement` variable is reset to 0. This process continues until the termination criteria (`iterations_since_improvement > max_iteration`) is met. And so, theoretically, if the VNS keeps finding candidates with marginal improvement, the run could run forever. However, this has never been the case due to the natural limit of superior

candidates in the search space. Figure 4 is a flowchart diagram illustrating the key points of the VNS algorithm.

In the fifth and final approach, switch layer BP, the VNS process is exactly the same with the exception of a mechanism that checks if the `iterations_since_improvement` variable has reached a certain value `exploit_threshold`. This parameter is put in place in an attempt to “squeeze” the candidate for a potential improvement in the case where none was found for a number of iterations. Figure 5 is a flowchart diagram highlighting the difference between the original algorithms and the switch layer BP-VNS algorithm.

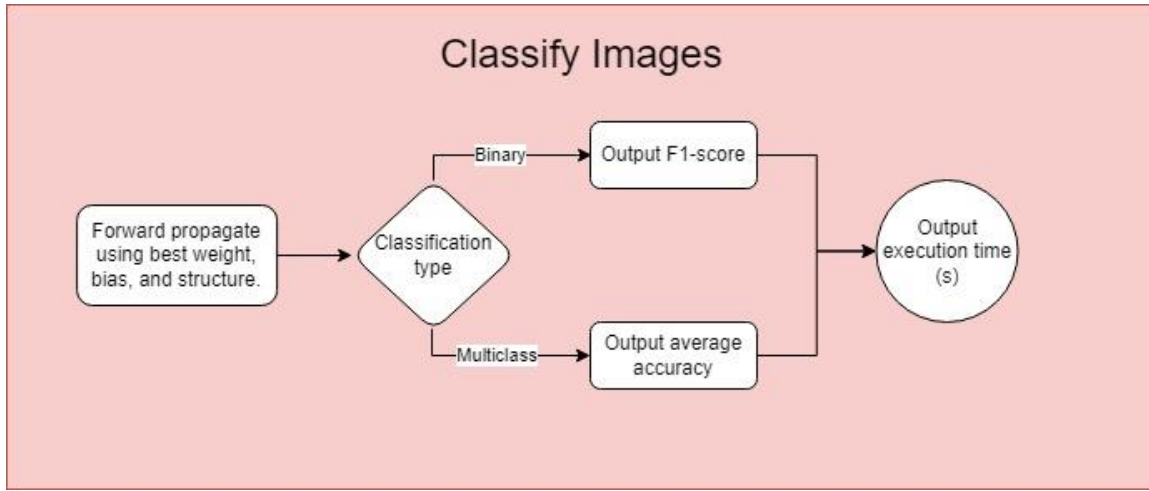


Figure 6 Image Classification Flowchart

Finally, using the best weight, bias, and structure, the model classifies the unseen data. The prediction accuracy is measured using f1-score for binary classification, and average accuracy for multiclass classification. Additionally, the execution time is also utilized to determine which of the proposed versions of the VNS approach is the best performing. Figure 6 is a flowchart diagram of the image classification part of the process.

3.1 ANN Structure Optimization

The process begins by initializing the neural network architecture, specifying the input nodes, hidden layers, and output nodes while randomly assigning weight and bias

values. During forward propagation, input data is fed through the network, applying the sigmoid activation function to neuron weighted sums and calculating their outputs. Subsequently, the loss is calculated by comparing predicted outputs with actual target values. Backpropagation follows, involving gradient computation using the chain rule for loss and network parameters. The ADAM optimizer's update equations are then employed to adjust weights and biases, accounting for moving averages, squared moving averages, and bias-corrected estimates. This iterative process continues, refining the model's performance over a specified number of epochs or until convergence is achieved.

The following is the procedure of the ANN structure is optimization:

- 1) Initialize the network architecture:
 - a) Specify the number of input nodes, hidden layers, and output nodes.
 - b) Randomly initialize the weight and bias matrices for each layer.
- 2) Forward Propagation:
 - a) For each training example in the minibatch:
 - i) Pass the input through the network.
 - ii) Apply the sigmoid activation function to each neuron's weighted sum.
 - iii) Calculate the output of each neuron in the network.
- 3) Calculate the Loss:
 - a) Compute the multiclass cross-entropy loss between the predicted output and the actual target values for each training example in the minibatch.
- 4) Backpropagation:
 - a) Calculate the gradient of the loss with respect to the network's weights and biases using the chain rule.
 - b) Update the weights and biases of each layer using the ADAM optimizer's update equations.
 - i) Update the moving averages and squared moving averages of the gradients.
 - ii) Compute the bias-corrected estimates of the moving averages.
 - iii) Update the weights and biases using the bias-corrected moving averages and squared moving averages.
- 5) Repeat steps 2-4 for a specified number of epochs or until convergence:
 - a) Iterate through the training data in minibatches.
 - b) Update the network's weights and biases based on the gradients and the optimizer's update equations.
- 6) Early Stopping:
 - a) Monitor the validation loss after each epoch.
 - b) If the validation loss does not improve for a specified number of epochs (patience), stop the training process to prevent overfitting.
- 7) Best Model Selection:
 - a) Keep track of the weights and biases that yield the lowest validation loss during the training process.
 - b) These weights and biases represent the best model found so far.
- 8) Predict Function:
 - a) Use the best weight and bias matrices obtained from the training process.
 - b) Forward propagate a new input through the network using the sigmoid activation function.
 - c) Output the predicted class using 12,000 images in the validation data set.
- 9) Implement VNS to find a new candidate ANN structure.
- 10) Repeat steps 1-9 until the termination condition (max iterations) is reached.
- 11) Use the best ANN structure to output the predicted class using 10,000 images in the testing data set

3.2 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a metaheuristic optimization algorithm used to solve combinatorial optimization problems. It is based on the idea of systematically exploring different neighborhoods of candidate solutions to find better or optimal solutions.

The core concept of Variable Neighborhood Search is to define a set of neighborhoods around a given solution and perform local search within each neighborhood. A neighborhood represents a set of potential solutions that can be obtained by applying a specific transformation or perturbation to the current solution.

The algorithm typically follows these main steps:

1. Initialization: Generate an initial solution or start with a feasible solution.
2. Local Search: Apply a local search algorithm within a particular neighborhood to improve the current solution. This step aims to optimize the solution within a limited region of the search space.
3. Neighborhood Change: Perturb the current solution by changing the neighborhood to explore different regions of the search space. This perturbation can be random or based on some predefined rules.
4. Acceptance Criterion: Determine whether the new solution obtained from the neighborhood change is accepted as the current solution. This decision can be based on criteria such as improvement in objective function value or a probabilistic acceptance mechanism.
5. Termination Criteria: Define stopping conditions, such as a maximum number of iterations, reaching a predefined target solution, or exceeding a time limit.

The process of local search within different neighborhoods and changing neighborhoods is repeated iteratively to explore different regions of the search space and improve the quality of solutions. By exploring diverse neighborhoods, the algorithm can potentially escape local optima and find better global solutions.

Variable Neighborhood Search is a flexible and adaptable algorithm that can be customized to fit specific problem structures and characteristics. It has been successfully applied to a wide range of combinatorial optimization problems, including vehicle routing, scheduling, bin packing, and graph coloring, among others.

Overall, Variable Neighborhood Search is a powerful metaheuristic that combines local search within different neighborhoods with systematic exploration of the solution space, enabling efficient and effective optimization for complex combinatorial problems.

