UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Optimization Problems Using the Particle Swarm Optimization Algorithm

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Chancay Moreira Stalyn Javier

Tutor:

Ph.D. Rigoberto Fonseca

Co-Tutor:

Ph.D. Israel Pineda

Urcuquí, noviembre 2022

Autoría

Yo, **Stalyn Javier Chancay Moreira**, con cédula de identidad 180514577, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, noviembre 2022.

Stalyn Javier Chancay Moreira

CI: 1805145677

Autorización de publicación

Yo, **Stalyn Javier Chancay Moreira**, con cédula de identidad 1805145677, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, noviembre 2022.

Stalyn Javier Chancay Moreira

CI: 1805145677

Dedication

"To my family, my friends, and co-tutor because their motivation and support helped me to reach this point in my life."

Stalyn Javier Chancay Moreira

Acknowledgment

Firstly, I would like to acknowledge Rigoberto Fonseca, Ph.D., who collaborated in the final stages of my project. His comments and guidance supported me in my work. His support was undoubtedly appreciated, particularly in the drafting and structure of the present document.

Furthermore, I want to convey my profound thanks to my co-adviser, Israel Pineda, Ph.D., for his unwavering support of my studies and research and his perseverance, responsibility, inspiration, and enthusiasm. Moreover, I completed this thesis thanks to his advice, which I used throughout my research and writing. Besides, I want to thank Md Sakibul Islam, University of New Mexico, who accompanied me in developing this project.

I want to express a special thanks to my friends, Luis Murillo, Evelyn Callataxi, Valeria Lucero, Matéo Oleas, Liseth Encarnación, Carlos Mendoza, and Belén Maldonado, with whom I shared the most significant moments of my university life.

Lastly, I want to thank my mom (Nieve Moreira) and brothers (Jonathan Chancay, Bryan Chancay, Jair Chancay, Josue Chancay, and Nicolás Chancay) for always inspiring and helping me through college. I would not have finished this phase of my life without their advice. Further, I thank my darling Fernanda Caiza, who, together with my family, has always been a source of strength and inspiration during good and bad times.

Stalyn Javier Chancay Moreira

Resumen

En la actualidad existen diversas técnicas y modelos para el entrenamiento de redes neuronales artificiales, como la arquitectura del perceptrón multicapa. El perceptrón multicapa tiene un mapeo no lineal entre entradas y salidas de NN. Además, Backpropagation es el algoritmo tradicional para entrenar una red neuronal. Por otro lado, en los últimos años se han implementado algoritmos metaheurísticos inspirados en la naturaleza para optimizar los parámetros de ANN. Un algoritmo popular para esta tarea es PSO, que tiene una emocionante versión cuántica (QDPSO). Por lo tanto, esta tesis propone la integración de QDPSO en un perceptrón multicapa para problemas de clasificación y lo compara con PSO, PSO-bound, L-BFGS, Adam y SGD. Las contribuciones de este trabajo son la arquitectura e integración del QDPSO, la validación del modelo propuesto comparándolo con optimizadores basados en metaheurísticas y gradiente utilizando conjuntos de datos benchmark, y el análisis del comportamiento de entrenamiento aumentando el número de clases y muestras del conjunto de datos circular. Además, proponemos una técnica de clasificación de imágenes usando Isomp como algoritmo de reducción. Isomap reduce seis veces las características de la imagen para la capa de entrada. Además, se compara con MSD, TSNE y PCA utilizando los conjuntos de datos de cáncer de mama e iris. Finalmente, los resultados de validación y comparación demostraron que la arquitectura y la técnica propuesta en esta tesis tienen una excelente clasificación de los conjuntos de datos benchmark y MCW. Además, el optimizador QDPSO tiene una convergencia más rápida y un comportamiento admirable durante el entrenamiento para conjuntos de datos balanceados.

Palabras Clave:
Red Neuronal Artificial, Retropropagación, Optimización de Enjambre de Partículas Delta de Comportamiento Cuántico, Datos Benchmark y Meteorológicos Multiclase.

Abstract

Several techniques and models for training artificial neural networks exist, such as the architecture of the multi-layer perceptron. Multi-layer perceptron has a non-linear mapping between inputs and outputs of the neural network. Furthermore, Backpropagation is the traditional algorithm for training a neural network. On the other hand, in recent years, nature-inspired metaheuristic algorithms have been implemented to optimize the parameters of ANN. A popular algorithm for this task is PSO, which has a quantum version (QDPSO). Thus, this thesis proposes the integration of QDPSO in a multi-layer perceptron for classification problems and compares it with PSO, PSO-bound, L-BFGS, Adam, and SGD. The contributions of this work are the architecture and integration of the QDPSO, validation of the model proposed comparing with optimizers based on metaheuristics and gradient using benchmark datasets, and analysis of the training behavior increasing the classes and samples number of the circle dataset. Besides, we propose a technique for image classification using Isomap as a reduction algorithm. Isomap reduces six times the image features for the input layer. Also, it is compared with MSD, TSNE, and PCA using the iris and breast cancer datasets. Finally, the validation and comparison results demonstrated that the architecture and technique proposed in this thesis have an excellent classification of the benchmark and MCW datasets. Moreover, the QDPSO optimizer has faster convergence and adequate behavior during the training for balanced datasets.

Keywords:
Artificial Neural Network, Backpropagation, Quantum-Behaved Delta Particle Swarm Optimization.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

In the last decade, neural networks (NN), also known as artificial neural networks (ANNs), have become the most popular family of machine learning algorithms. Artificial Neural Networks (ANN) as a computational model have existed since the middle of the last century but have gained momentum thanks to improvements in techniques and technology. The name of ANN and structure is inspired by the human brain, imitating how biological neurons signal to one another [6] [7]. As a result, the artificial neural networks have expanded in many fields of scientific research, such as biology [8] [9], chemistry [10] [11], physic [12] [13], ecology [14] [15], and others. Their advances are so overwhelming that they can currently do character recognition [16] [17], image recognition [18], voice recognition [19], text generation [20], language translation [21] [22], autonomous driving [23], disease prognosis [24] [25], and so on. ANNs are also well suited to assisting individuals in real-world scenarios with complex difficulties. For example, they can learn and model the relationships between inputs and outputs that are nonlinear and complex and reveal hidden relationships, patterns, and predictions, among others.

Hilton *et al.* in 1986 proposed the first method for training an artificial neural network: backpropagation (BP) [26]. BP is a technique for adjusting the weights of ANN depending on the error rate recorded in the previous epoch, *i.e.*, iteration. Appropriately tweaking the weights may lower error rates and improve the reliability of the model by broadening its applicability. Several methods perform this task called optimization. The optimizer al-

gorithm optimizes the objective function that requires minimizing its parameters. In other words, it changes the attributes of the artificial neural network, such as weights, to reduce the losses by optimizing a minimization problem. There are many optimization algorithms, such as gradient methods, evolutionary algorithms, and metaheuristics. However, the most common optimizer for training a neural network is the gradient descent with its variants [27] [28].

On the other hand, recently, it has been proposed to train an artificial network with metaheuristic algorithms such as Particle Swarm Optimization (PSO) [3] [4]. PSO was proposed in 1995 by Kennedy and Eberhart [29]. It is a heuristic optimization method aimed at finding global minimums or maximums. The behavior of bird flocks inspired PSO, where the movement of each individual is the result of each decision individual with the behavior of the rest. Otherwise, PSO has several variations, such as guaranteed convergence PSO [30], Non-dominated Sorting PSO [31], improved PSO [32], and so on. However, an exciting implementation, QDPSO, was proposed by Sun *et al.* in 2004 [33]. QDPSO is a quantum version of PSO, which uses a Contraction-Expansion parameter called Delta. Thus, the project proposal integrates QDPSO for training ANNs, using benchmark datasets, classification metrics, and different optimizers for validation and comparison. Furthermore, it proposes a technical for image classification using a features-reducing algorithm and analyses the behavior of QDPSO during the training, increasing the number of samples and classes.

1.2 Problem statement

Training ANNs is complex because the error surface is non-convex, comprises local minima and flat patches, and is highly multi-dimensional. Moreover, the weights of a neural network with hidden layers are highly interdependent. In this sense, many researchers improve the training method using different techniques such as parameter initialization, loss and activation functions, data preprocessing, optimizers, and so forth. For these reasons, this project presents an integration of QDPSO for training ANNs.

1.3 Objectives

1.3.1 General Objective

Train an artificial neural network with Particle Swarm Optimization using particles based on quantum behavior.

1.3.2 Specific Objectives

- Implement an artificial neural network training using QDPSO.
- Benchmark the proposed optimizer with other optimizers found in the literature review using different datasets, classes, and dimensions.
- Tune the parameters of the algorithm to improve the training result.

Chapter 2

Theoretical Framework

This chapter introduces the fundamental ideas required to comprehend the current work. So, it starts with the bases of artificial neural networks and their current status. After that, it describes and explains the mathematical background of the optimizers used in this work with their principal features.

2.1 Artificial Neural Network

In 1958, it was made known as a hardware system that learns from data with a perceptron program [34]. Then, in 1959 Hubel *et al.* did some experiments with cat neurons. The idea was to see how the activity in neural network responds when changing a slide manually [35]. This paper won the Nobel prize in psychology and vision. After that, in 1969, Minsky showed that the perceptron does not learn from the XOR function [26]. Then, Fukushima proposed a complex architecture with pooling and convolutions that is not a practical training algorithm [36]. However, in 1986, Hilton *et al.* proposed the backpropagation, a differentiation function that allows us to train a perceptron with multiple layers [26]. After that, LeCun *et al.* combined the Fukushima architecture with BP and proposed the Convolutional Neural Network (CNN) [37]. It is important to remark that Hilton is the father of BP, and LeCun is the father of CNN. Finally, in 2012, AlexNet [38], a deep learning (DL) architecture that used multiple layers convolutions, and the ImageNet Challenge. Also, AlexNet accepted it at CVPR, a conference for computer vision. From there, the DL suffered an explosion, and the people used artificial neural network in image classification [39], object detection [40], pose recognition [41], and image segmentation [42].

ANN is based on the operation of a biological neural network [7]. Biological neural networks are made up of dendrites, cell bodies, and axons: the dendrites are responsible for capturing the nerve impulses emitted by other neurons. These impulses are processed in the soma and transmitted through the axon, which sends a nerve impulse to neighboring neurons.

In the case of artificial neurons, the sum of the inputs multiplied by the associated weights determines the neural input that the neuron receives. This value is processed within the neuron by an activation function that returns the value sent as its output. Like our brain, which consists of interconnected neurons, an ANN consists of interconnected artificial neurons, also known as nodes, grouped at different levels called layers. Three main parts of ANNs are the following

- An input layer that represents the inputs field.
- One or more hidden layers.
- An output layer that represents the target variables.

The input data is presented in the first layer. Then, the values are propagated from each neuron to each neuron in the next layer. Finally, the result from the output layer, see Figure 2.1.

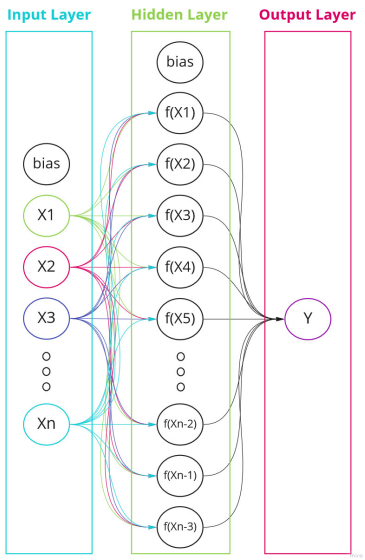


Figure 2.1: The main parts of the artificial neural network.

2.1.1 Functionality

Firstly, the basic unit of ANNs is a neuron. A neuron receives inputs, performs mathematical operations, and outputs one result, see Figure 2.2. Then, the combination of neurons forms an ANN, a system capable of learning.

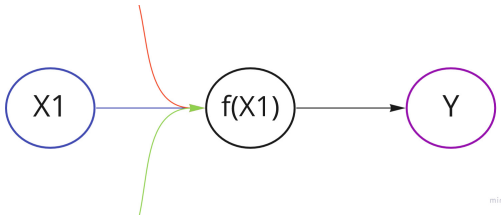


Figure 2.2: The unit basic of artificial neural network, a neuron.

Figure 2.3 sees an artificial neural network (multi-layer perceptron), which shows how the neurons have multiple input connections (input data). Inside, the neuron makes a weighted sum of input data. Each input weighting is given by each of the weights assigned to each input connection. In short words, each input is multiplied by a weight. Afterward, all the weighted inputs are added together with a bias b. Finally, the sum result is passed through an activation function.

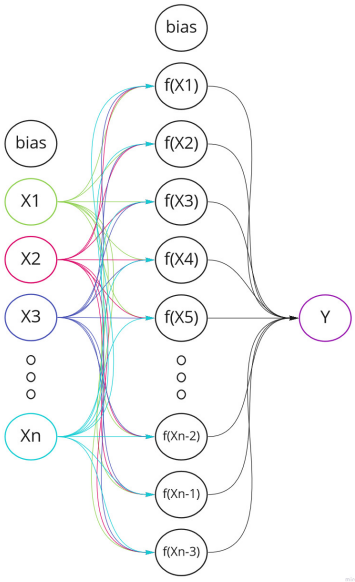


Figure 2.3: A Multi-Layer Perceptron.

2.1.2 Activation Functions

The activation functions allow an ANN to learn nonlinear mappings. Generally, it is between 0 to 1 or -1 to 1. Moreover, it can decide whether a neuron will fire or not, given all of its inputs. Several activation functions exist, such as sigmoid, hyperbolic tang, algebraic sigmoid, rectifier linear unit (ReLU), and so forth. Finally, it is presented the most common activation functions are described below [1].

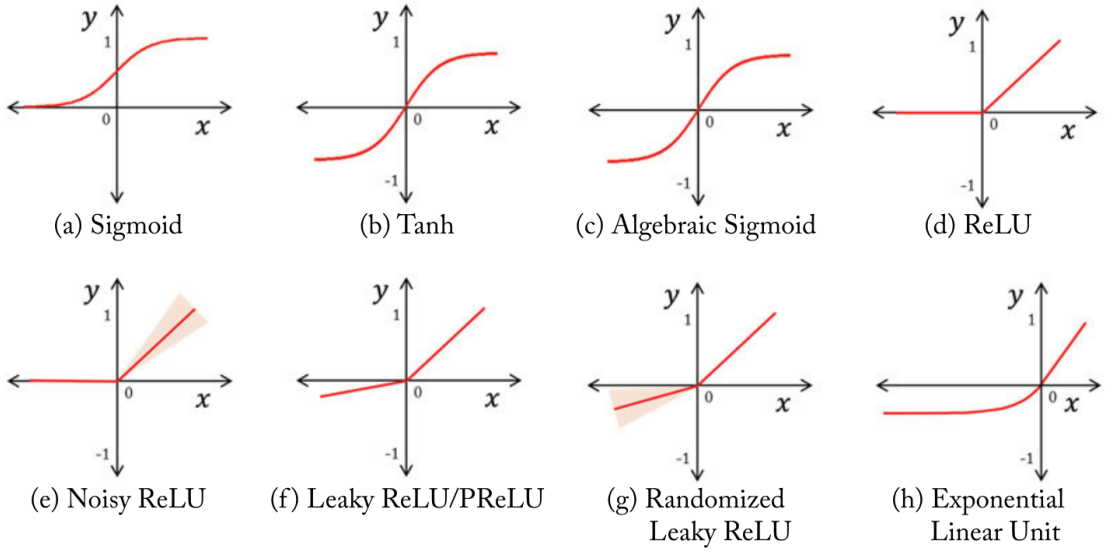


Figure 2.4: Some of the common activation functions that are used in deep neural networks, retrieved from [1]

Sigmoid Function

It catches an actual number as input and gives an output in the range of between 0 to 1. It is defined as

$$f_{\text{sigm}}(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

Algebraic Sigmoid

It maps the input within the range [-1, 1]. It is given by

$$f_{a-\text{sig}}(x) = \frac{x}{\sqrt{1+x^2}} \tag{2.2}$$

Rectifier Linear Unit

Rectifier Linear Unit (ReLU) is a simple activation function of particular practical importance because of its quick computation. If the input is harmful, the ReLU function transfers it to a 0; otherwise, it leaves the value untouched. ReLU can be represented as follow

$$f_{\text{relu}}(x) = \max(0, x) \tag{2.3}$$

Noisy ReLU

It adds a sample drawn from a Gaussian distribution with a mean zero and a variance that depends on the input value ($\sigma(x)$) in the positive input. It can be represented as follows

$$f_{n-rel}(x) = \max(0, x + \epsilon), \quad \epsilon \sim \mathcal{N}(0, \sigma(x)) \tag{2.4}$$

Leaky ReLU

It does not reduce the output to a zero value; instead, it outputs a down-scaled version of the negative input. It usually takes c as a small value (ex: 0.01) or a (parametric) that learns during the training. This function is represented as

$$\begin{aligned} f_{l-rel}(x) &= \begin{cases} x & \text{if } x > 0 \\ cx & \text{if } x \leq 0 \end{cases} \\ f_{p-rel}(x) &= \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases} \end{aligned} \tag{2.5}$$

Randomized Leaky ReLU

It selects the leak factor in the leaky ReLU function from a uniform distribution.

$$f_{r-rel}(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases} \tag{2.6}$$

The factor a is randomly chosen during training. Furthermore, set to a mean value during the test phase to get the contribution of all samples.

$$\begin{aligned} a &\sim \mathcal{U}(I, u) \quad \textit{during training} \\ a &= \frac{I+u}{2} \quad \textit{during testing.} \end{aligned} \tag{2.7}$$

Hyperbolic Tangent

It squashes the input values within the range of [-1, 1]. Moreover, it is represented as follows

$$f_{\tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.8}$$

Exponential Linear Unit

It has positive and negative values, and they try to push the mean activations toward zero (similar to batch normalization). The training process is speeding up while achieving better performance.

$$f_{elu}(x) = \begin{cases} x & \text{if } x > 0 \\ a\left(e^{x} - 1\right) & \text{if } x \leq 0 \end{cases} \tag{2.9}$$

2.1.3 Regularization

Any adjustment to a learning algorithm that we make, known as regularization, aims to lower both its generalization error and training error, but not always both [43] [44]. In short, regularization reduces generalization errors or training errors. Moreover, it helps to prevent the model from overfitting and improves the training optimization. Several regularizations exist, such as weight decay (L_1 and L_2), noisy inputs (Gaussian noise), data augmentation, and so forth. Nevertheless, we discuss L_1 and L_2 regularization and dropout in this work.