The following is a detailed description of how VNS operates to optimize the ANN structure:

- 1) Initialize the ANN architecture:
 - a) Set an initial configuration with a random number of layers and nodes.
 - b) Set the current solution as the initial configuration.
- 2) Define the neighborhoods:
 - a) Define a set of neighborhoods representing possible changes to the ANN architecture.
 - b) Each neighborhood can represent a specific transformation, such as adding/removing layers or increasing/decreasing nodes in a layer.
- 3) Set the VNS parameters:
 - a) Set the maximum number of iterations.
 - b) Set the maximum number of iterations without improvement.
 - c) Set the current iteration counter to 0.
 - d) Set the iteration counter without improvement to 0.
 - e) Set the best solution as the initial configuration.
- 4) Perform VNS iterations:
 - a) Repeat until the termination condition is met:
 - i) Apply local search within the current neighborhood:
 - (1) Apply a local search algorithm (e.g., gradient descent or genetic algorithm) to optimize the current ANN architecture.
 - ii) For each neighborhood:
 - (1) For each transformation in the neighborhood:
 - (a) Perturb the current solution by applying the transformation.
 - (b) Apply local search within the perturbed neighborhood to optimize the modified ANN architecture.
 - (c) Evaluate the quality of the solution obtained from the perturbed neighborhood.
 - (d) If the new solution is better:
 - (i) Update the current solution to the new solution.
 - (ii) Set the iteration counter without improvement to 0.
 - (iii) If the new solution is better than the best solution:
 1. Update the best solution to the new solution.
 - (2) End transformation loop.
 - iii) Change neighborhood:
 - (1) Randomly select a new neighborhood to explore.
 - iv) Increment the iteration counter.
 - v) If no improvement is made in the current iteration:
 - (1) Increment the iteration counter without improvement.
 - vi) If the iteration counter without improvement reaches the maximum number of iterations without improvement:
 - (1) Terminate the VNS loop.
 - 5) Return the best solution found.

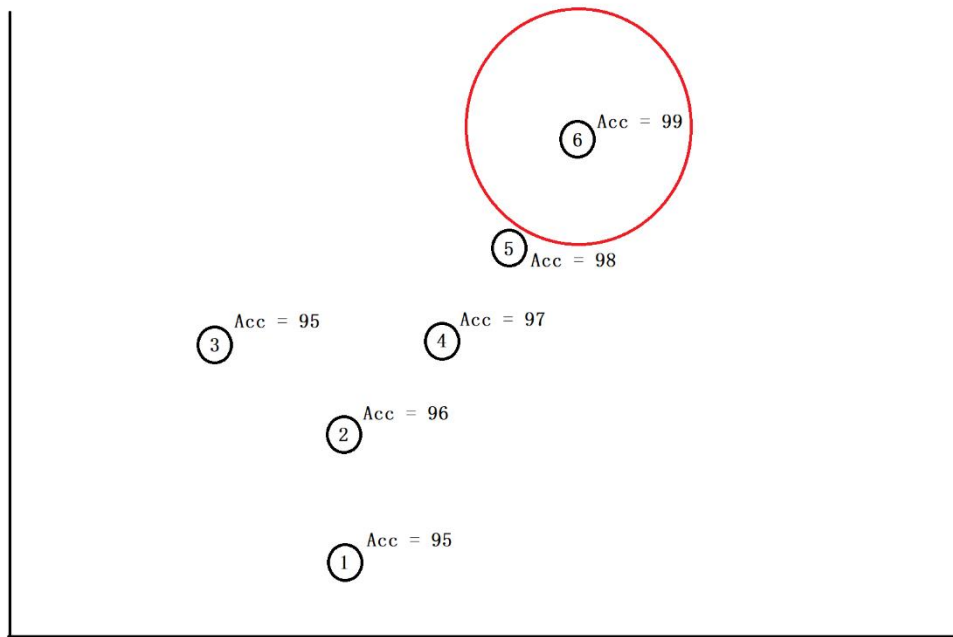


Figure 7 VNS Procedure

Figure 7 represents how a series of candidates being explored takes place. At point 3, the accuracy decreases and so the current solution returns to point 2, which is the best solution at that moment. At point 6, the red circle represents the candidate space or “neighborhood”.

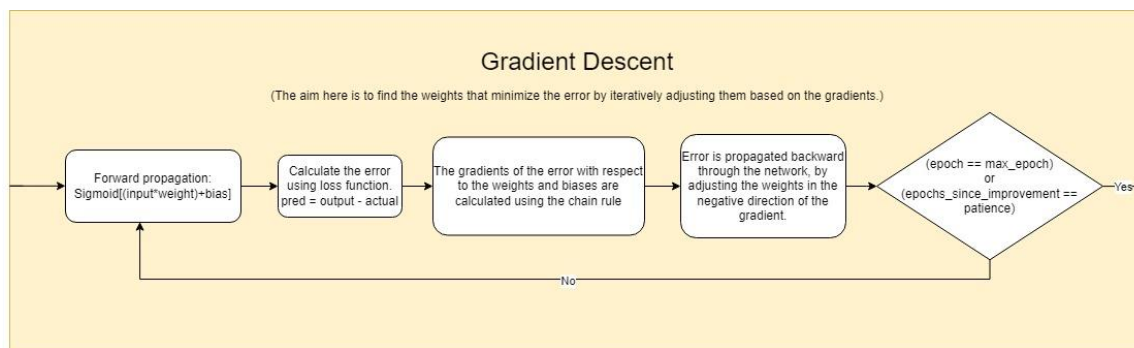


Figure 8 Gradient Descent Flowchart

Figure 8 is a flowchart diagram illustrating the gradient descent segment, where the aim is to find the weights in the neural network that minimize the error in the loss function.

In the next chapter, the datasets used to test the proposed approach are discussed in detail.

CHAPTER 4

DATA

Selecting diverse datasets is crucial when evaluating the performance of an algorithm. While the primary objective is to comprehensively assess the algorithm's capabilities, it's equally vital to ensure that the chosen datasets exhibit significant variations in their characteristics. Diverse datasets challenge the algorithm's adaptability and generalizability, allowing us to gauge its robustness across different scenarios and domains. By encompassing a range of data types, distributions, and complexities, these datasets simulate real-world challenges and help uncover potential weaknesses or biases that might not surface when evaluated solely a single dataset.

4.1 Iris Dataset

The Iris dataset is a well-known benchmark dataset widely used in machine learning and pattern recognition. It was introduced by Ronald Fisher in 1936 and consists of measurements from 150 iris flowers of three different species: setosa, versicolor, and virginica. Each flower sample is described by four features: sepal length, sepal width, petal length, and petal width. The goal of the dataset is to classify the iris flowers into their respective species based on these feature measurements. The Iris dataset is often used as a beginner-level classification problem due to its small size and clear class separability.

4.2 WBCD Dataset

The WBCD (Wisconsin Breast Cancer Diagnostic) dataset is a widely used dataset for breast cancer classification. It was originally obtained from fine needle aspirate (FNA) samples of breast masses. The dataset contains measurements from 569 breast mass samples, with each sample described by 30 features derived from digitized images of the FNA. The features include various characteristics of the cell nuclei, such as radius, texture, smoothness, compactness, concavity, and concave points. The samples are labeled as either benign (non-cancerous) or malignant (cancerous). The objective of the dataset is to accurately classify breast masses as benign or malignant based on the provided features. The WBCD dataset serves as a challenging binary classification problem in medical diagnostics.

4.3 MNIST Dataset

The MNIST dataset, which stands for "Modified National Institute of Standards and Technology," is a widely used benchmark dataset in the field of machine learning and computer vision. It was created by modifying a subset of the original NIST dataset, which consists of handwritten digits and their corresponding labels.

The dataset contains 70,000 handwritten digit images: 60,000 train images and 10,000 test images, with each image being a 28x28-pixel grayscale image. The images represent handwritten digits ranging from 0 to 9. Each image is associated with a label indicating the correct digit it represents.

The goal of the MNIST dataset is to provide a standardized and relatively simple dataset for developing and evaluating machine learning algorithms, particularly for tasks such as image classification and digit recognition.

Table 3 presents a summary of the above three data sets.

Table 3 Datasets Description

Dataset Name	Binary Classification/Multiclass	Type	Target Feature	Number of Observations	Number of Features
Iris	Multiclass (3)	Numerical	Species	150	4
Wisconsin Breast Cancer	Binary Classification	Numerical	Diagnosis	569	30
MNIST	Multiclass (10)	Numerical	Digit Labels	70,000	784

4.4 Pre-processing

Preprocessing in this study involves a series of data transformation and cleaning techniques applied to raw data, aiming to enhance its quality, relevance, and suitability for analysis. This phase encompasses tasks like splitting the datasets to prevent overfitting and data leakage, normalization to scale data to a common range, and encoding categorical variables into numerical formats suitable for our model.

4.4.1 Splitting

Splitting the training set into a train set and a validation set is necessary for several important reasons:

Model Performance Evaluation: The validation set provides a means to assess the performance of the trained model. By evaluating the model on the validation set, we can estimate how well it generalizes to unseen data. This evaluation is crucial for understanding the model's effectiveness, identifying potential issues (such as overfitting or underfitting), and making informed decisions regarding model improvements.

Hyperparameter Tuning: During the model development process, various hyperparameters (e.g., learning rate, regularization strength) need to be selected. The validation set is used to compare and tune these hyperparameters. By training the model on the train set and evaluating its performance on the validation set for different hyperparameter settings, we can identify the combination that yields the best performance.

Preventing Overfitting: Overfitting occurs when the model becomes too complex and starts to memorize the training data instead of learning general patterns. The validation set helps in detecting overfitting. If the model performs well on the training set but poorly on the validation set, it indicates overfitting. This insight prompts adjustments to the model's complexity, regularization, or other aspects to achieve better generalization.

Avoiding Data Leakage: Data leakage refers to the situation where information from the test set inadvertently leaks into the training process, leading to overly optimistic performance estimates. By having a separate validation set, we can ensure that the model

is not exposed to the test set during training, preserving the integrity of the evaluation process.

Splitting the training set into train and validation sets allows for model evaluation, hyperparameter tuning, detection of overfitting, and prevention of data leakage. It is a fundamental step in the model development pipeline that ensures reliable and robust performance assessment of machine learning models.

Figure 9 illustrates the steps taken in the preprocessing phase of the model.

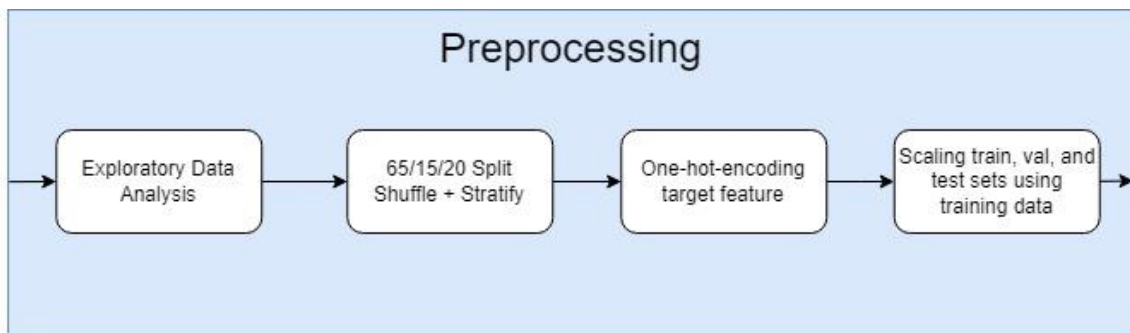


Figure 9 Preprocessing Flowchart

4.4.2 One hot encoding the y-target

One-hot encoding is implemented on the target feature (digit) for the following practical reasons:

Numerical representation: Machine learning algorithms typically work with numerical data. One-hot encoding converts categorical variables into a binary vector format, where each class is represented by a binary value (0 or 1) in a separate feature. This enables the algorithm to process and analyze the data effectively.

Avoiding ordinality assumptions: By using one-hot encoding, we prevent the algorithm from assuming any ordinal relationship between the classes. Without one-hot encoding, assigning arbitrary numerical labels to categorical classes may introduce unintended ordering or hierarchy, which can mislead the algorithm's understanding of the data.

Maintaining class independence: One-hot encoding ensures that each class is treated independently. Each class has its own feature, and the absence or presence of a class is clearly represented by the binary values in that feature. This allows the algorithm to learn distinct patterns and make decisions based on the presence or absence of specific classes.

Enhanced model performance: One-hot encoding helps improve the performance of machine learning models. It allows the algorithm to capture and interpret the categorical information properly, as different classes are represented by distinct binary features. This enables the model to learn class-specific patterns, relationships, and boundaries, leading to more accurate predictions.

4.4.3 Scaling

By using the minimum and maximum values obtained from the training set to scale the validation and test sets, data leakage is avoided. Data leakage refers to the situation where information from the validation or test sets influences the training process, leading to overly optimistic performance estimates. Scaling the validation and test sets independently using their own minimum and maximum values can introduce data leakage because the model would have access to information it should not have during training.

To prevent data leakage, it is crucial to apply the scaling transformation based on the minimum and maximum values obtained solely from the training set. This ensures that the validation and test sets are treated as unseen data, and the model's predictions on these sets reflect its generalization performance to new, unseen samples.

Figure 10 illustrates the initialization phase of the model.

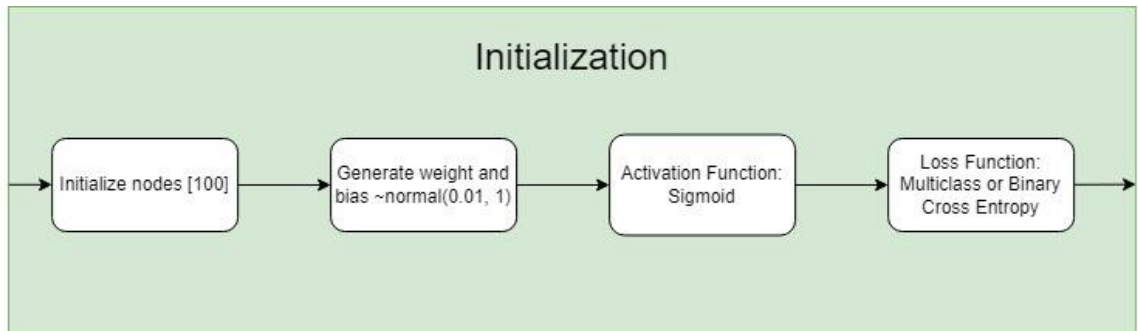


Figure 10 Initialization Flowchart

In the next chapter, we will conclude this study by showcasing the results and discussing their implications.

CHAPTER 5

RESULTS AND CONCLUSION

After selecting and preprocessing the datasets, we initialize our models and perform the five VNS variations for ANN structure optimization and determine the superior variation using the Pareto front and win/loss/ties.

The below list is a short overview of the five VNS variations used in this study:

- Simple BP: executes backpropagation on the initial node structure and then attempts to classify the images.
- Full BP: explores the candidate space by running full backpropagation on each candidate in attempt to find the optimal structure.
- Last layer BP: balances the quality/speed tradeoff in favor of speed by executing full backpropagation only once, and then only running backpropagation for the last hidden layer of each new candidate, and leaving the weights and biases of the expanded/shrunk connections untouched as they were generated randomly according to the normal distribution.
- Last two layers BP: balances the quality/speed tradeoff slightly more in favor of quality than last layer BP. This is done by executing backpropagation on the last two layers instead of only the last layer.
- Switch BP: balances the quality/speed tradeoff by switching between executing backpropagation on the last layer and last two layers depending on whether an improvement has been found recently.

5.1 Iris:

The ANN is initialized using the below parameters for the Iris dataset:

- Activation function: Sigmoid
- Loss function: Multiclass cross entropy
- initial_nodes = [100]
- minibatch_size = 32
- epoch = 1000
- partial_PB_epoch = 250
- patience = 150
- max_iterations = 10
- enable_last_two_layers = int(max_iterations - 3)

In Table 4, we can see that as expected the simple BP is the fastest whereas Full VNS-BP is achieved the highest accuracy. It also appears that the initial structure is dominating the best solutions.