L_1 and L_2 Regularization

L_1 and L_2 regularization add a penalization term at the end of the loss function, which helps to prevent the model from overfitting. Equation 2.10 shows an example of regularization for the Cross-Entropy function.

$$L(W) = -\sum_{i=1}^{n} Y_i \log(p_i) + \gamma R(W) \tag{2.10}$$

where, p_i is the Softmax probability, Y is the truth label, and γ is the parameter control for the regularization term $R(W)$.

Dropout

Dropout prevents overfitting by randomly shutting down some output units. You can think of dropout as adding an extra layer to the forward process. Equation 2.11 shows a mathematical representation of the dropout, and Figure 2.5 is its graphic presentation.

$$\begin{gathered} \textit{Without Dropout} \\ Z^l = W^l A^{l-1} + b^l \\ A^l = g^l\left(Z^l\right) \\ \text{With Dropout} \\ Z^l = W^l A^{l-1} + b^l \\ A^l = g^l\left(Z^l\right) \\ A^l = D^l\left(A^l\right) \end{gathered} \tag{2.11}$$

where, W is the weight matrix, Z is the weighted sum, A is the activation function, b is the bias, and D is a dropout regularization.

2.2 Backpropagation

The popularity of ANNs skyrocketed in 1986 when Hilton *et al.* proposed BP. This work experimentally shows how a learning algorithm can readjust the parameters of an ANN, making it learn an internal representation of the information it is processing. Further-

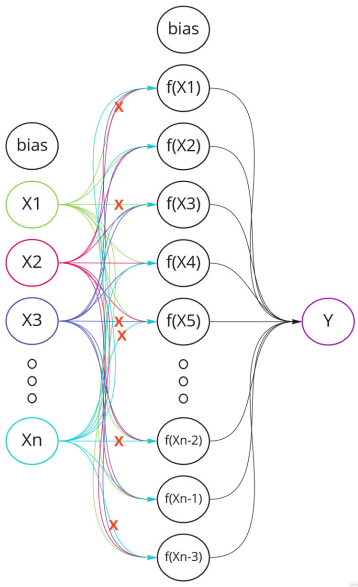


Figure 2.5: Droupout graphic representation.

more, the backward propagation algorithm calculates the partial derivatives of the cost concerning each network parameter. In short, it computes the cost derivative concerning the parameters and the cost derivative concerning the bias see equations 2.12 and 2.13.

$$\frac{\partial C}{\partial W} = \delta \cdot \frac{\partial z}{\partial W} \tag{2.12}$$

$$\frac{\partial C}{\partial b} = \delta \cdot \frac{\partial z}{\partial b} \tag{2.13}$$

$$\delta = \frac{\partial C}{\partial a} \cdot \frac{\partial a}{\partial z} \tag{2.14}$$

where δ is the error attributed to the neuron, it is computing for Equation 2.14. Then, C and A are the cost and activation functions, respectively. Also, W is the parameter, and Z is the weighted sum.

The backpropagation algorithm is computed in three essential stages; computing the error in the last layer, backward the error in the previous layer, and computing the derivative of the layer using the error see equations 2.15, 2.16, and 2.17, respectively. And so on, going through all the network layers until the final one, the input layer. Otherwise, Figure 2.6 shows a graphic representation of the backpropagation algorithm.

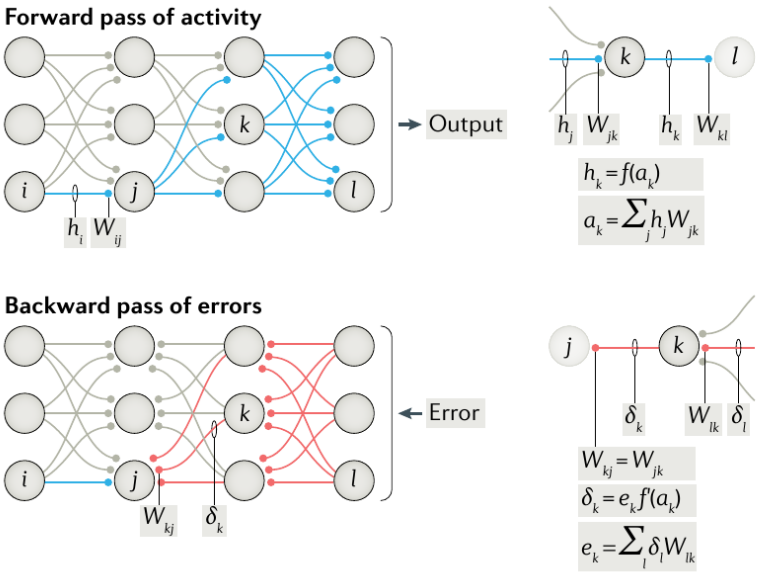


Figure 2.6: The backpropagation algorithm [2].

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \tag{2.15}$$

$$\delta^{l-1} = W^l \delta^l \cdot \frac{\partial a^{l-1}}{\partial z^{l-1}} \tag{2.16}$$

$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1} \quad \frac{\partial C}{\partial w^{l-1}} = \delta^{l-1} a^{l-2} \tag{2.17}$$

2.2.1 Loss Function

The loss function computes the predicted error of the artificial neural network across the training. This error reveals the difference between the current prediction and the truth. Moreover, this error helps the optimizers update the parameters during the training. Nowadays, there are several loss functions. However, in this subsection, we discuss the cross-entropy, mean square, and mean absolute error [45].

Cross-Entropy

Cross-Entropy (L_{CE}) is also called the log loss function. Further, it works well with the softmax activations (the output class probability) in the output layer for classification.

Moreover, the mathematical representation of L_{CE} is defined by Equation 2.18.

$$L_{CE}(p, y) = -\sum_{i} y_i \log(p_i) \quad \text{where,} \quad i \in [1, N] \tag{2.18}$$

where, y is the desired outputs, p is the outputs class probability, and N is the number of neurons in the output layer.

Mean Square Error

Mean Square Error (L_{MSE}) is also called Euclidean Loss Function. This loss function is so used for regression problems. Then, L_{MSE} is represented by:

$$L_{MSE}(p, y) = \frac{1}{2N} \sum_{i=1}^{N} (p_i - y_i)^2 \tag{2.19}$$

where, p is the predicted labels, y is the ground truth labels, and N is the number of neurons in the output layer.

Hinge

Hinge (L_H) is commonly used in the problem of maximum-margin-based classification (supported vector machines) and binary classification. Moreover, L_H is described by:

$$L_H(p, y) = \sum_{i=1}^{N} \max(0, m - (2y_i - 1)\, p_i) \quad \text{where, } m \text{ is the margin.} \tag{2.20}$$

And where, p is the predicted outputs, y is the desired outputs, and N is the number of neurons in the output layer

2.3 Optimizers

Optimizers are algorithms that change the ANN weights to reduce losses. There are several optimizers available now for ANN training. However, this work uses metaheuristic-based optimizers (i.e., PSO, PSO_bound, and QDPSO) and gradient-based optimizers (i.e., SGD, Adam, and L-BFGS).

2.3.1 Gradient-based Optimizers

The gradient descent was suggested in 1847 by Augustin-Louis Cauchy and proposed by Jacques Hadamard in 1907 [46]. The principle of gradient descent repeatedly moves in the opposite direction of the gradient function (or approximative gradient) at the current location since this is where the steepest drop occurs. Gradient descent has several versions. This subsection reviews Stochastic Gradient Descent, Adaptive Moment Estimation, and Limited-memory BFGS.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a version of Gradient Descent. It aims to do more frequent parameter updates for the model. In this, the model parameters are changed following the computation of the loss of each training session [47]. In short, the examples chosen randomly during each iteration determine the stochastic process. SGD is defined by

$$w_{t+1} = w_t - \overbrace{\alpha_t \nabla_w J\left(z_t, w_t\right)}^{\text{Backpropagation}} \tag{2.21}$$

where,

- w is the parameter.
- α is the learning rate.
- ∇ is the gradient.
- J is the objective function.

Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) uses first and second-order momentums [48]. The idea behind Adam is that we should slow down a little bit for a thorough search rather than rolling so quickly merely to be able to jump over the minimum. Like AdaDelta, Adam additionally stores an exponentially decaying average of previously squared gradients in addition to an exponentially decaying average of past gradients. The Adam method is defined by

$$
\begin{gathered}
w_{t+1} = w_t - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \\
\text{where,} \\
\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \\
\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\
m_t = (1 - \beta_1) \nabla_w J(z_t, w_t) + \beta_1 m_{t-1} \\
v_t = (1 - \beta_2) \nabla_w J(z_t, w_t)^2 + \beta_2 v_{t-1}
\end{gathered}
\tag{2.22}
$$

and where,

- w is the parameter.
- α is the learning rate.
- $\hat{m}$ and $\hat{v}$ are the first and second momentum, respectively.
- ϵ is the term preventing division by zero.
- β_1 and β_2 are the first and second momentum term, respectively.
- ∇ is the gradient.
- J is the objective function.

Limited-memory BFGS

Limited-memory BFGS (L-BFGS) is an optimization algorithm that finds a minimum of an objective function using the values and gradient of the objective function. Furthermore, it is the family of quasi-Newton methods. And it is an extension of the Broyden Fletcher-Goldfarb Shanno algorithm (BFGS). Moreover, the main idea of L-BFGS is to build the Hessian approximation using the curvature information from the most recent iterations [49]. In short, it discards the earlier curvature information that is not relevant to the actual behavior of the Hessian. The L-BFGS is defined by

$$
w_{k+1} = w_k - \overbrace{\alpha_k H_k \nabla f_k}^{\text{Backpropagation}} \tag{2.23}
$$

where,

- w is the parameter.
- α is the the step length.
- H is the inverse Hessian approximation.
- ∇ is the gradient.
- f is the function being minimized.

2.3.2 Metaheuristic-based Optimizers

For a few decades, there has been an increase in the study of metaheuristics, which are generic algorithmic frameworks frequently inspired by nature to address challenging optimization issues. Metaheuristics are proving to be effective alternatives to more traditional methods of addressing optimization problems, even those whose mathematical formulations contain uncertain, stochastic, and dynamic information. However, Metaheuristic algorithms can generate only a feasible solution, but it does not guarantee an optimal solution [50] [51]. There are many algorithms with this technic. This subsection reviews PSO, PSO_bound, and QDPSO.

Particle Swarm Optimization

In 1995, Kennedy and Eberhart proposed PSO [29], a heuristic optimization method to find global minimums or maximums. This algorithm is inspired by bird flocking behavior. Each particle in PSO indicates a possible solution. So, while one particle may be considered the ultimate answer, extracting the solution from the swarm is possible too. There are several variations of this algorithm. However, these always follow the following steps:

Creating Swarm: Create an initial swarm of n random particles. Each particle consists of 4 elements: a position that represents a specific combination of variable values, the value of the objective function at the place where the particle is located, a velocity that indicates how and where the particle moves, and a record of the best position in which the particle has been so far.

Evaluating Particles: Compute the value of the objective function in the position the particle occupies at that moment. Each particle also stores the position with its best

value to identify if a new position is better than the previous one. For that, it is necessary to know if it is a minimization or maximization problem.

Moving Particle: The algorithm optimizes the objective function, moving the particle in this part. Thus, it updated the position and speed of each particle. Hence, the algorithm uses the flight 2.24 and position 2.25 equations.

The Flight Equation is responsible for the speed of the particles in each iteration and is defined by

$$v_i^{t+1} = wv_i^t + c1r_p(P_{best,i}^t - x_i) + c2r_g(G_{best} - x_i) \tag{2.24}$$

where,

- v_i^{t+1}: The particle velocity i at the time $t+1$, i.e., the new speed.
- v_i^t: The particle velocity i at the time t, i.e., the current speed.
- w: The inertia coefficient reduces or increases the particle speed. It is usually between 0.8 and 1.2. If $w < 1$, the particle slows down as the iterations progress; this results in less exploration but a convergence towards the fastest optimum. If $w > 1$, the particle accelerates, allowing to explore more areas of the function space but making convergence difficult.
- $c1$: The cognitive coefficient. It usually bounded between 0 to 2.
- r_p: The uniform random numbers between 0 and 1.
- $P_{best,i}^t$: The best position in which the particle has been i until now.
- x_i: The particle position i at the time t.
- $c2$: The social coefficient. It usually bounded between 0 to 2.
- r_g: The uniform random numbers between 0 and 1.
- G_{best}: The best position of the entire swarm at the moment t, the global best value.

The flight function 2.24 has three crucial components: inertial, cognitive, and social component. The wv_i^t defines the inertial component responsible for keeping the particle

moving. The $c1r_p(P^t_{best,i} - x_i)$ is the cognitive component responsible for the particle moving towards the position where it has obtained better results individually. Furthermore, The $c2r_g(G_{best} - x_i)$ is the social component responsible for the particle tending to move toward the best position the swarm found. It can be interpreted as collective knowledge.

The Position Equation updates the position of the particle at each iteration and is defined by equation 2.25 and the graphic representation shows in Figure 2.7.

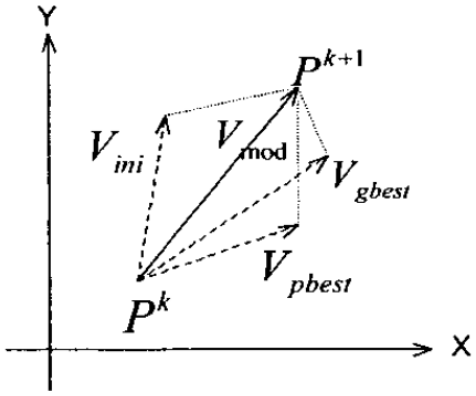


Figure 2.7: Concept of changing a particle of position in PSO [3].

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.25}$$

where,

- x_i^{t+1}: The particle position i at the time $t+1$, i.e., the new position.
- x_i^t: The particle position i at the time t, i.e., the current position.
- v_i^{t+1}: The particle's velocity in the time $t+1$.

Checking Criterion: If a stop criterion is not met, return to Evaluating Particles. Usually, the stop criterion is the error between prediction and target or the maximum iteration.

PSO has two implementation versions; local and global. However, the second version is the most used; see Algorithm 6. Here, it is shown from position and velocity initialize passing for flight function in each particle of the swarm until to update their position. Thus, each particle in PSO is a potential solution. On the other hand, the particles of

PSO tend to get stuck in local minima, causing that considerable research to be focused on solving it. In addition, the computational cost depends on the number of particles generated in the swarm and their dimensions.

Quantum-behaved Delta Particle Swarm Optimization

In 2004, Sun *et al.* proposed individual quantum-behaved particles for PSO [33]. Then, the wave function $\psi(\vec{x}, t)$ interprets the quantum-behaved particles, whose squared value indicates the likelihood of the existence of the particle at the position $\vec{x}$. Thus, a Delta potential well model is presented in the quantum world with the center on point p. Moreover, the probability density and distribution function are solved using the Monte Carlo technique to solve the Schrödinger equation. On the other hand, in terms of classical mechanics, a particle is depicted by its position vector $\vec{x}$ and velocity vector $\vec{v}$, which determine the trajectory of the particle. In Newtonian mechanics, the particle follows a predetermined path, but this is not the case in quantum mechanics. Therefore, the concept of trajectory has no relevance in the quantum realm; it is meaningless [52]. Because $\vec{v}$ and $\vec{x}$ of a particle can not be determined simultaneously according to the uncertainty principle, finally, the position of the *ith* particle can be computed by Equation 2.26

$$x = p \pm \frac{L}{2} \ln\left(\frac{1}{u}\right) \tag{2.26}$$

where,

$$p_i^t = u_i^t P_{best,i}^t + \left(1 - u_i^t\right) G_{best}^t + \Delta_i^t \tag{2.27}$$

$$\Delta_i^t = \frac{P_{best,a}^t - P_{best,c}^t}{2} \tag{2.28}$$

$$L_{i,j} = 2\beta \cdot \| \text{ mbest }_j - x_{i,j} \| \tag{2.29}$$

$$\beta = \frac{(1.0 - 0.5)(MAXITER - t)}{MAXITER - 0.5} \tag{2.30}$$

$$\text{mbest} = \frac{1}{N}\sum_{i=1}^{N} \text{P}_{best,i} \tag{2.31}$$

Equation 2.27 is the particle motion center, also called the attractor. Then, the characteristic length of the potential well delta, whose value is directly related to the convergence speed and searching ability, is defined by Equation 2.29. Finally, inside Equation 2.29, the Contraction-Expansion (CE) factor allows the convergence speed of algorithms. Moreover, the Mean Best Position is defined as the mean value of all particles in the swarm. Both are defined by the equations 2.30 y 2.31, respectively. Otherwise, u is a uniformly distributed random number between 0 and 1. $P^t_{best,i}$ is the best position in which the particle has been ith until now. G^t_{best} is the best position of the entire swarm at the moment t, the global best value. Finally, $x_{i,j}$ is the particle position ith at time t.

The QDPSO implementation has four essential parts, which are very similar to PSO implementation

- Creating the swarm
- Evaluating the particle
- Moving the particle
- Checking criteria

Nevertheless, the main differences are in the first and third stages of the algorithm. In the first stage, the QDPSO only initializes the position of each particle swarm and evaluates the objective function to find a global best of the swarm. In the third stage, the QDPSO uses Equation 2.26 to move the particles. Algorithm 7 shows the four stages of the QDPSO implementation. Besides, similar to PSO, the computational cost of the QDPSO depends on the number and dimension of the particle. However, the QDPSO has a faster convergence compared with PSO [33].

Chapter 3

State of the Art

This chapter discusses a few techniques for solving training neural networks using various optimizers. Optimizers are techniques that modify the weights of the neural networks to minimize losses. Nowadays, there are several optimizers for training neural networks. However, this project uses gradient-based optimizers and metaheuristic-based described in the previous chapter.