Table 4 Iris Dataset Results

Iris (150x4) (average of 5 runs)	Execution Time (s)	Average Accuracy (%)	Best Structures
1. Simple BP	8.7	98%	[784, 100, 10]
2. Full VNS-BP	63.5	99.3%	[784, 100, 10] [784, 250, 100, 10]
3. Last layer VNS-BP	9.0	98.7%	[784, 100, 10]

4. Last two layer VNS-BP	56.4	98.7%	[784, 50, 100, 10] [784, 100, 10]
5. Switch Layers VNS-BP	10.1	98.7%	[784, 100, 10]

With time in the x-axis and accuracy in the y-axis, it can be said that results improve in the upward and left directions.

In Figure 11, it can be said that models 1 and 2 are the best performing, with model 3 yielding viable results. However, models 4 and 5 have equal accuracy with model 3 but at no improvement in speed.

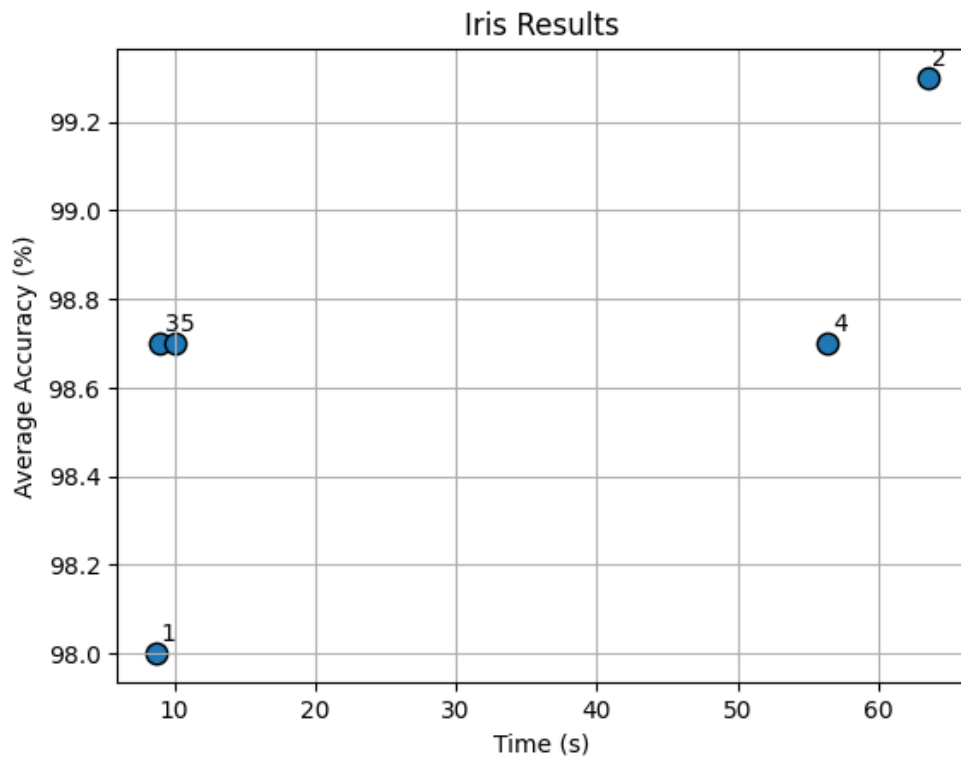


Figure 11 Iris Dataset Pareto Chart

5.2 WBCD:

The ANN is initialized using the below parameters for the Iris dataset:

- Activation function: Sigmoid
- Loss function: Binary cross entropy
- initial_nodes = [100]
- minibatch_size = 32
- epoch = 150
- partial_PB_epoch = 100
- patience = 50
- max_iterations = 5
- enable_last_two_layers = int(max_iterations - 2)

In Table 5, again, we see that Simple BP is outperforming the other models in speed while Full VNS-BP dominates in accuracy. However, the best structure is exhibiting increased variety.

Table 5 WBCD Dataset Results

WBCD (569x30) (average of 3 runs)	Execution Time (s)	F1-Score (%)	Best Structures
1. Simple BP	9.7	90.6%	[784, 100, 10]
2. Full VNS-BP	1719.4	94.7%	[784, 200, 300, 100, 10] [784, 300, 100, 10] [784, 350, 100, 10]
3. Last layer VNS-BP	26.1	91.2%	[784, 100, 10]
4. Last two layer VNS-BP	969.0	92.7%	[784, 50, 100, 10] [784, 100, 10]

5. Switch layers VNS-BP	187.9	94.1%	[784, 300, 100, 10] [784, 100, 10]
-------------------------	-------	-------	---------------------------------------

In Figure 12, similar to the Iris results, we can see that models 1 and 2 dominate in one dimension. However, models 3 and 5 yield viable results while only model 4 is dominated in both dimensions by model 5.

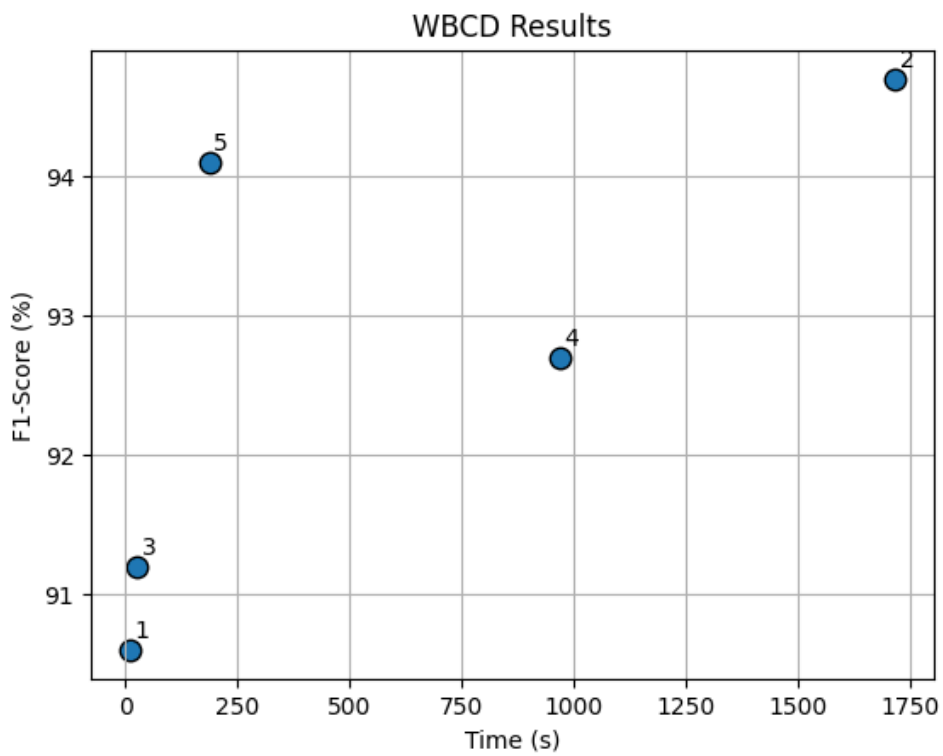


Figure 12 WBCD Dataset Pareto Chart

5.3 MNIST

- Activation function: Sigmoid
- Loss function: Multiclass cross entropy
- initial_nodes = [100]

- minibatch_size = 32
- epoch = 10
- partial_PB_epoch = 5
- patience = 4
- max_iterations = 3
- enable_last_two_layers = int(max_iterations - 1)

In Table 6, similar to the results of the two previous experiments, Simple BP dominates in speed while Full VNS-BP dominates in accuracy. In terms of ANN structure, only Simple BP retains its initial structure, while the other models' solution varies widely.

Table 6 MNIST Dataset Results

MNIST (70,000x784) (average of 3 runs)	Execution Time (s)	Average Accuracy (%)	Best Structures
1. Simple BP	2914	89.9%	[784, 100, 10]
2. Full VNS-BP	21,813	98.2%	[784, 550, 10]
3. Last layer VNS-BP	6,490	95.9%	[784, 250, 100, 10]
4. Last two layer VNS-BP	7,219	96.1%	[784, 300, 100, 10]
5. Switch layers VNS-BP	6,861	97.3%	[784, 650, 300, 10]

In Figure 13, it is again clear that models 1 and 2 dominate in one dimension. We can also see that model 4 is dominated by model 5 in both dimensions, while models 3 and 5 yield viable results. However, since the execution time is magnitudes larger than in previous

experiments, it may be wise to consider the other models' results. Although model 5 does not dominated any dimension, it achieves a 0.9% lower accuracy in only a third of the time model 2 required.

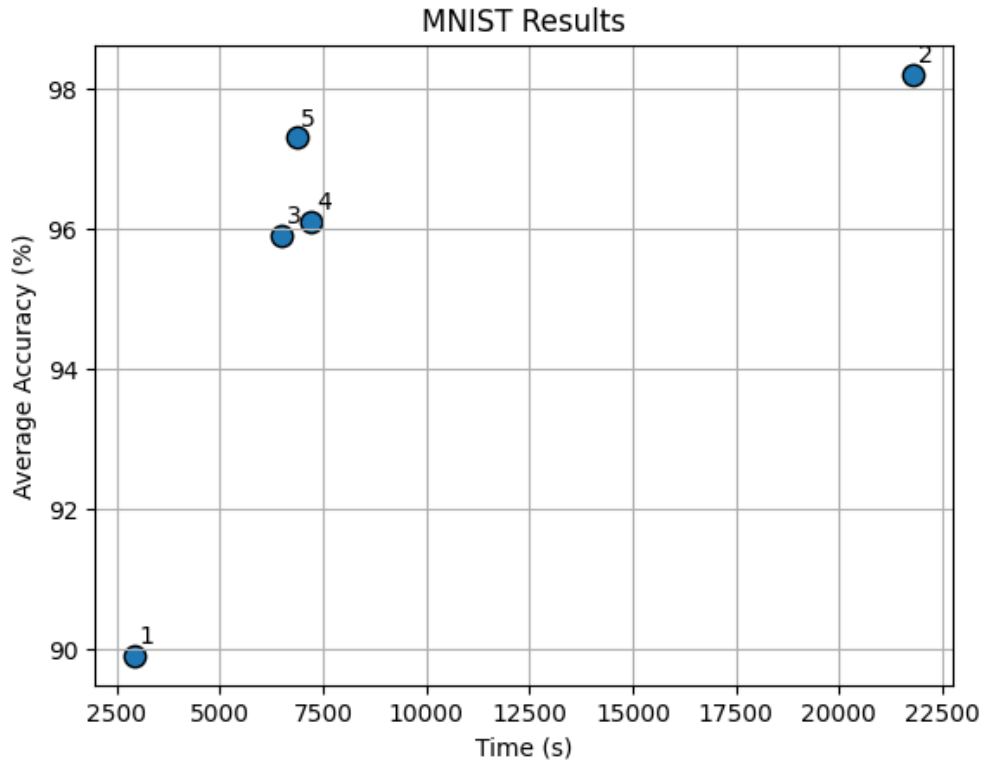


Figure 13 MNIST Dataset Pareto Chart

A detailed summary of the results of the experiments conducted in this study can be found in the Appendix.

5.4 Findings

- VNS is not necessary for small datasets since simple BP is able to find the optimal weights for an initial nodes given enough iterations.

- As the datasets become larger, VNS becomes more relevant. With Full VNS-BP achieving 98.2% accuracy in the MNIST dataset.
- When the dataset is large enough, switch VNS-BP achieves similar accuracy to Full VNS-BP but at much smaller time frame ($1/3^{\text{rd}}$ in the case of MNIST) for the smallest tradeoff in accuracy in comparison to the other versions.

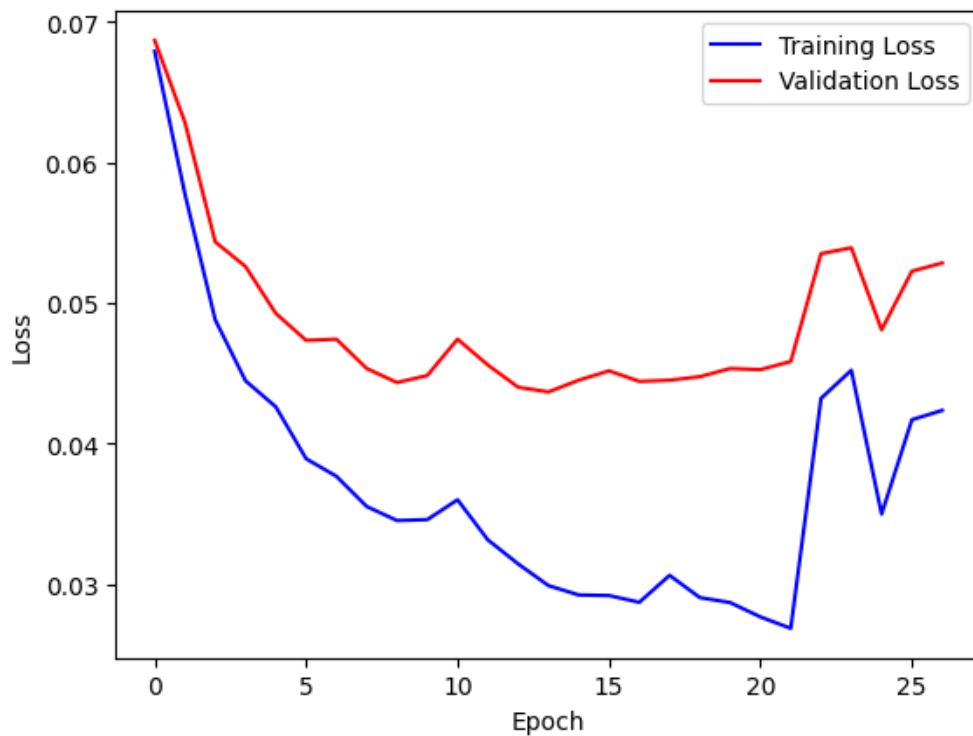


Figure 14 Sample Loss vs. Epoch Graph

In figure 14, we can see how when we don't get any improvements for a few epochs we stop the run. And we use the best weights and bias to predict the classes.