3.1 Gradient-based Optimizers

3.1.1 Stochastic Gradient Descent

In the recent decade, many researchers have improved gradient descent-based algorithms. For instance, Xinyu and Fei-Yue proposed an accelerating minibatch for SGD [53]. This method solves the slow convergence problem due to the considerable noise of the gradient approximations. Thus, the main idea is that not all training samples are equally crucial in gradient estimation. On each iteration, only parts of the training set are needed to roughly guide the correct direction of the parameters update. In addition, the authors give an experimental and mathematical explanation. Conversely, Algorithm 1 shows an implementation of the typical batch using SGD, and Algorithm 2 of the typical batch embedding using SGD and t-distributed stochastic neighbor (t-SNE) embedding.

Algorithm 1 build the subset $\mathcal{H}$ directly from the training set $\mathcal{X}$ and add it to the other samples in the subset $\mathcal{L}$. While Algorithm 2 computes 2-dimensional $\mathcal{X}'$ using t-SNE and the density $\mathcal{D}$ of each sample from $\mathcal{X}'$. Then, it builds the subset $\mathcal{H}$ by selecting the

Algorithm 1: Typical Batch SGD [53].

Require: Global learning rate η;
Require: Batch size m;
Require: Training set $\mathcal{X} = \{x_1, x_2, \ldots\ldots x_n\}$;
Require: Initial model parameter θ_0;
Require: Batch selection rate $\gamma \in (0, 0.5)$;
Build subset: demarcate subset $\mathcal{H}$ from training set $\mathcal{X}$ and put the other samples in subset $\mathcal{L}$;
while θ_k *not converged* **do**
    Update iteration: $k \leftarrow k + 1$;
    Select sub-batch $\mathcal{H}_k$ of size n_1 from subset $\mathcal{H}$ by SRS;
    Select sub-batch $\mathcal{L}_k$ of size n_2 from subset $\mathcal{L}$ by SRS;
    Get batch: $\mathcal{B} \leftarrow H_k + L_k$;
    Compute gradient: $\nabla J_{\mathcal{B}}(\theta_k) = \sum_{i \in \mathcal{B}} \nabla J_i(\theta)/m$;
    Apply update: $\theta_{k+1} = \theta_k - \eta * \nabla J_{\mathcal{B}}(\theta_k)$;
end

Algorithm 2: Typical Batch SGD: t-SNE Embedding [53].

Require: Global learning rate η;
Require: Batch size m;
Require: Training set $\mathcal{X} = \{x_1, x_2, \ldots\ldots x_n\}$;
Require: Initial model parameter θ_0;
Require: Batch selection rate $\gamma \in (0, 0.5)$;
Compute 2-dimensional data representation by t-SNE: $\mathcal{X}' = \{x'_1, x'_2, \ldots\ldots x'_n\}$;
Compute the density of each sample in $\mathcal{X}' : \mathcal{D} = \left\{d_{x'_1}, d_{x'_2}, \ldots\ldots d_{x'_n}\right\}$;
Build subset $\mathcal{H}$: select the top $n * \gamma$ samples from $\mathcal{D}$;
Build subset $\mathcal{L}$: select the remaining samples in $\mathcal{D}$;
while θ_k *not converged* **do**
    Update iteration: $k \leftarrow k + 1$;
    Select sub-batch $\mathcal{H}_k$ of size n_1 from subset $\mathcal{H}$ by SRS;
    Select sub-batch $\mathcal{L}_k$ of size n_2 from subset $\mathcal{L}$ by SRS;
    Get batch: $\mathcal{B} \leftarrow H_k + L_k$;
    Compute gradient: $\nabla J_{\mathcal{B}}(\theta_k) = \sum_{i \in \mathcal{B}} \nabla J_i(\theta)/m$;
    Apply update: $\theta_{k+1} = \theta_k - \eta * \nabla J_{\mathcal{B}}(\theta_k)$;
end

top samples from $\mathcal{D}$ and building the subset $\mathcal{L}$ by selecting the remaining samples in $\mathcal{D}$, which is the main difference between algorithm 1. Finally, in some way, this technic is implemented by Adam optimizer.

3.1.2 Adaptive Moment Estimation

Adam is a first-order gradient-based optimization technique for stochastic objective functions that Kingma and Lei Ba introduced in 2017 [48]. It is based on adaptive estimations of lower-order moments. Adam is simple to use, computationally effective, requires minimal memory, invariant to diagonal rescaling of the gradients, and works well for issues with many parameters or data. Thus, this method has a mathematical and experimental explanation. On the other way, the Adam implementation is described in Algorithm 3.

Algorithm 3: Adam optimizer [48].

Require: α: Stepsize;
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates ;
Require: $f(\theta)$: Stochastic objective function with parameters θ;
Require: θ_0 : Initial parameter vector;
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector);
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector);
$t \leftarrow 0$ (Initialize timestep);
while θ_t *not converged* **do**
    $t \leftarrow t + 1$;
    $g_t \leftarrow \nabla_\theta f_t\left(\theta_{t-1}\right)$ (Get gradients w.r.t. stochastic objective at timestep t);
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate);
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate);
    $\widehat{m}_t \leftarrow m_t / \left(1 - \beta_1^t\right)$ (Compute bias-corrected first moment estimate);
    $\widehat{v}_t \leftarrow v_t / \left(1 - \beta_2^t\right)$ (Compute bias-corrected second raw moment estimate);
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / \left(\sqrt{\widehat{v}_t} + \epsilon\right)$ (Update parameters);
end
Return θ_t (Resulting parameters);

The authors made the logistic regression, multi-layer neural network, and convolutional neural network experiments using Adam as an optimizer. Then, They compare it with other optimizers such as AdaGrad, RMSProp, SGDNesterov, and AdaDelta. Also, Kingma and Lei Ba propose proposes an extension for Adam based on the infinity norm, called AdaMAX [48]. Then, the implementation of the Adamax is described in Algorithm 4, where the exponentially weighted infinity norm updates the second momentum.

Algorithm 4: AdaMAX optimizer [48].

Require: α: Stepsize;
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates ;
Require: $f(\theta)$: Stochastic objective function with parameters θ;
Require: θ_0 : Initial parameter vector;
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector);
$u_0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm);
$t \leftarrow 0$ (Initialize timestep);
while θ_t *not converged* **do**
    $t \leftarrow t + 1$;
    $g_t \leftarrow \nabla_\theta f_t\left(\theta_{t-1}\right)$ (Get gradients w.r.t. stochastic objective at timestep t);
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate);
    $u_t \leftarrow \max\left(\beta_2 \cdot u_{t-1}, g_t \mid\right)$ (Update the exponentially weighted infinity norm);
    $\theta_t \leftarrow \theta_{t-1} - \left(\alpha / \left(1 - \beta_1^t\right)\right) \cdot m_t / u_t$ (Update parameters);
end
Return θ_t (Resulting parameters);

3.1.3 Limite-memory BFGS

In 2019 Chang *et al.* proposed an accelerated linearly convergent stochastic L-BFGS (AccsL-BFGS) Algorithm for training neural networks [54]. This technic has a mathematical and experimental explanation. Moreover, It is based on the conventional L-BFGS update rule and is described in Algorithm 5. On the other hand, the AccsL-BFGS algorithm uses six datasets and six optimizers to compare and prove that AccsL-BFGS converges faster in some cases.

Finally, several papers briefly overview the gradient-based optimizers for training a neural network in recent years. For example, Alzubaidi *et al.* reviewed deep learning, focusing on concepts, CNN architectures, challenges, and applications [45]. In the same way, Tian *et al.* overviewed deep learning for image denoising [55]. Moreover, Balaha *et al.* use deep learning and genetic algorithm for hybrid COVID-19 segmentation and recognition framework [56]. On the other hand, Borhani uses the L-BFGS algorithm for document categorization [57]. Also, Jacob and Roummel propose an improving L-BFGS initialization for trust-region methods [58]. Then, Kennedy *et al.* proposed a parallelization and distribution stochastic gradient descent implementation using commodity clusters [59]. And so on.

Algorithm 5: Accelerated Stochastic L-BFGS optimizer [54].

Require: mini-batch size b, b_H, step size η, inner iterations m, memory length M, constant K, sampling distribution P, initialize $w_0, r = 0, \bar{y}_0 = w_0$.;
for *t = 0, 1, 2 ... T-1* **do**
  Compute $\mu_t = \nabla f(w_t)$;
  Let $x_0 = w_t, \quad y_0 = w_t$;
  for *k = 0, 1, 2 ... m-1* **do**
    Select b indices $I_k \subset \{1, 2, \dots n\}$ with distribution P;
    Compute $v_k = \frac{1}{b} \sum_{i \in I_k} \frac{1}{N p_i} (\nabla f_i(y_k) - \nabla f_i(w_t) + \mu_t)$;
    Compute $H_r v_k$ with $\{s_i, u_i\}_{i=r-M+1}^{r}$ (All the r pairs of $\{s_i, u_i\}$ are used if $r < M$);
    Update $x_{k+1} = y_k - \eta H_r v_k$;
    Update $y_{k+1} = x_{k+1} + \frac{1-\sqrt{\gamma\bar{\mu}\eta}}{1+\sqrt{\gamma\bar{\mu}\eta}} (x_{k+1} - x_k) + \frac{\eta}{1+\sqrt{\gamma\bar{\mu}\eta}} (H_r - \gamma I) v_k$;
    if $mt + k > 0$ *and* $\mathrm{mod}(mt + k, K) \equiv 0$ **then**
      Let $r = r + 1$;
      Compute $\bar{y}_r = \frac{1}{K} \left(\sum_{i=\max\{0, k-K+1\}}^{k} y_i \ + \sum_{i=\min\{m+1, k-K+m+2\}}^{m} \tilde{y}_i \right)$;
      Compute $s_r = \bar{y}_r - \bar{y}_{r-1}$;
      Select b_H indices uniformly and compute $u_r = \nabla^2 f_{b_H}(\bar{y}_r) s_r$;
    end
  end
end
Ensure: w_t;

3.2 Metaheuristic-based Optimizers

3.2.1 Particle Swarm Optimization

In recent years, PSO has gained momentum in neural network training, drawing the attention of many researchers in this field. An example of this is the PSO used in recurrent neural networks, which was proposed by Aziz *et al.* in 2021 [4]. The authors use a benchmark dataset for classification, such as lung cancer, breast cancer, Pima Indian diabetes, yeast, etc. The proposed method is described in Figure 3.1, where the main difference is the structure optimizer for training neural networks. Furthermore, the traditional optimizer of PSO is described in Algorithm 6.

The main contribution is the equation to compute the dimensions for the PSO algorithm (see Equation 3.1) and the model architecture. On the other hand, Aziz *et al.* based its architecture on the primary Elman Recurrent Neural Network (ERNN) proposed by Cheng and Shen in 2010 [60]. Thus, the ERNN architecture is described in Figure 3.2, which shows

Algorithm 6: Global Best Particle Swarm Optimization.

```
initialization;
for each particle i=1, ..., s do
    x_i ← Uniform randomly within permissible range;
    p_i ← x_i;
    if f(p_i) < f(g) then
        g ← p_i;
    end
    v_i ← Uniform randomly within permissible range;
end
while termination criterion is not met do
    for each particle i=1, ..., s do
        for each dimension d = 1, ..., n do
            r_p, r_g ← Uniform randomly within (0, 1);
            v_{i,d} ← Flight Equation 2.24;
        end
        x_i ← Position Equation 2.25;
        if f(x_i) < f(p_i) then
            p_i ← x_i;
            if f(p_i) < f(g) then
                g ← p_i;
            end
        end
    end
end
```

the three layers: the input layer with the context unit, the hidden layer with the hidden unit, and the output layer.

$$
\begin{aligned}
D = \sum_{i,j,k=1}^{n} \Big(\Big(X_{i1} \times H_{j1} \Big) + \Big(H_{j1} \times O_{k1} \Big) + H_{j1} + O_{k1} \Big) + \\
\Big(\Big(X_{i2} \times H_{j2} \Big) + \Big(H_{j2} \times O_{k2} \Big) + H_{j2} + O_{k2} \Big) \\
+ \cdots + \\
\Big(\Big(X_{in} \times H_{jn} \Big) + \Big(H_{jn} \times O_{kn} \Big) + H_{jn} + O_{kn} \Big)
\end{aligned}
\tag{3.1}
$$

3.2.2 Quantum-behaved Delta Particle Swarm Optimization

In 2021, Liu *et al.* proposed the coal and gas outbursts prediction with hybrid feature extraction using a QPSO as an optimizer [5]. In addition, the authors use a multi-layer

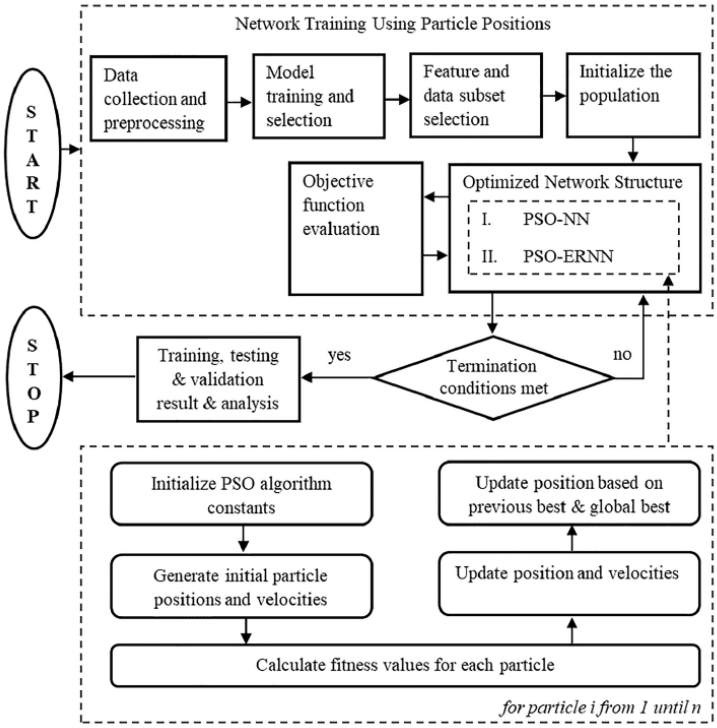


Figure 3.1: Generalized schematic of PSO-NN and PSO-ERNN [4].

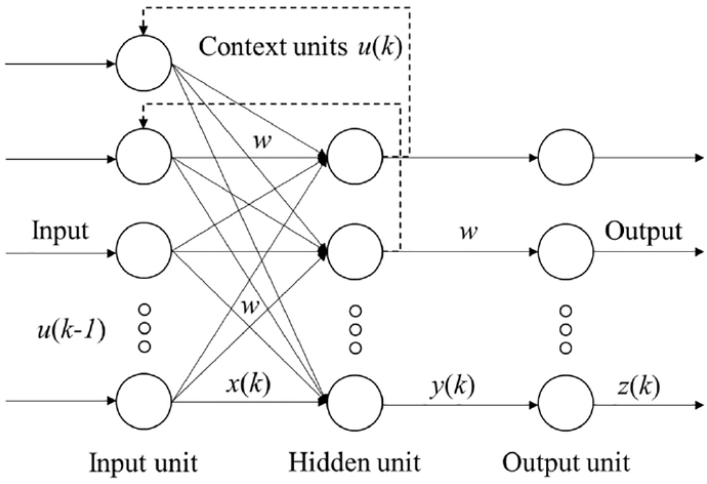


Figure 3.2: The ERNN architecture [4].

neural network called DELM, which is highly efficient in learning time and has good generalization ability. Thus, the flowchart is described in Figure 3.3, which shows that the QPSO optimizes the parameter for classification.

Algorithm 7 is shown the QPSO implementation using a Delta potential well model. Conversely, the contribution of the authors is the feature preprocessing of coal and gas outbursts using the wavelet transform algorithm and DELM initialization for the QPSO

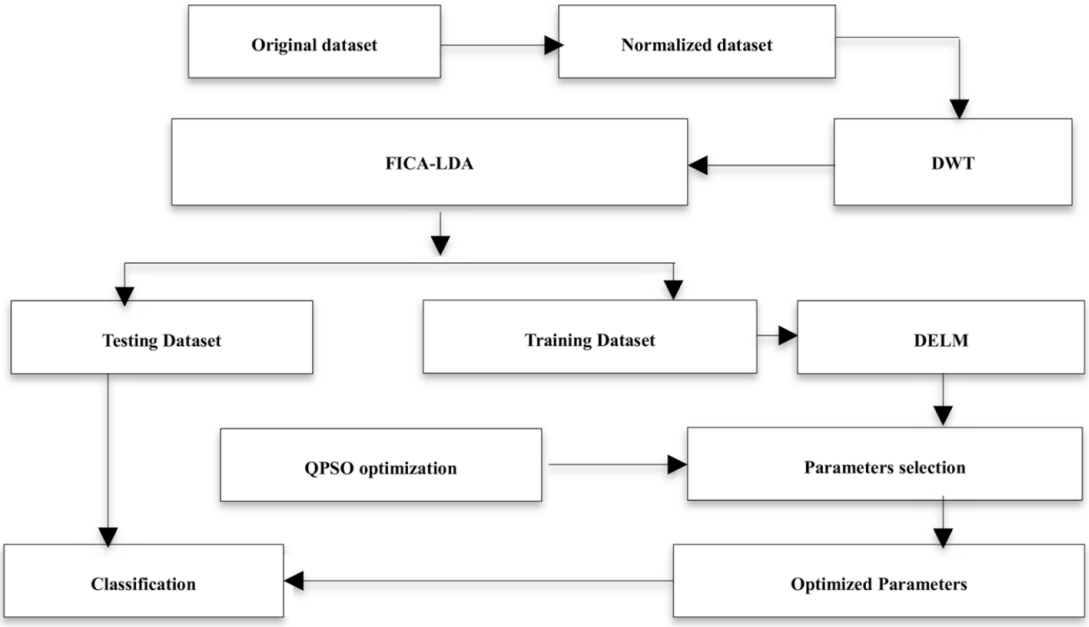


Figure 3.3: Flowchart of prediction model using QPSO-DELM as an optimizer [5].

optimizer.

Some papers have recently used PSO and QPSO for training neural networks. For example, Ly *et al.* used PSO to predict foamed concrete compressive strength [61]. Further, Green II *et al.* use Central Fores Optimization and PSO to classify the iris dataset and compare them [62]. Moreover, Chanda and Biswas proposed PSO to identify plant diseases [63]. In 2019, Fernandes Junior and Yen proposed a deep neural network for image classification using PSO and CNN [64]. Then, in 2021, Abdolrasol *et al.* used PSO for microgrid optimal energy scheduling [65].