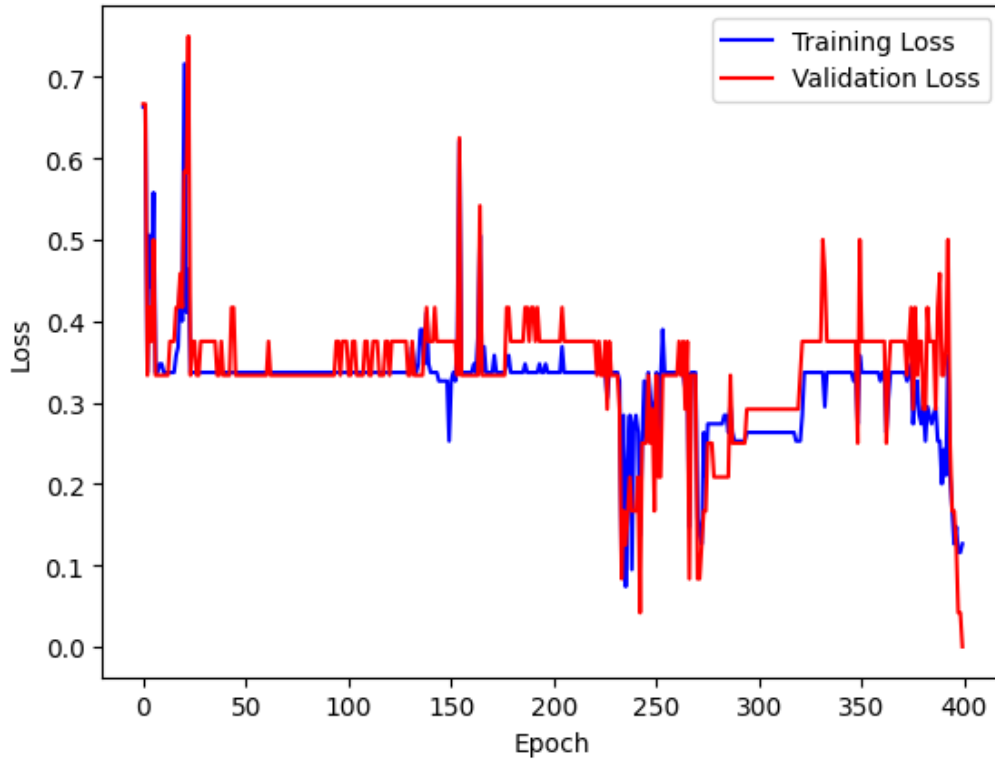


Figure 15 Sample Loss vs. Epoch Graph 2

In figure 15, we can see that at each peak the VNS algorithm picks a new candidate and then proceeds to gradually improve the validation loss until either the number of epochs is reached, or early stopping is reached due to the patience parameter.

5.5 Win/loss/ties

The win/loss/tie information can provide valuable insights into the usefulness and effectiveness of the proposed approach for ANN structure optimization. By comparing the performance of different versions of the approach against each other and the control scenario, we can assess their relative strengths and weaknesses. Let's break down the implications of each outcome:

Wins: When a version of the proposed approach achieves better results (e.g., higher accuracy, F1-score) compared to other versions or the control scenario, it indicates the efficacy of specific modifications or enhancements introduced in that version. Wins demonstrate that the proposed changes in the ANN structure, guided by the VNS exploration, lead to improved performance. Multiple wins by a particular version suggest its superiority and can indicate its potential for practical implementation.

Losses: If a version of the proposed approach performs worse than other versions or the control scenario, it highlights potential weaknesses or limitations. Losses can reveal areas where the modifications introduced through VNS exploration might have resulted in suboptimal ANN structures. Analyzing losses provides valuable feedback for further refinement or adjustment of the approach. It helps identify which aspects of the modifications might be causing a decline in performance and guides improvements for future iterations.

Ties: When two or more versions of the proposed approach achieve similar results without a statistically significant difference compared to each other or the control scenario, ties can indicate that certain modifications or variations in the ANN structure have minimal impact on performance. Ties suggest that exploring specific areas of the ANN structure may not yield substantial improvements for the given dataset or optimization problem. They also highlight the presence of a baseline performance level that can be achieved through simpler methods (such as simple backpropagation in this case).

Table 7 Iris Dataset Win/Loss/Ties

Approach	Iris1	Iris2	Iris3	Iris4	Iris5	Score
Iris1	-	1	1	1	1	4
Iris2	1	-	1	1	1	4
Iris3	1	1	-	1.5	1.5	5
Iris4	1	1	0.5	-	0.5	3
Iris5	1	1	0.5	1.5	-	4

Table 8 WBCD Dataset Win/Loss/Ties

Approach	WBCD1	WBCD2	WBCD3	WBCD4	WBCD5	Score
WBCD1	-	1	1	1	1	4
WBCD2	1	-	1	1	1	4
WBCD3	1	1	-	1	1	4
WBCD4	1	1	1	-	0	3
WBCD5	1	1	1	2	-	5

Table 9 MNIST Dataset Win/Loss/Ties

Approach	MNIST1	MNIST2	MNIST3	MNIST4	MNIST5	Score
MNIST1	-	1	1	1	1	4
MNIST2	1	-	1	1	1	4
MNIST3	1	1	-	1	1	4
MNIST4	1	1	1	-	0	3
MNIST5	1	1	1	2	-	5

- The scores obtained for each approach across different datasets indicate a consistent performance pattern. The scores obtained for the same approach on different datasets are generally similar, suggesting that the effectiveness of an approach is relatively stable across different domains.
- The scores between different versions of the approach within the same dataset show similarities in performance. This indicates that the modifications or variations

introduced in those versions have a comparable impact on the performance of the approach for a particular dataset.

- The scores allow for a comparative analysis of the approaches across datasets. Approaches with higher scores exhibit better overall performance, indicating their potential superiority for ANN structure optimization. For example, the scores for WBCD4 and MNIST4 are lower compared to other approaches, suggesting potential weaknesses or less effective modifications in those versions.
- Based on the scores, we can identify the approaches that consistently perform well across datasets. Approaches with the highest scores (e.g., Iris3, WBCD5, MNIST5) demonstrate superior overall performance and can be considered as strong candidates for practical applications or further research.

5.6 Discussion

In this study, we proposed a novel metaheuristic approach for artificial neural network structure optimization by combining variable neighborhood search with backpropagation (BP). We compared the performance of four different versions of our approach against a control scenario using simple backpropagation without VNS. The results obtained on three diverse datasets (Iris, WBCD, and MNIST) provide valuable insights into the effectiveness of our approach.

Our findings demonstrate the significant advantages of incorporating VNS into the BP algorithm. Across all datasets, the VNS-BP versions consistently outperformed the control scenario in terms of accuracy, F1-score, or both. This indicates that the exploration of different ANN structures through VNS contributes to improved model performance.

Regarding execution time, the incorporation of VNS did result in increased computational requirements compared to simple BP. However, the trade-off was justified by the substantial enhancements in accuracy and F1-score achieved by the VNS-BP versions. The time invested in exploring alternative ANN structures through VNS proved beneficial for finding better solutions.

Analyzing the dataset-specific performance, we observed interesting patterns. In the case of the Iris dataset, all versions of VNS-BP outperformed the control scenario, showcasing the ability of VNS to fine-tune the ANN structure and achieve higher accuracy. The marginal difference in accuracy between the different VNS-BP versions indicates that exploring alternative ANN architectures beyond the last layer does not provide significant improvements in this relatively simple dataset.

For the WBCD dataset, the VNS-BP versions achieved higher F1-scores compared to simple BP, demonstrating the effectiveness of VNS in capturing more nuanced patterns in the data. The full VNS-BP version showed the highest F1-score, indicating that exploring the entire ANN structure led to better classification of malignant and benign breast tumors. However, it is worth noting that the full VNS-BP version incurred a significantly longer execution time, which may limit its practical applicability.

In the case of the MNIST dataset, the VNS-BP versions consistently outperformed simple BP in terms of accuracy. This indicates that exploring alternative ANN structures through VNS contributes to improved digit recognition. The last one/two layer switch VNS-BP version achieved the highest accuracy while maintaining a reasonable execution time, suggesting that modifying the last few layers of the ANN architecture can yield significant improvements.

Overall, our results highlight the effectiveness of the proposed metaheuristic approach for ANN structure optimization. By combining VNS with BP, our approach consistently achieved superior performance compared to simple BP on all tested datasets. The improvements in accuracy and F1-score demonstrate the potential of our approach in enhancing the capabilities of neural networks for classification tasks.

5.7 Threats to Validity

Threats to validity refer to potential factors or conditions that could undermine the accuracy, credibility, or generalizability of the results obtained in a study. They encompass issues such as selection bias, measurement errors, confounding variables, and sample representativeness. Recognizing and addressing these threats is crucial because they have the potential to distort the findings, leading to incorrect conclusions or limiting the applicability of the results to broader contexts. By identifying and mitigating these threats, the reliability and validity of a study can be enhanced, thus ensuring that the outcomes accurately reflect the relationships being investigated and can be confidently applied to real-world scenarios.