On the other hand, Ma *et al.* proposed network anomaly detection as QPSO as an optimizer for training RBF neural networks [66]. Moreover, Fu-Guang and Xian-Xin use an improved QDPSO for training the hypersphere [67]. Besides, Li *et al.* proposed a binary encoding for image classification using the QDPSO algorithm [68]. In 2020, Feng *et al.* used QDPSO for monthly runoff prediction [69]. Then, in 2021, Concepcion II *et al.* presented QDPSO and recurrent neural network for diseased surface assessment of maize Cercospora leaf [70].

Algorithm 7: Quantum Delta Potential well based PSO Algorithm [33].

initialization;
for *each particle i=1, ..., s* **do**
    $x_i \leftarrow$ Uniform randomly within permissible range;
    $p_i \leftarrow x_i$;
    if $f(p_i) < f(g)$ **then**
        $g \leftarrow p_i$;
    end
end
while *termination criterion is not met* **do**
    for *each particle i=1, ..., s* **do**
        for *each dimension d = 1, ..., n* **do**
            $r_p, r_g, u \leftarrow$ Uniform randomly within (0, 1);
            $p \leftarrow$ Attractor equation 2.27;
            $L \leftarrow$ Characteristic Length equation 2.29;
        end
        if $rand(0, 1) > 0.5$ **then**
            $x_{id} \leftarrow$ Delta Potential well Equation with symbol minus 2.26;
        end
        else
            $x_{id} \leftarrow$ Delta Potential well Equation with symbol plus 2.26;
        end
        if $f(x_i) < f(p_i)$ **then**
            $p_i \leftarrow x_i$;
            if $f(p_i) < f(g)$ **then**
                $g \leftarrow p_i$;
            end
        end
    end
end

Chapter 4

Methodology

This chapter describes the processes used to achieve the goals and the rationale for their usage. As a result, this chapter begins by explaining the problem resolution of the chronological steps. Then, the model is shown with its respective configurations. Finally, we describe the benchmark dataset used and the method for the result analysis.

4.1 Phases of Problem Solving

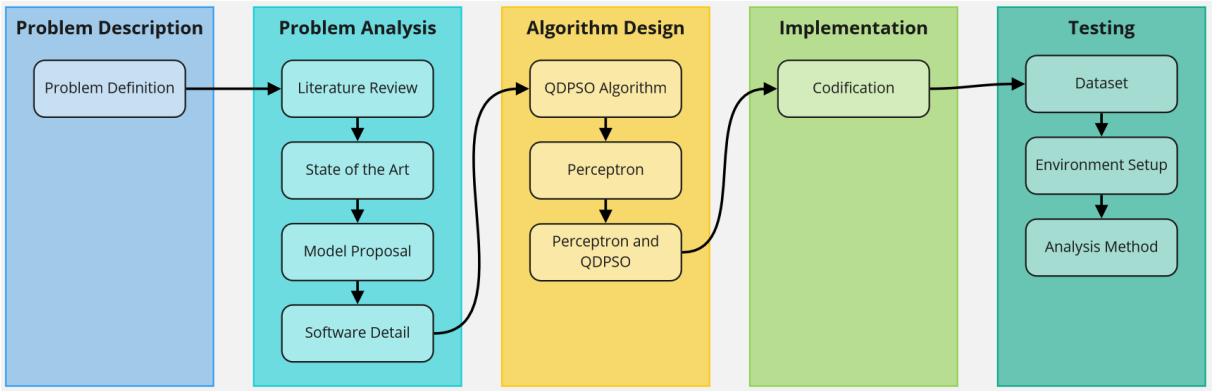


Figure 4.1: Flowchart of the problem-solving.

4.1.1 Description of the Problem

This phase is the foundation for project development since it covers the core issue, potential challenges, and solutions. In this way, in the early steps, we establish the fundamental meaning of artificial neural networks and their general characteristics. Then, we identified several potential issues and selected which ones to fix. Later, we developed a small model

approach to address this issue. Finally, we created a methodology for designing and developing the model proposal, see Figure 4.1. The steps outlined in this phase are consolidated in Chapters 1 and 4.

4.1.2 Analysis of the Problem

This phase comprises background and foundational knowledge to comprehend the current project better. Thus, in Chapter 2, we develop a solid understanding of the training of artificial neural network and their properties. In this sense, we focused on metaheuristic-based optimizers.

4.1.3 Algorithm Design

This phase consists of the algorithmic design of the neural network training using gradient-based and metaheuristic optimizers. These optimizers are explained in Chapters 2 and 3. Moreover, we designed the artificial neural network model for the classification problem because in the literature review, in more cases, used for a regression problem. Finally, the same model design is used for the image classification process.

4.1.4 Implementation

This phase involves programming the algorithms based on the preliminary design. Thus, the implementation consists of a prototype using python 3.9.7 with scikit-learn 1.0.2, qpso 0.0.1, and pyswarm 1.3.0 frameworks. On the other hand, we use python because it is a language that helps us to spot mistakes fast. Finally, the first runs of the algorithm did make it in the cluster of Cedia because the first stages consumed many resources. And the code is available in the GitHub repository `https://github.com/stalyn21/nn_qdpso.git`.

4.1.5 Testing

The performance of the suggested algorithm is measured and analyzed in this phase. For this purpose, we evaluate the iteration numbers, the error, recall, precision, and accuracy. Moreover, we analyze the algorithm behavior, increasing classes in the circle dataset and

reducing dimensions in the iris and breast cancer datasets. On the other hand, we compare the metaheuristic-based and gradient-based optimizers with the same environment setup.

4.2 Model Proposal

This section describes the proposed model to train an artificial neural network using QDPSO as an optimizer. In this sense, we start to build a perceptron for classification and define the fitness function. Then, we minimize the objective function using the QDPSO algorithm and get the parameters of ANN.

4.2.1 Multi-Layer Perceptron

The most straightforward unit of a neural network is a single-layer perceptron. Thus, the input values, weights and a bias, a weighted sum, and an activation function comprise a muti-layer perceptron. Consequently, a multi-layer perceptron has three layers: the input, hidden, and output, as described in Section 2.1.1 and Figure 2.3.

4.2.2 Fitness Function

The objective or fitness function is defined by feed-forward, as seen in Equation 4.1. Therefore, the objective function comprises an input, weights, bias, weighted sum, activation function, and output value.

$$Fitness\ Function = L(a(Z)) \tag{4.1}$$

where, $L()$ is the cost function, $a()$ is the activation function, and Z is the weighted sum.

4.2.3 Optimizer

The QDPSO optimizer is described in more detail in Sections 2.3.2 and 3.2.2, and its pseudocode is in Algorithm 7. Then, the optimizer minimizes the loss function together feed-forward. Furthermore, the illustrated implementation of the operation of the QDPSO optimizer is shown in Figure 4.2.

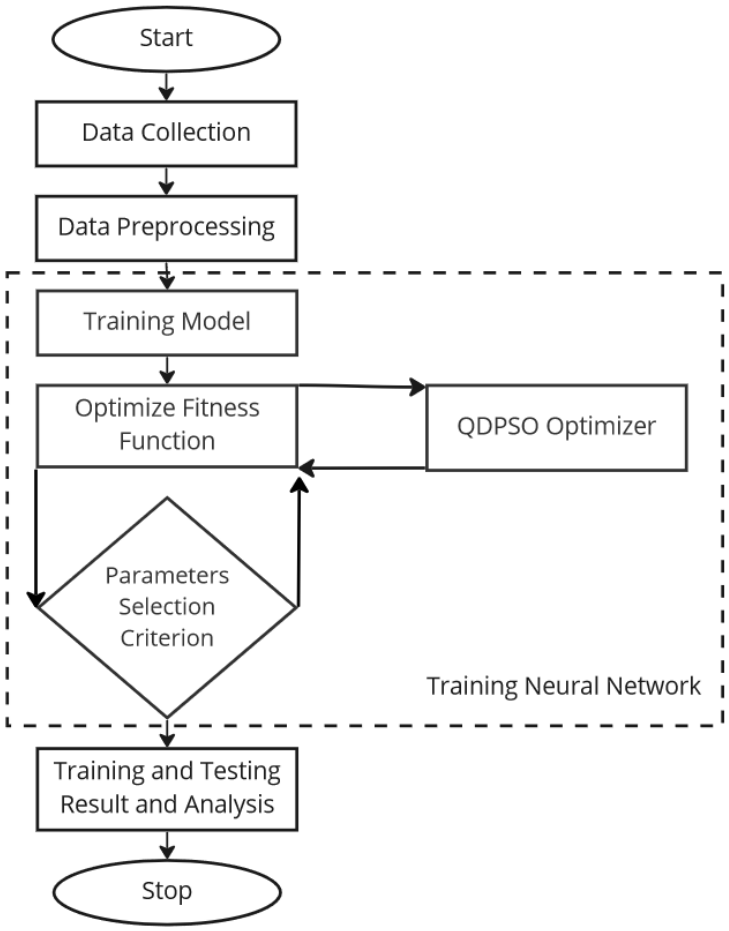


Figure 4.2: Flowchart of the artificial neural network training using the QDPSO optimizer.

4.3 Experimental Setup

In this section, we define the benchmark dataset, the environment setup for optimizers, and a brief justification of its use.

4.3.1 Benchmark Datasets

The performances of the QDPSO optimizer are assessed by experimenting with four benchmark datasets from the scikit-learn framework. Thus, the datasets selected are circle, iris, wine, and breast cancer, which are used for classification testing. Then, a brief description of these datasets is detailed in Table 4.1. On the other hand, each dataset is divided into 80% training and 20% testing. Then, we normalize the dataset in the range of 0 to 1. Furthermore, 10 runs on each dataset are performed to obtain reliable findings and the average accuracy results. Table 4.2 depicts the distribution of the datasets.

No	Dataset	Short Name	Samples	Features	Classes	Features/Classes
1	circle	circle	500	2	2	250
2	iris	iris	150	4	3	50
3	wine	wine	178	13	3	59, 71, 48
4	breast cancer	breast	569	30	2	212,357

Table 4.1: The description of the berchmark dataset.

No	Short Name	Samples	Training	Testing
1	circle	500	400	100
2	iris	150	120	30
3	wine	178	142	36
4	breast	569	455	114

Table 4.2: The distribution of training and testing dataset.

4.3.2 Image Classification Dataset

In the image classification problem, there are several datasets. Nevertheless, we check the performance of the QDPSO optimizer using the Multi-class Weather (MCW) Dataset [71]. Moreover, the MCW dataset has 1125 images divided into four categories based on sunrise, cloudy, rainy, and sunshine. Besides, we split the dataset into 30% for testing and 70% for training, and Table 4.3 describes the MCW dataset in detail. On the other hand, we use 3 technics to extract the global features of each image. Thus, we utilize the Color Histogram [72], Hu Moments [73], and Haralick Texture [74] to extract the quantified color, shape, and texture. Moreover, we reduce six times the global features with a reducing algorithm. The image classification flowchart is described in Figure 4.3.

Short Name	MCW
Samples	1125
Training	787
Testing	338
Features	150 x 150
Features Extracted	84
Features Reduced	14
Classes	4
Features/Classes	300, 215, 253, 357

Table 4.3: The description of the Multi-class Weather Dataset.

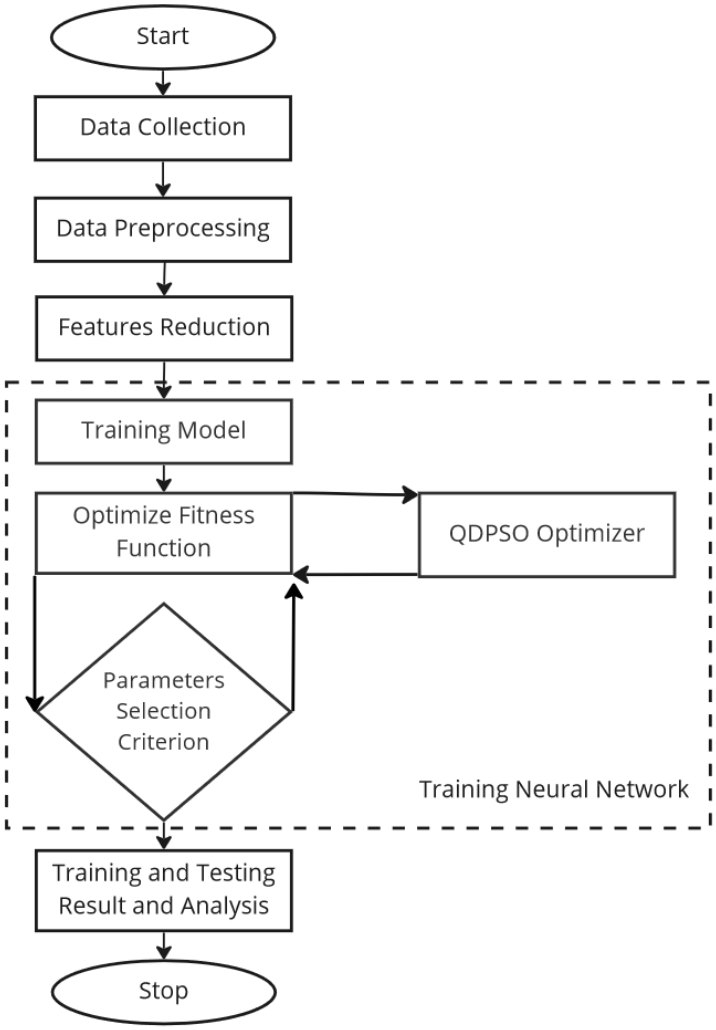


Figure 4.3: Flowchart of the artificial neural network training for image classification using the QDPSO optimizer.

4.3.3 Environment Setup

The same global environment configuration is used to compare the metaheuristic-based and gradient-based optimizers. Then, the numbers of the hidden nodes are calculated, which is three times the number of the input node (the value mentioned is obtained after many experiments). Besides, the hyperbolic tangent and softmax functions are used as activation functions in the hidden and output layer, respectively. Furthermore, the loss function is the cross-entropy with L_2 penalization, and we execute with 100 and 1000 iterations for the benchmark dataset and 1000 for the MCW dataset. L_2 is equal to 0.0005 over the number of samples. Thus, Table 4.4 describes the environment setup for each dataset in more detail. On the other hand, we use the F1 score, recall, precision, accuracy score, and

mean squared (MSE) error from scikit learn as metrics for multiclass classification.

Hyperparameters		**circle**	**iris**	**wine**	**breast**	**MCW**	**MCW***
Sample		500	150	178	569	1125	1125
Node	Input	2	4	13	30	84	14
	Hidden	3*	3*	3*	3*	3*	3*
	Ouput	2	3	3	2	4	4
Parameters		32	99	666	2972	22432	802
a()	Hidden	tanh	tanh	tanh	tanh	tanh	tanh
	Ouput	softmax	softmax	softmax	softmax	softmax	softmax
L()		L_{CE}	L_{CE}	L_{CE}	L_{CE}	L_{CE}	L_{CE}
Regularization		L_2	L_2	L_2	L_2	L_2	L_2
Max_iter		100,1000	100,1000	100,1000	100,1000	1000	1000
N_training		10	10	10	10	10	10

Table 4.4: The environment setup for QDPSO, PSO, PSO_bound, Adam, L-BFGS, and SGD optimizers.

Metaheuristic-Environment Setup

We use 0.5, 0.3, and 0.9 for cognitive, social, and inertial coefficients in the configuration of the PSO and PSO_bound environments. Furthermore, by PSO_bound, we add up and lower limits to -1 and 1. Conversely, for QDPSO, the Contraction-Expansion factor equals 1.13, and the upper and lower bounds are between -1 and 1.

Gradient-Environment Setup

We use the default parameters defined in the MLPClassifier module of scikit-learn for the gradient-based optimizers environments setup.

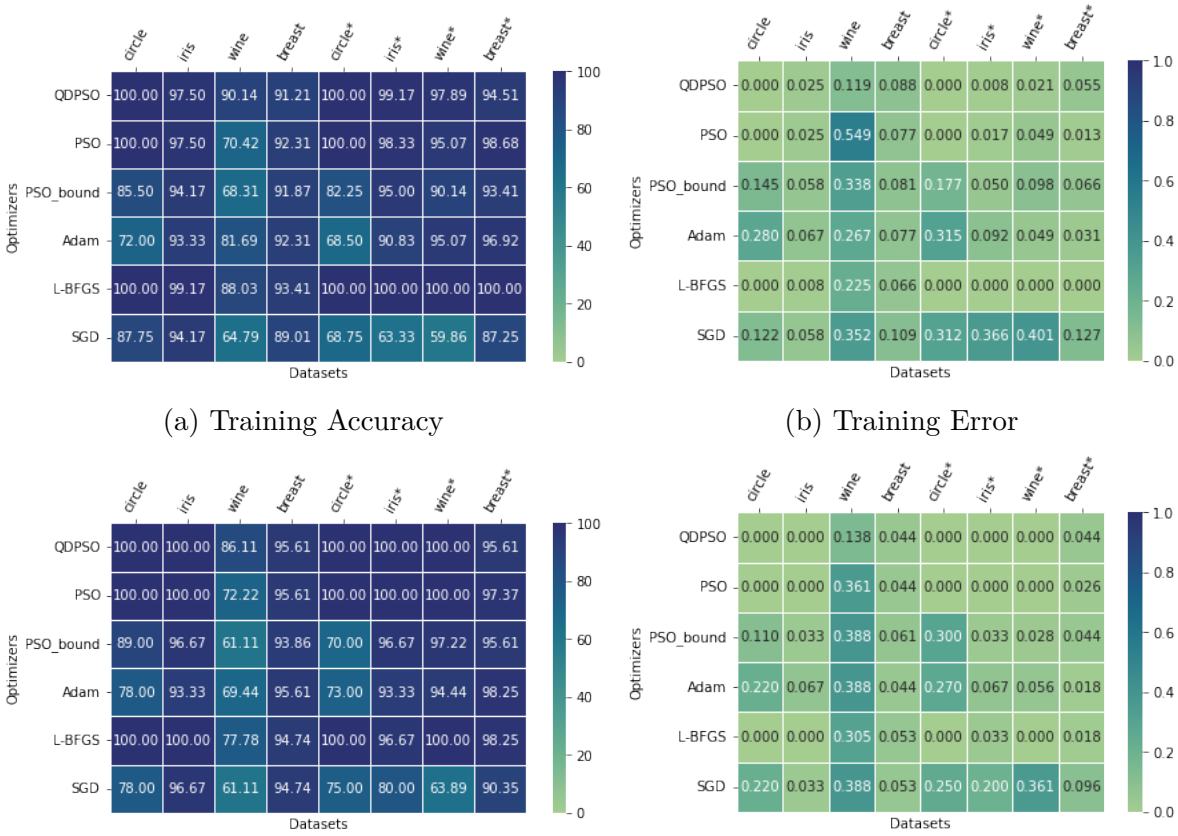
Chapter 5

Results and Discussion

This chapter describes the essential findings from the QDPSO optimizer in artificial neural networks. Firstly, we figure out the performances of the optimizer using the benchmark datasets. Then, we assess the dimension reduction using Multi-dimensional Scaling (MDS), t-distributed Stochastic Neighbor Embedding (TSNE), Isomap, and Principal Component Analysis (PCA) algorithms. Furthermore, we check the behavior of the QDPSO optimizer using the circle dataset with a different number of classes and the number of samples. Besides, we evaluate the results in image classification. Finally, we write a discussion related to the obtained results.