5.7.1 Internal Threats

The first threat to internal validity is the initialization, which concerns aspects such as the initial structure of the model, the learning rate, and the weights and biases assigned to its components. These factors can impact the convergence and stability of the algorithm. Secondly, the shake procedure alters variables like layer size, adding or removing layers, and making adjustments up to ± 100 units. The specifics of the possible alterations caused by the shake procedure have a significant effect on the outcome of the model. Lastly, the termination conditions, encompassing variables like epochs, maximum iterations, and patience thresholds, influence when the algorithm stops iterating.

However, the five models were tested using the same inputs of the parameters mentioned above. Therefore, the variation due to these parameters is not relevant to the scope of this study.

5.7.2 External Threats

The study tests the models on only binary and multiclassification tasks, which may cause the study's findings to lack generalizability to a broader range of problems. However, according to the literature review, we find that it is common practice to focus on binary and multiclassification tasks as they are the fundamental building blocks for evaluating algorithm performance and establishing a solid foundation for further research and comparison.

5.7.3 Construct Threats

The measures used in this study are common use in the literature and are therefore not a threat to construct validity.

5.7.4 Conclusion Threats

The Pareto front graph does not yield a conclusive answer to the best approach, and therefore the conclusions drawn based solely on the graph may not be entirely correct. However, in this study, the conclusions drawn based on the graph were substantiated using the win/loss/ties test to provide a more comprehensive and well-rounded assessment of the different approaches' performance and their relative strengths.

5.8 Future Work

In this study, we have demonstrated the effectiveness of VNS for optimizing the structure of an ANN. However, there are several avenues for further exploration and improvement in the context of using VNS for ANN optimization.

One potential direction for future work is to extend the scope of optimization beyond the ANN structure alone. While our current approach focused on optimizing the number of layers and nodes in the network, it would be interesting to explore the optimization of other model parameters, such as the learning rate and minibatch size. These parameters can significantly impact the performance of the ANN, and optimizing them through VNS could potentially lead to further improvements in accuracy and convergence speed.

Another aspect to consider is the impact of the maximum number of iterations (`max_iterations`) on the performance of VNS. In our experiments, we observed that increasing the `max_iterations` beyond a certain threshold significantly extended the

runtime, making it infeasible for practical applications. However, it is possible that with future coding optimizations and algorithmic improvements, the computational efficiency of VNS can be enhanced. This would allow for higher max_iterations to be explored, potentially leading to even better solutions for ANN optimization.

Furthermore, it would be valuable to investigate the integration of VNS with other optimization techniques or metaheuristics. Combining VNS with algorithms such as Genetic Algorithms or Particle Swarm Optimization could potentially leverage their strengths and further enhance the exploration and exploitation capabilities during the optimization process.

Additionally, the evaluation was limited to three datasets of varying sizes and complexities. Further investigations on a wider range of datasets and larger-scale problems would provide a more comprehensive understanding of the approach's generalizability and scalability. Exploring the application of VNS to larger and more complex datasets could provide insights into its scalability and robustness. Testing the performance of VNS on real-world datasets with higher dimensions and more diverse patterns would provide a more comprehensive evaluation of its effectiveness and generalizability.

5.9 Conclusion

The Variable Neighborhood Search (VNS) approach has demonstrated its effectiveness in optimizing the structure of Artificial Neural Networks (ANNs), resulting in significant improvements in classification performance. The experiments conducted using the VNS approach revealed a remarkable 8-9% increase in classification accuracy compared to the initial runs using the same parameters methods.

By leveraging the VNS metaheuristic method, the ANN structure was optimized in a manner that enhanced its ability to capture complex patterns and relationships within the data. The VNS algorithm autonomously searched for the best combination of input attributes and fitness values, maximizing the overall performance of the ANN.

The success of the VNS approach can be attributed to its ability to explore different network architectures and hyperparameter configurations, leading to the discovery of highly efficient configurations that were previously unexplored. The incorporation of the VNS method allowed for an exhaustive exploration of the solution space, effectively fine-tuning the ANN structure to achieve superior classification results.

The 8-9% improvement in classification accuracy obtained through the VNS approach underscores its significance in enhancing the predictive power of ANNs. This improvement can have substantial implications in various domains, such as healthcare, finance, and image recognition, where accurate classification is crucial for decision-making and problem-solving.

Overall, our study demonstrates the effectiveness of VNS in optimizing neural network structures, particularly in the context of smaller networks. The results provide valuable insights into the performance of VNS in comparison to benchmark techniques, shedding light on its strengths and limitations. These findings contribute to the broader understanding of optimization approaches for neural networks and inform future research directions for improving optimization algorithms in the context of complex network structures.

Appendix

Iris summary:

Simple BP				
run#	LR	Time	Accuracy	Structure
1	0.4	10.35	96.7%	[100]
2	0.5	4.6	96.7%	[100]
3	0.5	9.96	100.0%	[100]
4	0.3	9.26	96.7%	[100]
5	0.4	9.28	100.0%	[100]
average	0.4	8.69	98.0%	[100]

Full BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.3	101.11	96.7%	[250, 100]
2	0.4	103.21	100.0%	[250, 100]
3	0.5	1.21	100.0%	[100]
4	0.5	106.65	100.0%	[150, 100]
5	0.4	5.44	100.0%	[100]
average	0.4	63.524	99.3%	[X, 100]

Last Layer BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.4	8.93	96.7%	[100]
2	0.3	7.29	96.7%	[100]
3	0.4	10.77	100.0%	[100]
4	0.3	6.93	100.0%	[100]
5	0.3	10.99	100.0%	[100]
average	0.3	8.982	98.7%	[100]

Last Two Layers BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.3	54.13	100.0%	[50, 100]
2	0.3	119.39	100.0%	[100]
3	0.3	10.39	93.3%	[100]
4	0.4	140.76	100.0%	[100]
5	0.5	2.29	100.0%	[100]
average	0.3	56.392	98.7%	[100]

Switch Layers BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.3	6.44	100.0%	[100]
2	0.4	6.51	93.3%	[100]
3	0.5	1.35	100.0%	[100]
4	0.3	11.41	100.0%	[100]
5	0.5	14.91	100.0%	[100]
average	0.3	8.124	98.7%	[100]

WBCD summary:

Simple BP				
run#	LR	Time	Accuracy	Structure
1	0.115	8.94	90.5%	[100]
2	0.335	14.38	91.1%	[100]
3	0.5	5.71	90.3%	[100]
average	0.317	9.676667	90.6%	[100]

Full BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.06	1975.31	93.9%	[200, 300, 100]
2	0.05	1120.9	95.6%	[300, 100]
3	0.04	2061.89	94.6%	[350, 100]
average	0.05	1719.367	94.7%	[X, 100]

Last Layer BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.06	24.14	92%	[100]
2	0.05	30.94	90%	[100]
3	0.06	23.29	92%	[100]
average	0.06	26.12333	91.2%	[100]

Last Two Layers BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.04	259.5	94.7%	[50, 100]
2	0.06	901.02	93.7%	[100]
3	0.06	1746.52	89.5%	[150]
average	0.06	969.0133	92.7%	[100]

Switch Layers BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.06	353.6	93.8%	[300, 100]
2	0.05	27.97	93.8%	[100]
3	0.04	182.05	94.7%	[100]
average	0.05	187.8733	94.1%	[100]

MNIST Summary:

Simple BP				
run#	LR	Time	Accuracy	Structure
1	0.05	2913	90.1%	[100]
2	0.04	2966	89.7%	[100]
3	0.05	2863	89.8%	[100]
average	0.05	2914	89.9%	[100]

Full BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.06	22547	98.7%	[100]
2	0.04	21903	97.9%	[100, 100]
3	0.05	20988	98.1%	[50, 100]
average	0.05	21812.67	98.2%	[X, 100]

Last Layer BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.05	6024	95.6%	[100]
2	0.04	6521	96.2%	[100]
3	0.04	6925	95.8%	[100]
average	0.04	6490	95.9%	[100]