5.1 Performance of Benchmark Dataset

Figures 5.1 and 5.2 show the performance of the benchmark dataset using different optimizers. The classifiers can be divided into 2 groups; the optimizers based on metaheuristic (rows 1-3) and gradient (rows 4-6). The datasets are split into not normalized (columns 1-4) and normalized (columns 5-8). Furthermore, Figures a and b depict the accuracy and error results during the training phase, while c and d are of the testing stage. On the other hand, we describe the results of the training and testing stages for optimizers based on metaheuristics and gradients. In addition, we depict the matrix confusion and the loss graphic of the circle, iris, wine, and breast datasets. Finally, we analyze the results of the benchmark datasets.



(a) Training Accuracy

(b) Training Error

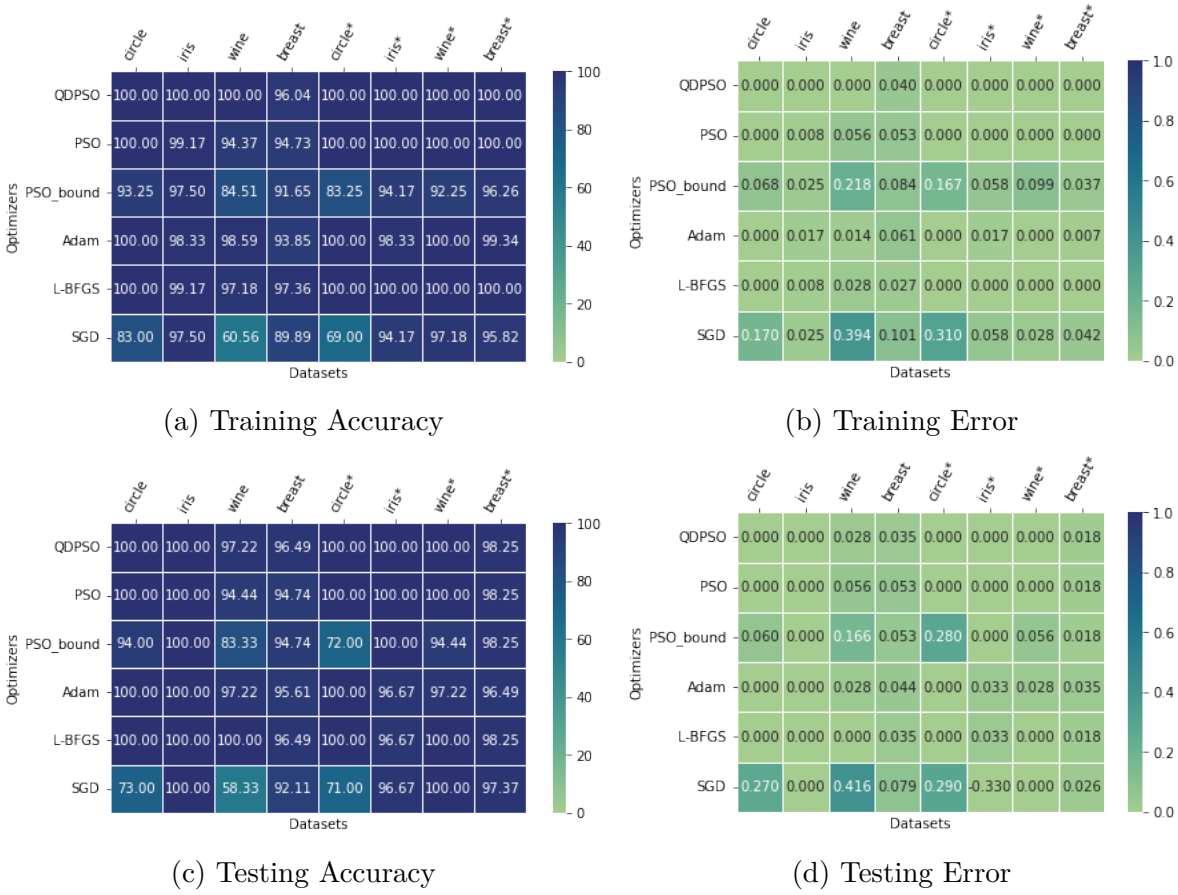
(c) Testing Accuracy

(d) Testing Error

Figure 5.1: Heatmaps depicting the performance of each optimizer with benchmark datasets using 100 iterations. Where (a) shows the accuracy of training in percent, (b) illustrates the mean squared error during the training, (c) sees the testing accuracy in percent, and (d) represents the mean squared error during the testing.

5.1.1 The result of the metaheuristic-based optimizers

In the training phase of the metaheuristic-based optimizers with 100 iterations and without normalization, QDPSO (100 and 97.50) and PSO (100 and 97.50) are better than PSO_bound (85.50 and 94.17) in the circle and iris dataset, see Figures 5.1a and 5.1b. Furthermore, QDPSO is better than PSO and PSO_bound in the wine dataset; 90.14, 70.42, and 68.31, respectively. PSO is better than QDPSO and PSO_bound in the breast dataset; correspondingly, 92.31, 91.21, and 91.87. However, with normalization, QDPSO (99.17 and 97.89) is better than PSO (98.33 and 95.07) and PSO_bound (95 and 90.14) in the iris and wine dataset. QDPSO and PSO are better than PSO_bound in the circle dataset; 100, 100, and 82.25, appropriately. PSO is better than QDPSO and PSO_bound in the breast dataset; 98.68, 94.51, and 93.41, respectively. Besides, with 1000 iterations and without



(a) Training Accuracy

(b) Training Error

(c) Testing Accuracy

(d) Testing Error

Figure 5.2: Heatmaps depicting the performance of each optimizer with benchmark datasets using 1000 iterations. Where (a) shows the accuracy of training in percent, (b) illustrates the mean squared error during the training, (c) sees the testing accuracy in percent, and (d) represents the mean squared error during the testing.

normalization, QDPSO (100, 100, and 96.04) is better than PSO (99.17, 94.37, and 94.73) and PSO_bound (97.50, 84.51, and 91.65) in the iris, wine, and breast datasets, see Figures 5.2a and 5.2b. Moreover, QDPSO and PSO are better than PSO_bound in the circle dataset; correspondingly, 100, 100, and 93.25. Nevertheless, with normalization, QDPSO (100, 100, 100, and 100) and PSO (100, 100, 100, and 100) are better than PSO_bound (83.25, 94.17, and 92.25, 96.26) in the circle, iris, wine, and breast datasets.

In the test phase of the metaheuristic-based optimizers without normalization and 100 iterations, QDPSO (100, 100, and 95.61) and PSO (100, 100, and 95.61) are better than PSO_bound (89, 96.67, and 93.86) in the circle, iris, and breast datasets. Moreover, QDPSO is better than PSO and PSO_bound in the wine dataset; 86.11, 72.22, and 61.11, appropriately, see Figures 5.1c and 5.1d. Nonetheless, QDPSO (100, 100, and 100) and

PSO (100, 100, and 100) are better than PSO_bound (70, 96.67, and 97.22) in the circle, iris, and wine datasets with normalization. Besides, PSO is better than QDPSO and PSO_bound in the breast dataset; 97.37, 95.61, and 95.61, respectively. On the other hand, without normalization and 1000 iterations, QDPSO (97.22 and 96.49) is better than PSO (94.44 and 94.74) and PSO_bound (83.33 and 94.74) in the wine and breast datasets, see Figures 5.2c and 5.2d. Likewise, QDPSO and PSO are better than PSO_bound in the circle dataset; correspondingly, 100, 100, and 94. However, with normalization, QDPSO (100 and 100) and PSO (100 and 100) are better than PSO_bound (72 and 94.44) in the circle and wine dataset. Moreover, the metaheuristic-based optimizers have the same results (100 and 98.25) in the iris and breast datasets.

5.1.2 The result of the gradient-based optimizers

In the training phase of the gradient-based optimizers with 100 iterations and without normalization, L-BFGS (100, 99.17, 88.03, and 93.41) is better than Adam (72, 93.33, 81.69, and 92.31) and SGD (87.75, 94.17, 64.79, and 89.01) in the circle, iris, wine, and breast datasets, see Figures 5.1a and 5.1b. Moreover, the same happened with the normalized dataset. On the other hand, with 1000 iterations and without normalization, L-BFGS (99.17 and 97.36) is better than Adam (98.33 and 93.85) and SGD (97.50 and 89.89) in the iris and breast datasets, see Figures 5.2a and 5.2b. Furthermore, Adam is better than L-BFGS and SGD in the wine dataset; 98.59, 97.18, and 60.56, appropriately. Further, L-BFGS and Adam are better than SGD in the circle dataset; 100, 100, and 83, respectively. However, with normalization, L-BFGS (100 and 100) is better than Adam (98.33 and 99.34) and SGD (94.17 and 95.82) in the iris and breast datasets. Moreover, L-BFGS (100 and 100) and Adam (100 and 100) are better than SGD (69 and 97.18) in the circle and wine datasets.

In the test phase of the gradient-based optimizers without normalization and 100 iterations, L-BFGS (100, 100, and 77.78) is better than Adam (78, 93.33, and 69.44) and SGD (78, 96.67, and 61.11) in the circle, iris, and wine datasets see Figures 5.1c and 5.1d. Furthermore, Adam is better than L-BFGS and SGD in the breast dataset; correspondingly, 95.61, 94.74, and 94.74. However, with normalization, L-BFGS (100, 96.67, and 100) is better than Adam (73, 93.33, and 94.44) and SGD (75, 80, and 63.89) in the circle,

iris, and wine datasets. Besides, Adam and L-BFGS are better than SGD in the breast dataset; 98.25, 98.25, and 90.35, appropriately. On the other hand, without normalization and 1000 iterations, L-BFGS (100 and 96.49) is better than Adam (97.22 and 95.61) and SGD (58.33 and 92.11) in the wine and breast datasets, see Figures 5.2c and 5.2d. In addition, Adam and L-BFGS are better than SGD in the circle dataset; 100, 100, and 73, respectively. In the iris dataset, the gradient based-optimizers have the same results, 100. Nonetheless, with normalization, Adam and L-BFGS are better than SGD in the circle dataset; correspondingly, 100, 100, and 71. Moreover, L-BFGS (100 and 98.25) and SGD (100 and 97.37) are better than Adam (97.22 and 96.49) in the wine and breast datasets. In addition, the same results, 96.67, have the gradient-based optimizers in the iris dataset.

5.1.3 The result of the metaheuristic-based and gradient-based optimizers

In the circle dataset and training stage, without normalization and 100 iterations, QDPSO, PSO, and L-BFGS are better than PSO_bound, Adam, and SGD; correspondingly, 100, 100, 100, 85.50, 72, and 87.75, see Figures 5.1a and 5.1b. Furthermore, in the iris and breast dataset, L-BFGS (99.17 and 93.41) is better than QDPSO (97.5 and 91.21), PSO (97.5 and 92.31), PSO_bound (95.17 and 91.87), Adam (93.33 and 92.31), and SGD (94.17 and 89.01). Besides, QDPSO is better than PSO, PSO_bound, Adam, L-BFGS, and SGD in the wine dataset; 90.14, 70.42, 68.31, 81.69, 88.03, and 64.79, appropriately. However, with normalization, QDPSO, PSO, and L-BFGS are better than PSO_bound, Adam, and SGD in the circle dataset; 100, 100, 100, 82.25, 68.50, and 68.75, respectively. In addition, L-BFGS (100, 100, and 100) is better than QDPSO (99.17, 97.89, and 64.51), PSO (98.33, 95.07, and 98.68), PSO_bound (95, 90.14, and 93.41), Adam (90.83, 95.07, and 96.92), and SGD (63.33, 59.86, and 87.25) in the iris, wine, and breast datasets. Conversely, without normalization and 1000 iterations, QDPSO, PSO, Adam, and L-BFGS are better than PSO_bound and SGD in the circle dataset; correspondingly, 100, 100, 100, 100, 93.25, and 83, see Figures 5.2a and 5.2b. L-BFGS is better than QDPSO, PSO, PSO_bound, Adam, and SGD in the breast dataset; 97.36, 96.04, 94.73, 91.65, 93.85, and 89.89, appropriately. Besides, QDPSO (100 and 100) is better than LBFGS (99.17 and 97.18), PSO (99.17 and 94.37), PSO_bound (97.50 and 84.51), Adam (98.33 and 98.59), and SGD (97.50 and

60.56) in the iris and wine dataset. Nonetheless, with normalization, QDPSO (100 and 100), PSO (100 and 100), Adam (100 and 100), and L-BFGS (100 and 100) are better than PSO_bound (83.25 and 92.25) and SGD (69 and 97.18) in the circle and wine dataset. Moreover, QDPSO (100 and 100), PSO (100 and 100), and L-BFGS (100 and 100) are better than Adam (98.33 and 99.34), PSO_bound (94.17 and 96.26), and SGD (94.17 and 95.82) in the iris and breast datasets.

In the testing phase, the circle and iris datasets without normalization and 100 iterations, QDPSO (100 and 100), PSO (100 and 100), and L-BFGS (100 and 100) are better than PSO_bound (89 and 96.67), Adam (78 and 93.33), and SGD (78 and 96.67), see Figures 5.1c and 5.1d. Furthermore, QDPSO is better than PSO, PSO_bound, Adam, L-BFGS, and SGD in the wine dataset; 86.61, 72.22, 61.11, 69.44, 77.78, and 61.11, respectively. In addition, in the breast dataset, QDPSO, PSO, and Adam are better than PSO_bound, L-BFGS, and SGD; correspondingly, 95.61, 95.61, 95.61, 93.86, 94.74, and 94.74. However, with normalization, QDPSO (100 and 100), PSO (100 and 100), and L-BFGS (100 and 100) are better than PSO_bound (70 and 97.22), Adam (73 and 94.44), and SGD (75 and 63.89) in the circle and wine datasets. Moreover, in the iris dataset, QDPSO and PSO are better than PSO_bound, Adam, L-BFGS, and SGD; 100, 100, 96.67, 93.33, 96.67, and 80, appropriately. In addition, L-BFGS and Adam are better than QDPSO, PSO, PSO_bound, and SGD in the breast dataset; 98.25, 98.25, 95.61, 97.37, 95.61, and 90.35, respectively. Conversely, without normalization and 1000 iterations, QDPSO, PSO, Adam, and L-BFGS are better than PSO_bound and SGD in the circle dataset; correspondingly, 100, 100, 100, 100, 94, and 73, see Figures 5.2c and 5.2d. Furthermore, in the breast dataset, QDPSO and L-BFGS are better than PSO, PSO_bound, Adam, and SGD; 96.49, 96.49, 94.74, 94.74, 95.61, and 92.11, appropriately. In addition, in the wine dataset, L-BFGS is better than QDPSO, PSO, PSO_bound, Adam, and SGD; 100, 97.22, 94.44, 83.33, 97.22, and 58.33, respectively. The metaheuristic-based and gradient-based optimizers have the same results, 100, in the iris dataset. Nonetheless, QDPSO, PSO, Adam, and L-BFGS are better than PSO_bound and SGD in the circle dataset with normalization; correspondingly, 100, 100, 100, 100, 72, and 71. Furthermore, the metaheuristic-based optimizers, 100, are better than gradient-based optimizers, 96.67, in the iris dataset. In addition, in the wine dataset, QDPSO, PSO, L-BFGS, and SGD are better than Adam and PSO_bound; 100,

100, 100, 100, 97.22, and 94.44, appropriately. Besides, in the breast dataset, QDPSO, PSO, PSO_bound, and L-BFGS are better than SGD and Adam; 98.25, 98.25, 98.25, 98.25, 97.37, 96.49, respectively.

5.1.4 The metrics of the metaheuristic-based optimizers

eee Table 5.1 describes the best, worst, average, and standard deviation of the loss cost using the metaheuristic-based and gradient-based optimizers for the benchmark datasets. Furthermore, we use 10 training with 1000 iterations and normalization. In the metaheuristic-based optimizers and the best cost, QDPSO (1.25E-06 and 3.27E-06) and PSO (1.25E-06 and 3.27E-06) are better than PSO_bound (6.56E-01 and 3.35E-01) in the circle and wine datasets. Moreover, in the iris dataset, QDPSO is better than PSO and PSO_bound; correspondingly, 4.16E-06, 4.37E-06, and 5.258E-01. Besides, PSO is better than QDPSO and PSO_bound in the breast dataset; 3.45E-04, 6.57E-03, and 1.40E-01, appropriately. However, at the worst cost, PSO_bound is worse than QDPSO and PSO in all benchmark datasets; see Table 5.1.

The average cost of QDPSO (1.25E-06 and 2.61E-03) is better than PSO (4.66E-02 and 2.77E-03) and PSO_bound (6.64E-01 and 5.81E-01) in the circle and iris datasets. Moreover, in the wine and breast datasets, PSO (3.54E-06 and 1.19E-02) is better than QDPSO (3.66E-06 and 1.71E-02) and PSO_bound (4.30E-01 and 1.69E-01). On the other hand, in the standard deviation cost, PSO (2.61E-03, 3.66E-08, and 6.46E-03) is better than QDPSO (6.06E-03, 2.98E-07, and 8.81E-03) and PSO_bound (3.67E-02, 4.01E-02, and 2.24E-02) in the iris, wine, and breast datasets. However, QDPSO is better than PSO and PSO_bound in the circle dataset; 5.06E-16, 7.13E-02, and 4.58E-03, respectively.