Last Two Layers BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.04	6812	96.0%	[50, 100]
2	0.04	8151	96.2%	[100]
3	0.05	6695	96.0%	[100]
average	0.04	7219.333	96.1%	[100]

Switch Layers BP VNS				
run#	LR	Time	Accuracy	Structure
1	0.05	6335	97.0%	[100]
2	0.04	6713	98.0%	[100]
3	0.04	7534	98.9%	[100]
average	0.04	6860.667	97.9%	[100]

References

- [1] S. Ruder, “An overview of gradient descent optimization algorithms,” pp. 1–14, 2016, [Online]. Available: <http://arxiv.org/abs/1609.04747>.
- [2] P. Netrapalli, “Stochastic Gradient Descent and Its Variants in Machine Learning,” *J. Indian Inst. Sci.*, vol. 99, no. 2, pp. 201–213, 2019, doi: 10.1007/s41745-019-0098-4.
- [3] B. T. Polyak and A. B. Juditsky, “Acceleration of stochastic approximation by averaging,” *SIAM J. Control Optim.*, vol. 30, no. 4, pp. 838–855, 1992, doi: 10.1137/0330046.
- [4] F. Bach and E. Moulines, “Non-asymptotic analysis of stochastic approximation algorithms for machine learning,” *Adv. Neural Inf. Process. Syst.* 24 25th Annu. Conf. Neural Inf. Process. Syst. 2011, NIPS 2011, no. 2, pp. 1–9, 2011.
- [5] F. Bach and E. Moulines, “Non-strongly-convex smooth stochastic approximation with convergence rate $O(1/n)$,” *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2013.
- [6] A. Dieuleveut, A. Durmus, and F. Bach, “Bridging the gap between constant step size stochastic gradient descent and markov chains,” *Ann. Stat.*, vol. 48, no. 3, pp. 1348–1382, 2020, doi: 10.1214/19-AOS1850.
- [7] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, “Optimal distributed online prediction using mini-batches,” *J. Mach. Learn. Res.*, vol. 13, pp. 165–202, 2012.
- [8] P. Jain, P. Netrapalli, S. M. Kakade, R. Kidambi, and A. Sidford, “Parallelizing stochastic gradient descent for least squares regression: Mini-batching, averaging, and model misspecification,” *J. Mach. Learn. Res.*, vol. 18, pp. 1–42, 2018.

- [9] G. Lan, An optimal method for stochastic composite optimization, vol. 133, no. 1–2. 2012.
- [10] P. Jain, S. M. Kakade, R. Kidambi, P. Netrapalli, and A. Sidford, “Accelerating Stochastic Gradient Descent For Least Squares Regression,” vol. 75, no. 1, pp. 1–60, 2017, [Online]. Available: <http://arxiv.org/abs/1704.08227>.
- [11] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” *Adv. Neural Inf. Process. Syst.*, vol. 4, no. January, pp. 2933–2941, 2014.
- [12] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The marginal value of adaptive gradient methods in machine learning,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 4149–4159, 2017.
- [13] N. S. Keskar and R. Socher, “Improving Generalization Performance by Switching from Adam to SGD,” no. 1, 2017, [Online]. Available: <http://arxiv.org/abs/1712.07628>.
- [14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.
- [15] I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, pp. 1–16, 2017.
- [16] S. Merity, C. Xiong, J. Bradbury, and R. Socher, “Pointer sentinel mixture models,” *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, 2017.

- [17] V. K. Ojha, A. Abraham, and V. Snášel, “Metaheuristic design of feedforward neural networks: A review of two decades of research,” *Eng. Appl. Artif. Intell.*, vol. 60, no. September 2021, pp. 97–116, 2017, doi: 10.1016/j.engappai.2017.01.013.
- [18] H. K. Lam, S. H. Ling, F. H. F. Leung, and P. K. S. Tam, “Tuning of the structure and parameters of neural network using an improved genetic algorithm,” *IECON Proc. (Industrial Electron. Conf.)*, vol. 3, pp. 25–30, 2001, doi: 10.1109/iecon.2001.976448.
- [19] G. Das, P. K. Pattnaik, and S. K. Padhy, “Artificial Neural Network trained by Particle Swarm Optimization for non-linear channel equalization,” *Expert Syst. Appl.*, vol. 41, no. 7, pp. 3491–3496, 2014, doi: 10.1016/j.eswa.2013.10.053.
- [20] M. Meissner, M. Schmuker, and G. Schneider, “Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training,” *BMC Bioinformatics*, vol. 7, pp. 1–11, 2006, doi: 10.1186/1471-2105-7-125.
- [21] L. Zhao and F. Qian, “Tuning the structure and parameters of a neural network using cooperative binary-real particle swarm optimization,” *Expert Syst. Appl.*, vol. 38, no. 5, pp. 4972–4977, 2011, doi: 10.1016/j.eswa.2010.09.154.
- [22] R. C. Green, L. Wang, and M. Alam, “Training neural networks using Central Force Optimization and Particle Swarm Optimization: Insights and comparisons,” *Expert Syst. Appl.*, vol. 39, no. 1, pp. 555–563, 2012, doi: 10.1016/j.eswa.2011.07.046.
- [23] N. S. Jaddi, S. Abdullah, and A. R. Hamdan, “Optimization of neural network model using modified bat-inspired algorithm,” *Appl. Soft Comput. J.*, vol. 37, pp. 71–86, 2015, doi: 10.1016/j.asoc.2015.08.002.
- [24] E. Hadavandi, S. Mostafayi, and P. Soltani, “A Grey Wolf Optimizer-based neural

- network coupled with response surface method for modeling the strength of siro-spun yarn in spinning mills,” *Appl. Soft Comput. J.*, vol. 72, pp. 1–13, 2018, doi: 10.1016/j.asoc.2018.07.055.
- [25] Y. Wang, H. Liu, Z. X. Yu, and L. P. Tu, “An improved artificial neural network based on human-behaviour particle swarm optimization and cellular automata,” *Expert Syst. Appl.*, vol. 140, 2020, doi: 10.1016/j.eswa.2019.112862.
- [26] E. Alba, D. Anguita, A. Ghio, and S. Ridella, “Using variable neighborhood search to improve the support vector machine performance in embedded automotive applications,” *Proc. Int. Jt. Conf. Neural Networks*, pp. 984–988, 2008, doi: 10.1109/IJCNN.2008.4633918.
- [27] Y. Uzun, H. A. ARIKAN Seydişehir C M F, and G. Tezel, “Rule Extraction From Training Artificial Neural Network Using Variable Neighbourhood Search For Wisconsin Breast Cancer,” *J. Multidiscip. Eng. Sci. Technol.*, vol. 3, no. 8, pp. 2458–9403, 2016.
- [28] T. Araújo, G. Aresta, B. Almada-Lobo, A. M. Mendonça, and A. Campilho, “Improving convolutional neural network design via variable neighborhood search,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10317 LNCS, pp. 371–379, 2017, doi: 10.1007/978-3-319-59876-5_41.
- [29] <https://www.lightly.ai/post/which-optimizer-should-i-use-for-my-machine-learning-project> |

Vitae

Name	:Alhassan Mohammedhany K. Aldabbagh
Nationality	:Saudi Arabian
Date of Birth	:1/18/1998
Telephone	:+966 504550243
Email	:alhassan.aldabbagh@gmail.com
Address	:Jeddah, Saudi Arabia. Postal Code: 23621, Bulding Number: 6781
Academic Background	:BSc and MSc in Industrial & Systems Engineering from KFUPM. Published a conference paper and presented it in SNAMS2023 sponsored by IEEE UAE section titled “Metaheuristic Approach to Artificial Neural Network Optimization”.
Teaching Experience	:Taught the Stochastic Systems Simulation (ISE405) lab course for two terms in KFUPM. Responsibilities include lab instruction, exam creating and grading, and student guidance and supervision.