5.1.5 The metrics of the gradient-based optimizers

L-BFGS is better than Adam and SGD in all benchmark datasets for the gradient-based optimizers and at the best and average cost, see Table 5.1. Moreover, SGD is worse than L-BFGS and Adam at the worst cost in all benchmark datasets. On the other hand, the standard deviation cost of L-BFGS (3.88E-04, 1.16E-04, and 1.14E-04) is better than Adam (3.14E-02, 2.84E-03, and 4.82E-04) and SGD (2.60E-02, 1.70E-02, and 1.79E-03) in

Optimizers	Datasets	Worst	Avg	Best	Std
QDPSO	circle	1.25E-06	**1.25E-06**	**1.25E-06**	**5.06E-16**
	iris	2.01E-02	**2.61E-03**	**4.16E-06**	6.06E-03
	wine	4.48E-06	3.66E-06	**3.52E-06**	2.98E-07
	breast	3.42E-02	1.71E-02	6.57E-03	8.81E-03
PSO	circle	1.64E-01	4.66E-02	**1.25E-06**	7.13E-02
	iris	7.59E-03	2.77E-03	4.37E-06	2.61E-03
	wine	3.61E-06	**3.54E-06**	**3.52E-06**	**3.66E-08**
	breast	2.12E-02	1.19E-02	**3.45E-04**	6.46E-03
PSO_bound	circle	6.72E-01	6.64E-01	6.56E-01	4.58E-03
	iris	**6.63E-01**	5.81E-01	5.25E-01	3.67E-02
	wine	**4.99E-01**	4.30E-01	3.35E-01	4.01E-02
	breast	**2.10E-01**	1.69E-01	1.40E-01	2.24E-02
Adam	circle	6.46E-01	4.93E-01	3.37E-01	9.83E-02
	iris	2.58E-01	2.01E-01	1.65E-01	3.14E-02
	wine	2.41E-02	1.89E-02	1.55E-02	2.84E-03
	breast	4.07E-02	3.98E-02	3.91E-02	4.82E-04
L-BFGS	circle	1.45E-01	1.54E-02	6.85E-04	4.33E-02
	iris	4.65E-03	3.85E-03	3.51E-03	**3.88E-04**
	wine	1.13E-03	9.83E-04	7.41E-04	1.16E-04
	breast	1.83E-03	**1.69E-03**	1.53E-03	**1.14E-04**
SGD	circle	**6.89E-01**	6.85E-01	6.80E-01	3.20E-03
	iris	5.62E-01	5.01E-01	4.62E-01	2.60E-02
	wine	2.96E-01	2.70E-01	2.45E-01	1.70E-02
	breast	1.47E-01	1.45E-01	1.42E-01	1.79E-03

Table 5.1: The best, worst, average, and standard deviation of the loss cost during the 10 training with 1000 iterations and normalization for the metaheuristic-based and gradient-based optimizers.

the iris, wine, and breast datasets. Furthermore, SGD is better than L-BFGS and Adam in the circle dataset; correspondingly, 3.20E-03, 4,33E-02, and 9,83E-02.

5.1.6 The metrics of the metaheuristic-based and gradient-based optimizers

In the metaheuristic-based and gradient-based optimizers, QDPSO (1.25E-06 and 3.52E-06) and PSO (1.25E-06 and 3.52E-06) are better than L-BFGS (6.85E-04 and 7.41E-04), Adam (3.37E-01 and 1.552E-02), PSO_bound (6.56E-01 and 3.35E-01), and SGD (6.80E-01 and 2.45E-01) in the best cost for the circle and wine datasets. Furthermore, in the iris dataset, QDPSO is better than PSO, L-BFGS, Adam, SGD, and PSO_bound; 4.16E-06,

4.37E-06, 3.51E-03, 1.65E-01, 4.62E-01, and 5.25E-01, appropriately. In addition, in the breast dataset, PSO is better than L-BFGS, QDPSO, Adam, PSO_bound, and SGD; 3.45E-04, 1.53E-03, 6.57E-03, 3.91E-02, 1.40E-01, and 1.42E-01, respectively. Nonetheless, in the average cost, QDPSO (1.25E-06 and 2.61E-03) is better than L-BFGS (1.54E-02 and 3.85E-03), PSO (4.66E-02 and 2.77E-03), Adam (4.93E-01 and 2.01E-01), PSO_bound (6.64E-01 and 5.81E-01), and SGD (6.85E-01 and 5.01E-01) in the circle and iris dataset. Moreover, in the wine dataset, PSO is better than QDPSO, L-BFGS, Adam, SGD, and PSO_bound; correspondingly, 3.54E-06, 3.66E-06, 9.83E-04, 1.89E-02, 2.70E-01, and 4.30E-01. Further, L-BFGS is better than PSO, QDPSO, Adam, SGD, and PSO_bound; 1.69E-03, 1.19E-02, 1.71E-02, 3.98E-02, 1.45E-01, and 1.69E-01, appropriately. Conversely, in the worst cost, PSO_bound is worse than SGD, Adam, L-BFGS, QDPSO, and PSO in the iris, wine, and breast datasets; see Table 5.1. Besides, SGD is worse than PSO_bound, Adam, PSO, L-BFGS, and QDPSO in the circle dataset.

The standard deviation of the L-BFGS (3.88E-04 and 1.14E-04) is better than PSO (2.61E-03 and 6.46E-03), QDPSO (6.06E-03 and 8.81E-03), Adam (3.14E-02 and 4.82E-04), SGD (2.60E-02 and 1.79E-03), and PSO_bound (3.67E-02 and 2.24E-02) in the iris and breast datasets. Furthermore, in the wine dataset, PSO is better than QDPSO, L-BFGS, Adam, SGD, and PSO_bound; 3.66E-08, 2.98E-07, 1.16E-04, 2.84E-03, 1.16E-04, and 4.01E-02, respectively. In addition, in the circle dataset, QDPSO is better than SGD, PSO_bound, L-BFGS, PSO, and Adam; correspondingly, 5.06E-16, 3.20E-03, 4.58E-03, 4.33E-02, 7.13E-02, 9.83E-02.

Table 5.2 shows the precision, recall, and f1 score of the circle, iris, and wine datasets with metaheuristic-based and gradient-based optimizers. Furthermore, we use the best training with 1000 iterations and a normalization dataset. On the other hand, 1.00 is the precision, recall, and f1 score of the metaheuristic-based optimizers in the circle, iris, and wine datasets. Nevertheless, in the breast dataset, the precision, recall, and f1 score are 9.82E-01. Conversely, the precision, recall, and f1 score are 1.00 for the L-BFGS and Adam optimizers in the circle dataset. In SGD, 8.28E-01, 7.10E-01, and 6.99E-01 correlate with the precision, recall, and f1 score. Furthermore, in the iris dataset, 9.69E-01, 9.67E-01, and 9.66E-01 correspond to the precision, recall, and f1 score, respectively, for the L-BFGS and Adam. In addition, in the wine dataset, L-BFGS and SGD have the same precision, recall,

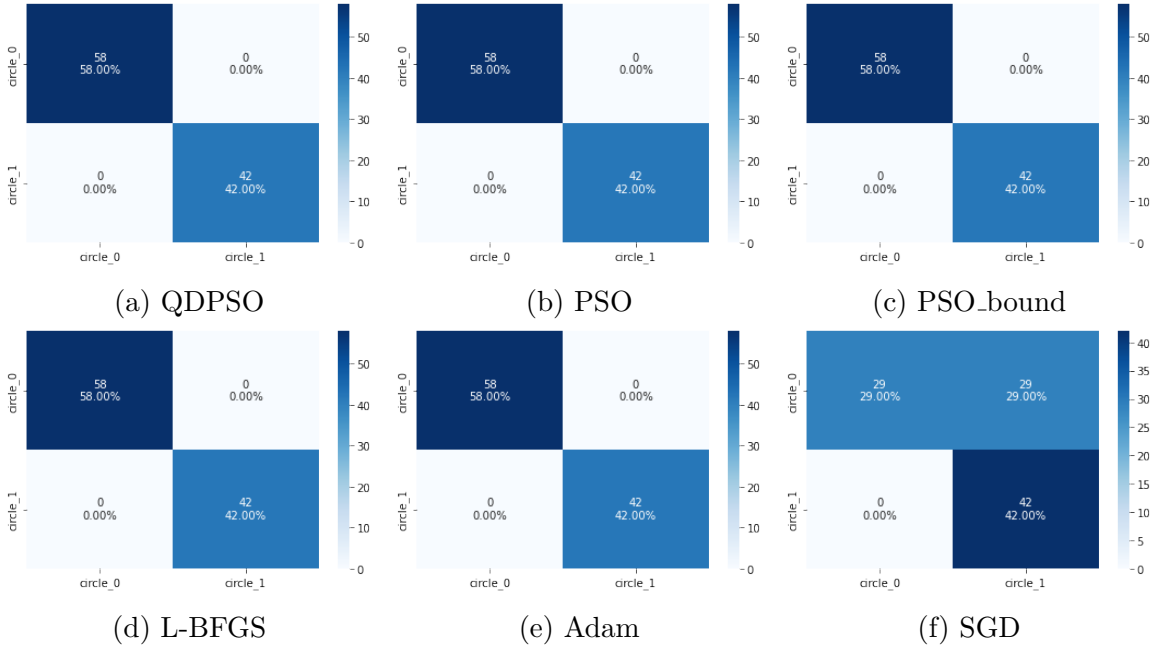
Optimizers	Datasets	Precision	Recall	F1 score
QDPSO	circle	1.00	1.00	1.00
	iris	1.00	1.00	1.00
	wine	1.00	1.00	1.00
	breast	9.82E-01	9.82E-01	9.82E-01
PSO	circle	1.00	1.00	1.00
	iris	1.00	1.00	1.00
	wine	1.00	1.00	1.00
	breast	9.82E-01	9.82E-01	9.82E-01
PSO_bound	circle	1.00	1.00	1.00
	iris	1.00	1.00	1.00
	wine	1.00	1.00	1.00
	breast	9.82E-01	9.82E-01	9.82E-01
Adam	circle	1.00	1.00	1.00
	iris	9.69E-01	9.67E-01	9.66E-01
	wine	9.74E-01	9.72E-01	9.72E-01
	breast	9.65E-01	9.65E-01	9.65E-01
L-BFGS	circle	1.00	1.00	1.00
	iris	9.69E-01	9.67E-01	9.66E-01
	wine	1.00	1.00	1.00
	breast	9.83E-01	9.82E-01	9.82E-01
SGD	circle	8.28E-01	7.10E-01	6.99E-01
	iris	9.69E-01	9.67E-01	9.66E-01
	wine	1.00	1.00	1.00
	breast	9.75E-01	9.74E-01	9.74E-01

Table 5.2: The precision, recall, and f1 score of benchmark datasets with 1000 iterations and normalization for the metaheuristic-based and gradient-based optimizers.

and f1 scores of 1.00. However, in Adam, the precision, recall, and f1 score are 9.74E-01, 9.72E-01, and 9.72E-01. Finally, in the breast dataset, 9.83E-01, 9.82E-01, and 9.82E-01 are the precision, recall, and f1 score of L-BFGS. However, in Adam, 9.65E-01 corresponds to the precision, recall, and f1 score. Besides, the precision, recall, and f1 score of SGD are 9.75E-01, 9.74E-01, and 9.74E-01, respectively.

5.1.7 Confusion Matrix and Loss Cost Curve

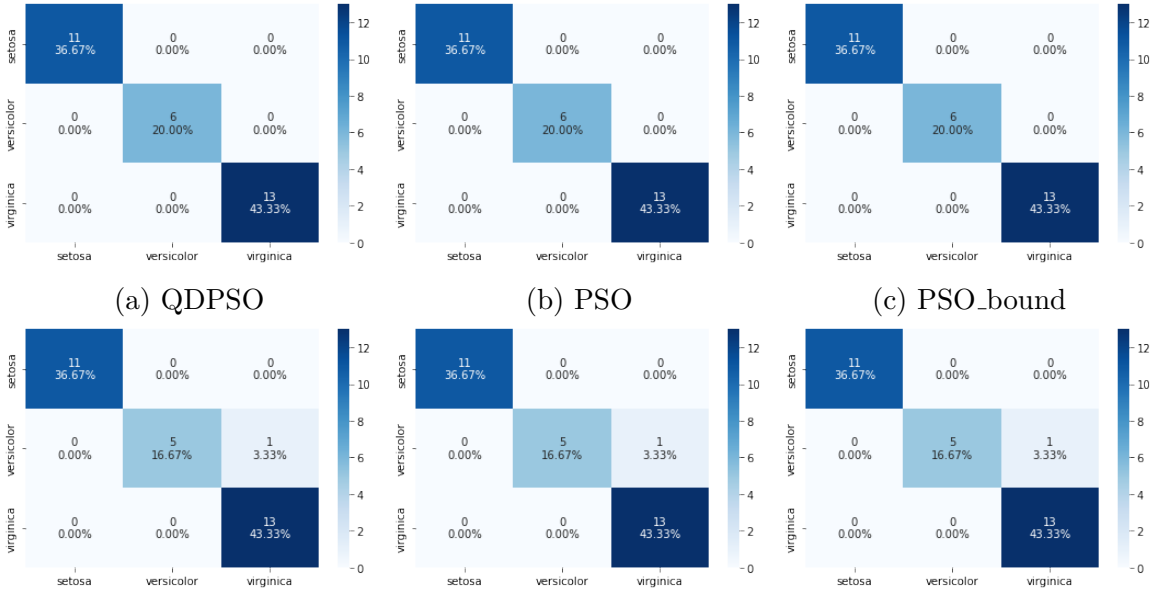
Figures 5.3, 5.4, 5.5, and 5.6 describe the confusion matrix of the circle, iris, wine, and breast datasets, respectively. Furthermore, Figures a, b, and c illustrate the metaheuristic-based optimizers, while Figures d, e, and f are the gradient-based optimizers. In addition, the confusion matrix uses the best training, dataset normalized, and 1000 iterations.



(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

Figure 5.3: The confusion matrix of the circle dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization and 1000 iterations.

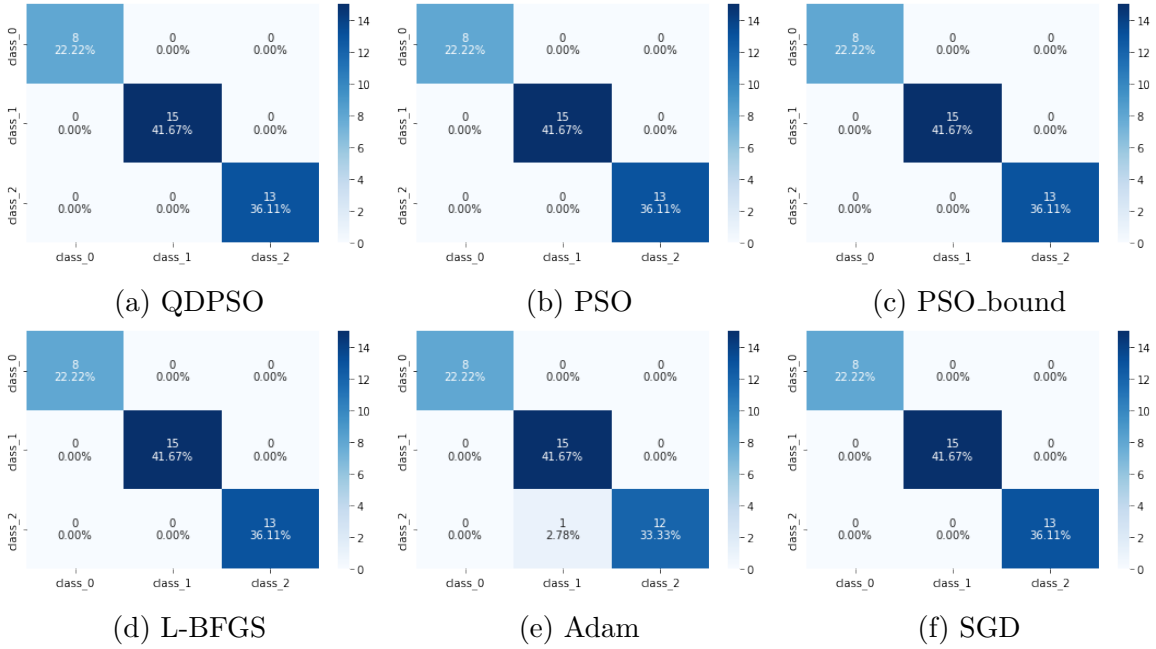


(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

Figure 5.4: The confusion matrix of the iris dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization and 1000 iterations.

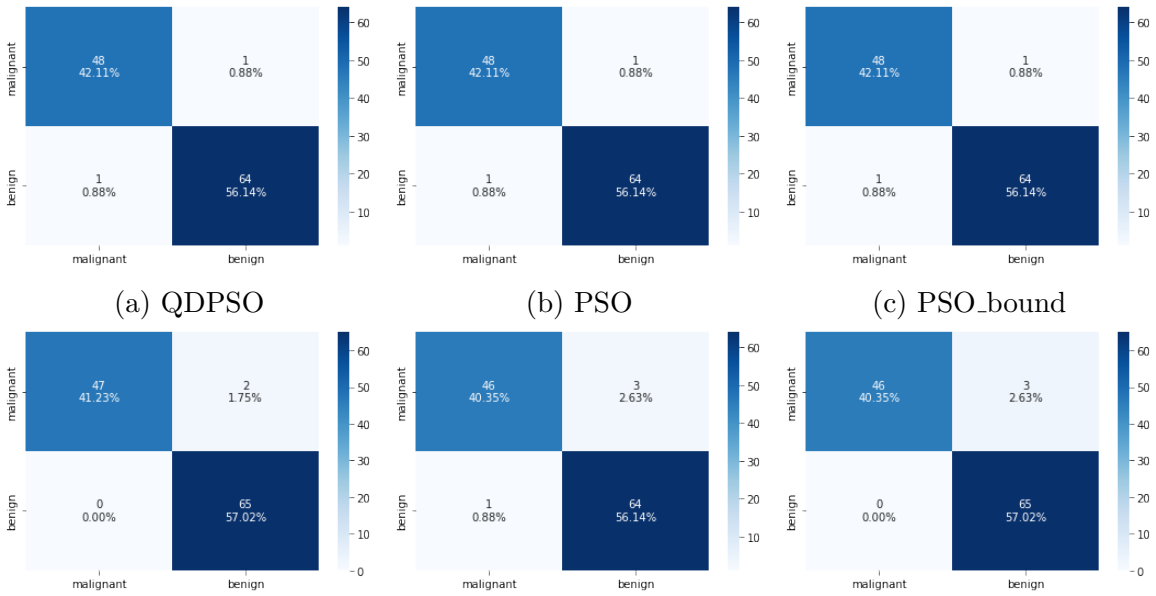
The sum main diagonal of Figure 5.3a shows the precision of the QDPSO classifier with the circle dataset, which is 100% divides into 58% to class 0 and 42% to class 1. Moreover, the same precision is for PSO, PSO_bound, L-BFGS, and adam; see Figures 5.3b, 5.3c,



(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

Figure 5.5: The confusion matrix of the wine dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization and 1000 iterations.



(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

Figure 5.6: The confusion matrix of the breast dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization and 1000 iterations.

5.3d, and 5.3e. Nevertheless, SGD has 71% of the precision divided into 29% for class 0 and 42% for class 1, see Figure 5.3f. Furthermore, 29% predict that class 0 is class 1.

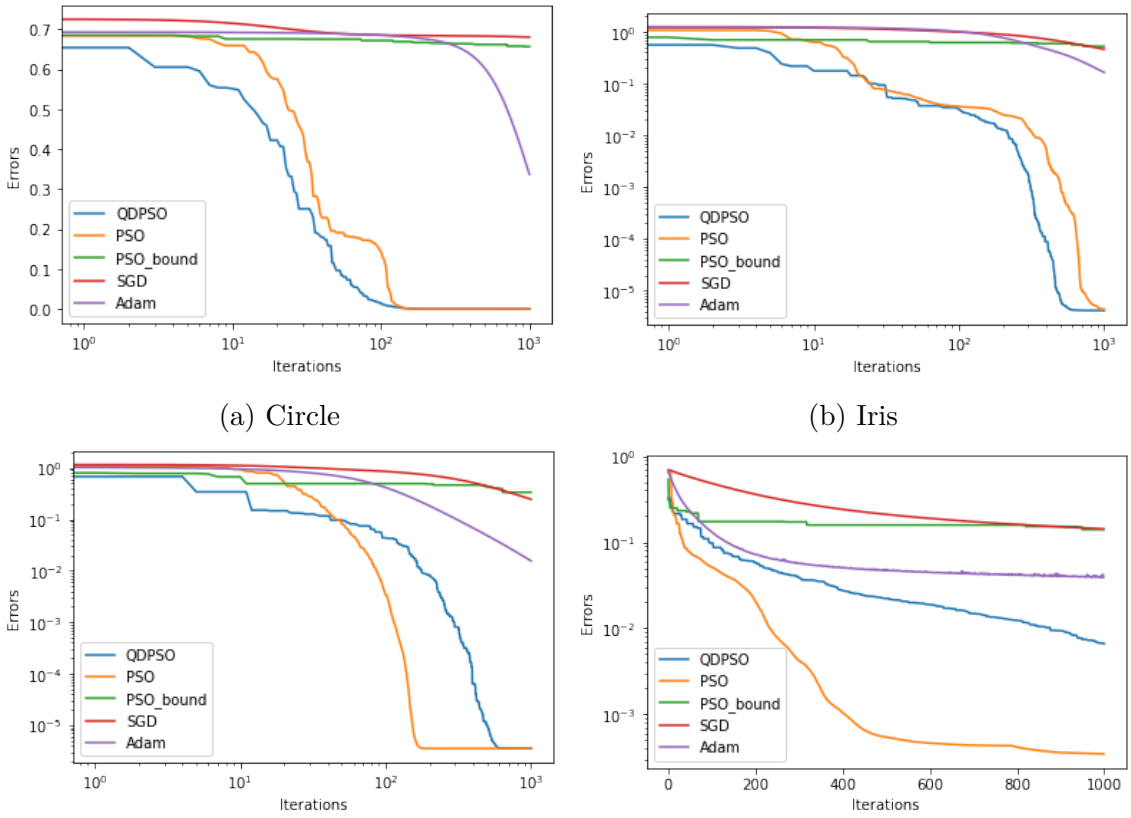
In Figure 5.4a, the QDPSO classifier in the iris dataset has 100% precision, divided into

36.67% for setosa, 20% for versicolor, and 43.33% for virginica. The same precision results have PSO and PSO_bound in Figures 5.4b and 5.4c. On the other hand, 96.67% of the precision for L-BFGS, Adam, and SGD; see Figures 5.4d, 5.4e, and 5.4f. This precision is divided into 36.67% to setosa, 16.67% to versicolor, and 43.33% to virginica. Furthermore, Figures 5.4d, 5.4e, and 5.4f show that the error of classification of the gradient-based optimizers is 3.33%. This error predicts that versicolor is a virginica.

The precision of the QDPSO classifier is 100% in the wine dataset, which is distributed in 22% for class 0, 41.67% for class 1, and 36.11% for class 2; see Figure 5.5a. This precision result is the same for the PSO, PSO_bound, L-BFGS, and SGD, as shown in Figures 5.5b, 5.5c, 5.5d, and 5.5f. Nonetheless, in Figure 5.5e, the Adam classifier has 97.22% of the precision divided into 22.22% for class 0, 41.67% for class 1, and 33.33% for class 2. Furthermore, the 2.78% corresponds to the error classification, which predicts that class 2 is class 1.

Figure 5.6a shows in the main diagonal that 98.25% corresponds to the precision of the QDPSO in the breast dataset. This precision is distributed in 42.11% for the malignant class and 56.14% for the benign class. Furthermore, 1.76% corresponds to the error classification, where 0.88% predicts that the malignant class is the benign class, and another 0.88% forecasts that the benign class is the malignant class. In addition, PSO and PSO_bound have the same results; see Figures 5.6b and 5.6c. However, the L-BFGS classifier has 98.25% precision, divided into 41.23% for the malignant class and 57.02% for the benign class; see Figure 5.6d. Besides, 1.75% corresponds to the error classification, which forecasts that the malignant class is the benign class. The Adam classifier of the main diagonal has 96.49% precision distributed in 40.35% for the malignant class and 56.14% for the benign class, see Figure 5.6e. In addition, 3.51% correlates with the error classification, of which 2.63% predicts that the malignant class is the benign class and 0.88% forecasts that the benign class is the malignant class. Finally, the precision of the SGD is 97.37%, divided into 40.35% for the malignant class and 57.02% for the benign class; see Figure 5.6f. The error classification is 2.63%, which thinks the malignant class is benign.

Figure 5.7 describes the convergence of the loss cost for the optimizers proposed in this work. Furthermore, we use the best training, 1000 iterations, and normalization of the datasets. Unfortunately, L-BFGS could not plot because the MLPClassifier framework



(a) Circle (b) Iris

(c) Wine (d) Breast

Figure 5.7: The loss curve of the benchmark datasets uses metaheuristic-based and gradient-based optimizers with normalization and 1000 iterations.

does not allow it. In metaheuristic-based optimizers, Figures 5.7a and 5.7b depict that the QDPSO curve is better than the PSO and PSO_bound curves in the circle and iris datasets. However, in the wine and breast datasets, the PSO curve is better than the QDPSO and PSO_bound curves, as shown in Figures 5.7c and 5.7d. In addition, the Adam curve is better than the SGD curve in the circle, iris, wine, and breast datasets for the gradient-based optimizers; see Figure 5.7. Conversely, in the metaheuristic-based and gradient-based optimizers, the QDPSO curve is better than the PSO, Adam, PSO_bound, and SGD curves. Nonetheless, in the wine and breast datasets, the PSO curve is better than QDPSO, Adam, SGD, and PSO_bound curves.

5.1.8 Discussion of Benchmark Performance

The metaheuristic-based and gradient-based optimizers handle the benchmark datasets according to the f1 score and deviation standard of Tables 5.2 and 5.1, appropriately.

Except, SDG has a problem with the circle dataset. On the other hand, the results of Figures 5.1 and 5.2 show that the metaheuristic-based classifier has better results than the gradient-based classifier with 100 and 1000 iterations. Furthermore, QDPSO, PSO, and L-BFGS have similar results in the training and testing phases in all benchmarks with 1000 iterations and normalization; see Figure 5.2. Besides, the results of the PSO_bound like to the results of the SGD.

QDPSO has the best training and testing phases without normalization and 100 iterations in the wine dataset. While with normalization, L-BFGS has the best training stage, in the testing phase, QDPSO and L-BFGS are the best than other classifiers. Furthermore, the results of the iris dataset without normalization are similar. However, in the circle dataset, QDPSO, PSO, and L-BFGS have the same results with and without normalization in the training and testing stages. In the breast dataset¡, the LBFGS is better than other optimizers. Then, with 1000 iterations, QDPSO, PSO, and L-BFGS have similar results with and without normalization in the circle, iris, and breast datasets. In the wine dataset, QDPSO has the best training phase. Nevertheless, in the testing stage, L-BFGS is the best; see Figures 5.1 and 5.2.

The PSO curve of the wine and breast datasets is better than other metaheuristic-based classifiers, as shown in Figure 5.7. However, the QDPSO is better than other metaheuristic-based optimizers in the circle and iris datasets because the datasets are balanced, contrary to the first case (see Table 4.1). Conversely, in the benchmark datasets, the behavior of the QDPSO and PSO curves have faster convergence than all optimizers proposed in this work. Thus, QDPSO has faster behavior and convergence than PSO_bound because the PSO_bound tends to get stuck during training. Finally, the PSO_bound curve has similar behavior to SGD.

5.2 Performance of Dimensions Reduction

This section describes the dimension reduction results for input nodes of the artificial neural network. Moreover, we use a small features dataset (Iris) and a virtually extensive features dataset (Breast Cancer) with 1000 iterations and the normalization of the datasets. Finally, we use reducing algorithms to check and analyze the dimension reduction results.

On the other hand, the selection criterion is given by the best result in the smallest possible dimension considering the original dimension in the testing phase and the maximum number of optimizers.

Alg.	**D**	**Dataset**	**QDPSO**		**PSO**		**PSO_bound**	
			MSE	**ACC**	**MSE**	**ACC**	**MSE**	**ACC**
NA	4	test	0.00	100.00	0.00	100.00	3.33E-02	96.67
		train	0.00	100.00	0.00	100.00	9.17E-02	90.83
MDS	3	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**
		train	0.00	100.00	0.00	100.00	4.17E-02	95.83
	2	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**
		train	0.00	100.00	0.00	100.00	1.08E-01	89.17
	1	test	1.67E-01	93.33	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**
		train	5.00E-02	95.00	9.17E-02	93.33	5.83E-02	94.17
TSNE	3	test	1.67E-01	83.33	3.33E-02	96.67	**0.00**	**100.00**
		train	0.00	100.00	3.33E-02	96.67	4.17E-02	95.83
	2	test	**0.00**	**100.00**	**0.00**	**100.00**	**0.00**	**100.00**
		train	2.50E-02	97.50	3.33E-02	96.67	4.17E-02	95.83
	1	test	**0.00**	**100.00**	1.33E-01	96.67	**0.00**	**100.00**
		train	5.00E-02	95.00	3.33E-02	96.67	4.17E-02	95.83
Isomap	**3**	test	**3.33E-02**	**96.67**	**0.00**	**100.00**	**3.33E-02**	**96.67**
		train	0.00	100.00	0.00	100.00	7.50E-02	92.50
	2	test	**3.33E-02**	**96.67**	3.33E-02	96.67	**3.33E-02**	**96.67**
		train	0.00	100.00	0.00	100.00	8.33E-02	91.67
	1	test	**3.33E-02**	**96.67**	3.33E-02	96.67	**3.33E-02**	**96.67**
		train	4.17E-02	95.83	4.17E-02	95.83	1.08E-01	89.17
PCA	**3**	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**0.00**	**100.00**
		train	0.00	100.00	8.33E-03	99.17	5.00E-02	95.00
	2	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	6.67E-02	93.33
		train	5.00E-02	95.00	5.83E-02	94.17	1.58E-01	84.17
	1	test	6.67E-02	93.33	6.67E-02	93.33	3.33E-02	96.67
		train	7.50E-02	92.50	7.50E-02	92.50	9.17E-02	90.83

Table 5.3: The dimension reduction results use the iris dataset and metaheuristic-based optimizers with MDS, TSNE, Isomap, and PCA as reducing algorithms.

5.2.1 Iris Dataset

Table 5.3 shows the results of the metaheuristic-based optimizers using the reducing algorithm. Moreover, in MDS and TSNE, all metaheuristic-based optimizers have the best result (96.67 for QDPSO, PSO, and PSO_bound) in 2 dimensions and Isomap and PCA in 3 dimensions (100 and 96.67 for PSO, 96.67 and 96.67 for QDPSO, and 96.67 and 100 PSO_bound). On the other hand, TSNE has the best result (100 for the metaheuristic-

Alg.	D	Dataset	Adam		SGD		L-BFGS	
			MSE	**ACC**	**MSE**	**ACC**	**MSE**	**ACC**
NA	4	test	3.33E-02	96.67	3.33E-02	96.67	3.33E-02	96.67
		train	4.17E-02	95.83	4.17E-02	95.83	0.00	100.00
MDS	3	test	**3.33E-02**	**96.67**	3.33E-02	96.67	**3.33E-02**	**96.67**
		train	2.50E-02	97.50	1.33E-01	86.67	0.00	100.00
	2	test	**3.33E-02**	**96.67**	**0.00**	**100.00**	**3.33E-02**	**96.67**
		train	1.08E-01	89.17	5.00E-02	95.00	8.33E-03	99.17
	1	test	**3.33E-02**	**96.67**	3.33E-02	96.67	1.67E-01	93.33
		train	9.17E-02	90.83	1.17E-01	88.33	8.33E-02	94.17
TSNE	3	test	3.33E-02	96.67	**0.00**	**100.00**	**0.00**	**100.00**
		train	2.50E-02	97.50	9.17E-02	95.83	0.00	100.00
	2	test	**0.00**	**100.00**	3.33E-02	96.67	**0.00**	**100.00**
		train	4.17E-02	95.83	5.00E-02	95.00	3.33E-02	96.67
	1	test	**0.00**	**100.00**	**0.00**	**100.00**	**0.00**	**100.00**
		train	5.00E-02	95.00	5.00E-02	95.00	3.33E-02	96.67
Isomap	**3**	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**
		train	5.83E-02	94.17	5.00E-02	95.00	0.00	100.00
	2	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**
		train	5.83E-02	94.17	5.00E-02	95.00	0.00	100.00
	1	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**
		train	8.33E-02	91.67	9.17E-02	90.83	4.17E-02	95.83
PCA	**3**	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	**0.00**	**100.00**
		train	2.50E-02	97.50	5.83E-02	94.17	0.00	100.00
	2	test	6.67E-02	93.33	6.67E-02	93.33	3.33E-02	96.67
		train	7.50E-02	92.50	1.50E-01	85.00	3.33E-02	96.67
	1	test	**3.33E-02**	**96.67**	**3.33E-02**	**96.67**	6.67E-02	93.33
		train	9.17E-02	90.83	9.17E-02	90.83	1.00E-01	90.00

Table 5.4: The dimension reduction results use the iris dataset and gradient-based optimizers with MDS, TSNE, Isomap, and PCA as reducing algorithms.

based optimizers) and reduction dimensions (2), and PSO_bound with 2 dimensions has better results (100) than the original dimensions (96.67).

The results of gradient-based optimizers with reducing algorithms are shown in Table 5.4. The best result in MDS (100 for PSO, 96.67 for QDPSO, and 96.67 for PSO_bound) is in 2 dimensions, TSNE (100 for the metaheuristic-based optimizers) is in 1 dimension, PCA (96.67 for QDPSO and PSO, and 100 for PSO_bound) is in 3 dimensions, and Isomap (96.67) is in all dimensions. However, the best reduction dimensions and result in TSNE and all gradient-based optimizers with 1 dimension have a better result than the original dimensions. Thus, Tables 5.3 and 5.4 show that TSNE is a better reducing algorithm than other Isomap, MDS, and PCA with 2 dimensions.

Alg.	D	Dataset	QDPSO		PSO		PSO_bound	
			MSE	ACC	MSE	ACC	MSE	ACC
NA	30	test	1.75E-02	98.25	8.77E-03	99.12	2.63E-02	97.37
		train	0.00	100.00	0.00	100.00	3.52E-02	96.48
MDS	25	test	3.51E-02	96.49	**1.75E-02**	**98.25**	**2.63E-02**	**97.37**
		train	0.00	100.00	4.40E-03	99.56	2.64E-02	97.36
	20	test	4.39E-02	95.61	3.51E-02	96.49	**2.63E-02**	**97.37**
		train	0.00	100.00	0.00	100.00	3.52E-02	96.48
	15	test	**8.77E-03**	**99.12**	2.63E-02	97.37	**2.63E-02**	**97.37**
		train	0.00	100.00	0.00	100.00	2.64E-02	97.36
	10	test	1.75E-02	98.25	**1.75E-02**	**98.25**	3.51E-02	96.49
		train	0.00	100.00	8.79E-03	99.12	3.30E-02	96.70
	5	test	2.63E-02	97.37	2.63E-02	97.37	**2.63E-02**	**97.37**
		train	0.00	100.00	1.32E-02	98.68	4.18E-02	95.82
TSNE	3	test	3.51E-02	96.49	2.63E-02	97.37	**2.63E-02**	**97.37**
		train	0.00	100.00	1.76E-02	98.24	2.20E-02	97.80
	2	test	**2.63E-02**	**97.37**	**1.75E-02**	**98.25**	4.39E-02	95.61
		train	2.86E-02	97.14	5.93E-02	94.07	4.18E-02	95.82
	1	test	3.51E-02	96.49	3.51E-02	96.49	7.02E-02	92.98
		train	2.86E-02	97.14	3.08E-02	96.92	5.71E-02	94.29
Isomap	25	test	1.75E-02	98.25	2.63E-02	97.37	2.63E-02	97.37
		train	0.00	100.00	4.40E-03	99.56	2.64E-02	97.36
	20	test	**8.77E-03**	**99.12**	1.75E-02	98.25	**1.75E-02**	**98.25**
		train	0.00	100.00	0.00	100.00	2.86E-02	97.14
	15	test	1.75E-02	98.25	**8.77E-03**	**99.12**	3.51E-02	96.49
		train	0.00	100.00	6.59E-03	99.34	3.08E-02	96.92
	10	test	2.63E-02	97.37	2.63E-02	97.37	2.63E-02	97.37
		train	0.00	100.00	2.20E-03	99.78	2.86E-02	97.14
	5	test	2.63E-02	97.37	**8.77E-03**	**99.12**	3.51E-02	96.49
		train	0.00	100.00	4.40E-03	99.56	2.64E-02	97.36
PCA	25	test	**1.75E-02**	**98.25**	1.75E-02	98.25	2.63E-02	97.37
		train	0.00	100.00	0.00	100.00	1.54E-02	98.46
	20	test	2.63E-02	97.37	2.63E-02	97.37	**1.75E-02**	**98.25**
		train	0.00	100.00	0.00	100.00	1.76E-02	98.24
	15	test	2.63E-02	97.37	**8.77E-03**	**99.12**	**1.75E-02**	**98.25**
		train	0.00	100.00	0.00	100.00	1.32E-02	98.68
	10	test	**1.75E-02**	**98.25**	1.75E-02	98.25	**1.75E-02**	**98.25**
		train	0.00	100.00	0.00	100.00	1.76E-02	98.24
	5	test	**1.75E-02**	**98.25**	2.63E-02	97.37	2.63E-02	97.37
		train	0.00	100.00	4.40E-03	99.56	2.64E-02	97.36

Table 5.5: The dimension reduction results use the breast cancer dataset and metaheuristic-based optimizers with MDS, TSNE, Isomap, and PCA as reducing algorithms.

Alg.	D	Dataset	Adam		SGD		L-BFGS	
			MSE	ACC	MSE	ACC	MSE	ACC
NA	30	test	2.63E-02	97.37	3.51E-02	96.49	1.75E-02	98.25
		train	8.79E-03	99.12	3.96E-02	96.04	0.00	100.00
MDS	25	test	**1.75E-02**	**98.25**	4.39E-02	95.61	3.51E-02	96.49
		train	4.40E-03	99.56	3.96E-02	96.04	0.00	100.00
	20	test	3.51E-02	96.49	**3.51E-02**	**96.49**	1.75E-02	98.25
		train	0.00	100.00	3.74E-02	96.26	0.00	100.00
	15	test	2.63E-02	97.37	4.39E-02	95.61	**8.77E-03**	**99.12**
		train	0.00	100.00	4.18E-02	95.82	0.00	100.00
	10	test	**1.75E-02**	**98.25**	4.39E-02	95.61	1.75E-02	98.25
		train	8.79E-03	99.12	4.18E-02	95.82	0.00	100.00
	5	test	2.63E-02	97.37	**3.51E-02**	**96.49**	1.75E-02	98.25
		train	1.32E-02	98.68	3.96E-02	96.04	0.00	100.00
TSNE	3	test	**2.63E-02**	**97.37**	**1.75E-02**	**98.25**	2.63E-02	97.37
		train	3.08E-02	96.92	3.08E-02	96.92	6.59E-03	99.34
	2	test	**2.63E-02**	**97.37**	**1.75E-02**	**98.25**	**1.75E-02**	**98.25**
		train	3.52E-02	96.48	5.49E-02	94.51	2.42E-02	97.58
	1	test	8.77E-02	91.23	6.14E-02	93.86	3.51E-02	96.49
		train	8.57E-02	91.43	6.37E-02	93.63	2.64E-02	97.36
Isomap	25	test	4.39E-02	95.61	4.39E-02	95.61	1.75E-02	98.25
		train	1.54E-02	98.46	3.96E-02	96.04	0.00	100.00
	20	test	**3.51E-02**	**96.49**	4.39E-02	95.61	2.63E-02	97.37
		train	1.98E-02	98.02	3.52E-02	96.48	0.00	100.00
	15	test	**3.51E-02**	**96.49**	3.51E-02	96.49	2.63E-02	97.37
		train	1.76E-02	98.24	3.74E-02	96.26	0.00	100.00
	10	test	4.39E-02	95.61	3.51E-02	96.49	**8.77E-03**	**99.12**
		train	1.76E-02	98.24	4.18E-02	95.82	0.00	100.00
	5	test	**3.51E-02**	**96.49**	**2.63E-02**	**97.37**	1.75E-02	98.25
		train	1.98E-02	98.02	3.74E-02	96.26	0.00	100.00
PCA	25	test	2.63E-02	97.37	2.63E-02	97.37	8.77E-03	99.12
		train	6.59E-03	99.34	1.54E-02	98.46	0.00	100.00
	20	test	**1.75E-02**	**98.25**	**1.75E-02**	**98.25**	1.75E-02	98.25
		train	6.59E-03	99.34	1.76E-02	98.24	0.00	100.00
	15	test	2.63E-02	97.37	**1.75E-02**	**98.25**	8.77E-03	99.12
		train	6.59E-03	99.34	1.98E-02	98.02	0.00	100.00
	10	test	2.63E-02	97.37	**1.75E-02**	**98.25**	2.63E-02	97.37
		train	1.10E-02	98.90	2.20E-02	97.80	0.00	100.00
	5	test	2.63E-02	97.37	2.63E-02	97.37	**0.00**	**100.00**
		train	1.98E-02	98.02	2.64E-02	97.36	0.00	100.00

Table 5.6: The dimension reduction results use the breast cancer dataset and gradient-based optimizers with MDS, TSNE, Isomap, and PCA as reducing algorithms.

5.2.2 Breast Cancer Dataset

The results of metaheuristic-based optimizers with reducing algorithms are shown in Table 5.5. Furthermore, the best results of MDS (99.12 for QDPSO, 97.37 for PSO, and 97.37

for PSO bound), TSNE (98.25 for PSO, 97.37 for QDPSO, and 95.61 for PSO_bound), Isomap (99.12 for QDPSO, 98.25 for PSO, and 98.25 for PSO_bound), and PCA (98.25 for the metaheuristic-based optimizers) are in 15, 2, 20, and 10 dimensions, respectively.

Table 5.6 shows the result of gradient-based optimizers using the reducing algorithms. Furthermore, Isomap (98.25 for L-BFGS, 97.37 for SGD, and 96.49 for Adam), TSNE (98.25 for SGD and L-BFGS, and 97.37 for Adam), and PCA (98.25 for the gradient-based optimizers) have the best result in 5, 2, and 20 dimensions, respectively. In addition, it is not clear for MDS. Thus, Isomap has the best result and dimensions (with 5 dimensions) for PSO (99.12), SGD (97.37), and Adam (96.49); see tables 5.5 and 5.6.

5.2.3 Discussion of Dimension Reduction

Tables 5.3 and 5.4 describe TSNE as the best option to reduce dimensions for datasets with small characteristics, such as the iris dataset. Furthermore, in the metaheuristic-based and gradient-based classifiers, TSNE has the best result in the test phase compared with the other reducing algorithms proposed in this work. In addition, with 1 and 2 dimensions in the test stage, the PSO_bound, Adam, SGD, and L-BFGS have better results than their original features. On the other hand, in Tables 5.5 and 5.6, Isomap is better for reducing algorithms than MDS, TSNE, and PCA to datasets with significant characteristics, such as the breast dataset. Moreover, in the gradient-based optimizers, PCA has the best result with 5 dimensions. Nonetheless, Isomap is better for metaheuristic-based optimizers. Thus, in this work, we choose the Isomap reduction algorithm because it works well with the metaheuristic-based classifiers, but PCA is a good option too.

5.3 Performance of Classes and Samples

Table 5.7 illustrates the results of the classes and samples increasing with metaheuristic-based optimizers. Moreover, QDPSO is better than PSO and PSO_bound in 500 and 2000 samples with 4 classes. Further, QDPSO and PSO are better than PSO_bound in 2 and 3 classes with 500, 2000, and 10000 samples.

The results of the gradient-based optimizers with different classes and samples are described in Table 5.8. Besides, L-BFGS is better than Adam and SGD with 4 classes

Sample	C	Dataset	QDPSO		PSO		PSO_bound	
			MSE	ACC	MSE	ACC	MSE	ACC
500	2	test	**0.00**	**100.00**	**0.00**	**100.00**	**1.60E-01**	**84.00**
		train	0.00	100.00	0.00	100.00	1.85E-01	81.50
	3	test	**0.00**	**100.00**	**0.00**	**100.00**	5.20E-01	48.00
		train	0.00	100.00	0.00	100.00	4.43E-01	55.75
	4	test	**0.00**	**100.00**	2.00E-02	98.00	2.41E+00	36.00
		train	5.00E-03	99.50	1.00E-02	99.00	2.45E+00	34.00
2000	2	test	**0.00**	**100.00**	**0.00**	**100.00**	**4.75E-02**	**95.25**
		train	0.00	100.00	0.00	100.00	5.56E-02	94.44
	3	test	**0.00**	**100.00**	**0.00**	**100.00**	4.73E-01	52.75
		train	0.00	100.00	0.00	100.00	4.45E-01	55.50
	4	test	**0.00**	**100.00**	3.70E-01	95.00	2.67E+00	36.75
		train	1.25E-03	99.88	2.79E-01	95.56	2.47E+00	38.94
10000	2	test	**0.00**	**100.00**	**0.00**	**100.00**	**6.50E-02**	**93.50**
		train	0.00	100.00	0.00	100.00	6.00E-02	94.00
	3	test	**0.00**	**100.00**	**0.00**	**100.00**	5.55E-01	65.65
		train	0.00	100.00	0.00	100.00	5.76E-01	64.81
	4	test	7.50E-03	99.65	4.35E-02	99.25	2.52E+00	36.20
		train	3.88E-03	99.71	5.53E-02	98.98	2.55E+00	35.95

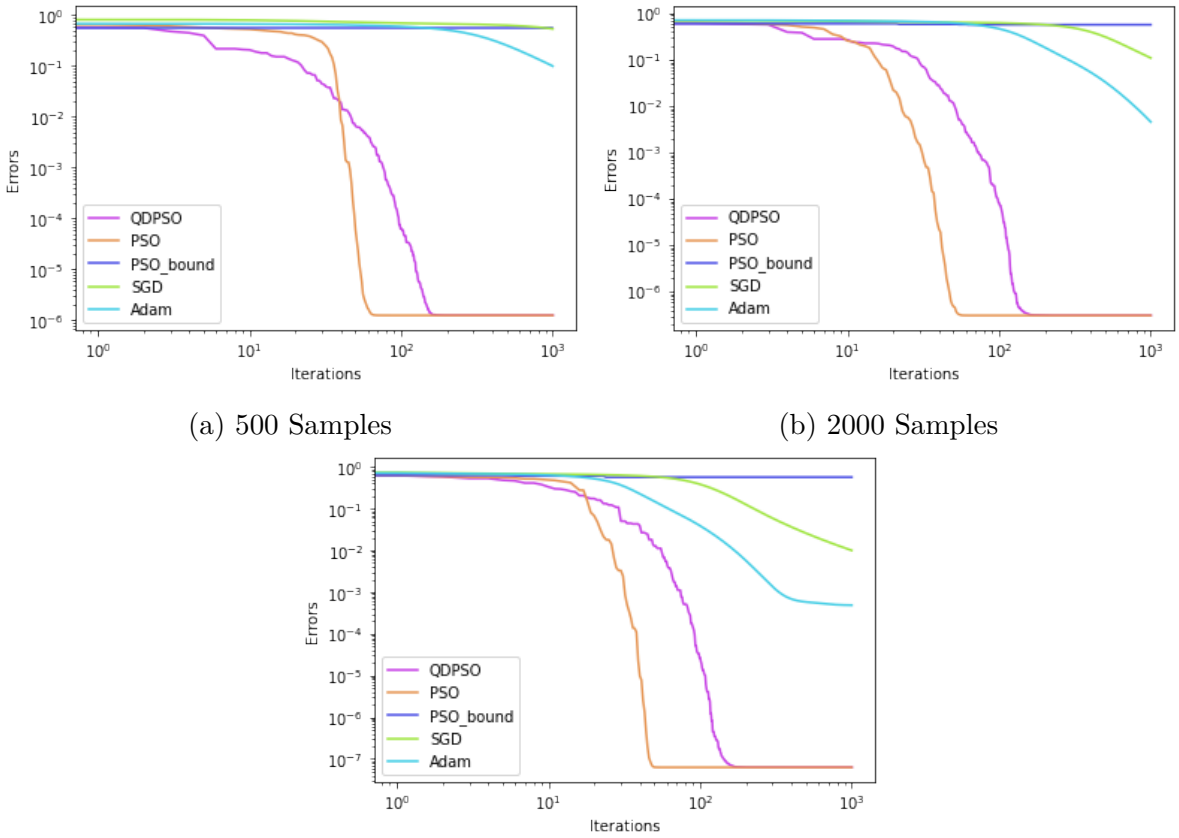
Table 5.7: The classes increasing results use the circle dataset and metaheuristic-based optimizers with 500, 2000, and 10000 random samples.

Sample	C	Dataset	Adam		SGD		L-BFGS	
			MSE	ACC	MSE	ACC	MSE	ACC
500	2	test	**0.00**	**100.00**	**1.60E-01**	**84.00**	**0.00**	**100.00**
		train	0.00	100.00	1.53E-01	84.75	0.00	100.00
	3	test	1.00E-02	99.00	4.60E-01	69.00	**0.00**	**100.00**
		train	1.00E-02	99.00	6.45E-01	64.00	0.00	100.00
	4	test	7.90E-01	69.00	1.95	44.00	**0.00**	**100.00**
		train	1.01	71.50	2.74	38.25	0.00	100.00
2000	2	test	**0.00**	**100.00**	**0.00**	**100.00**	**0.00**	**100.00**
		train	0.00	100.00	0.00	100.00	0.00	100.00
	3	test	**0.00**	**100.00**	9.50E-02	90.50	**0.00**	**100.00**
		train	6.25E-04	99.94	9.25E-02	90.75	0.00	100.00
	4	test	6.08E-01	91.25	1.08	66.50	**0.00**	**100.00**
		train	8.51E-01	89.38	1.10	69.25	0.00	100.00
10000	2	test	**0.00**	**100.00**	**0.00**	**100.00**	**0.00**	**100.00**
		train	0.00	100.00	0.00	100.00	0.00	100.00
	3	test	**0.00**	**100.00**	**0.00**	**100.00**	**0.00**	**100.00**
		train	0.00	100.00	0.00	100.00	0.00	100.00
	4	test	**0.00**	**100.00**	3.94E-01	95.45	**0.00**	**100.00**
		train	1.25E-04	99.99	3.83E-01	95.56	0.00	100.00

Table 5.8: The classes increasing results use the circle dataset and gradient-based optimizers with 500, 2000, and 10000 random samples.

and 500 and 2000 samples. Moreover, L-BFGS and Adam are better than SGD in 2 and 3 classes with 500 samples, 3 classes with 2000 samples, and 4 classes with 10000 samples. On the other hand, Tables 5.7 and 5.8 show that L-BFGS and QDPSO are better than PSO, PSO_bound, Adam, and SGD with 500 and 2000 samples and 4 classes. Furthermore, L-BFGS and Adam are better than QDPSO, PSO, PSO_bound, and SGD with 10000 samples and 4 classes.

5.3.1 The loss curves of classes and samples



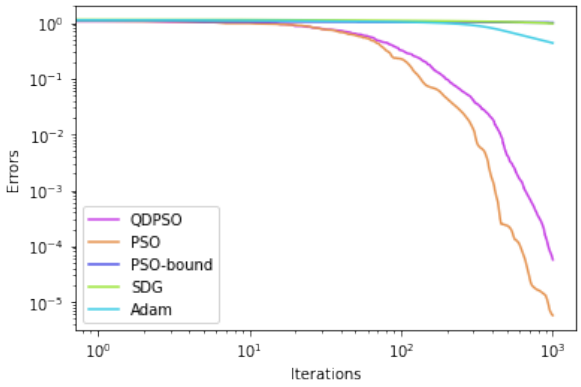
(a) 500 Samples

(b) 2000 Samples

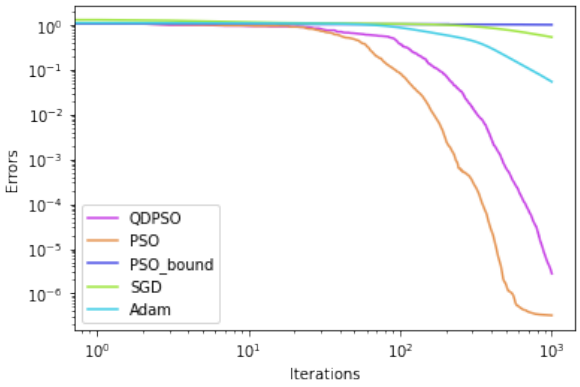
(c) 10000 Samples

Figure 5.8: The loss curve of the circle dataset uses metaheuristic-based and gradient-based optimizers with normalization, 2 classes, and 1000 iterations. Furthermore, (a) shows the loss curves with 500 samples, (b) with 2000 samples, and (c) with 10000 samples.

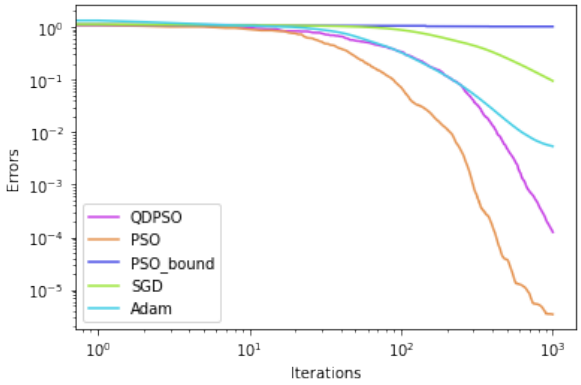
Figures 5.8, 5.9, and 5.10 illustrate the convergence and behavior of the loss cost during the best training. Moreover, they use normalization and 1000 iterations for metaheuristic-based and gradient-based optimizers with 2, 3, and 4 classes. Where sub-figure (a) uses 500 samples. Sub-figure (b) utilizes 2000 samples, and sub-figure (c) uses of 10000 samples.



(a) 500 Samples



(b) 2000 Samples



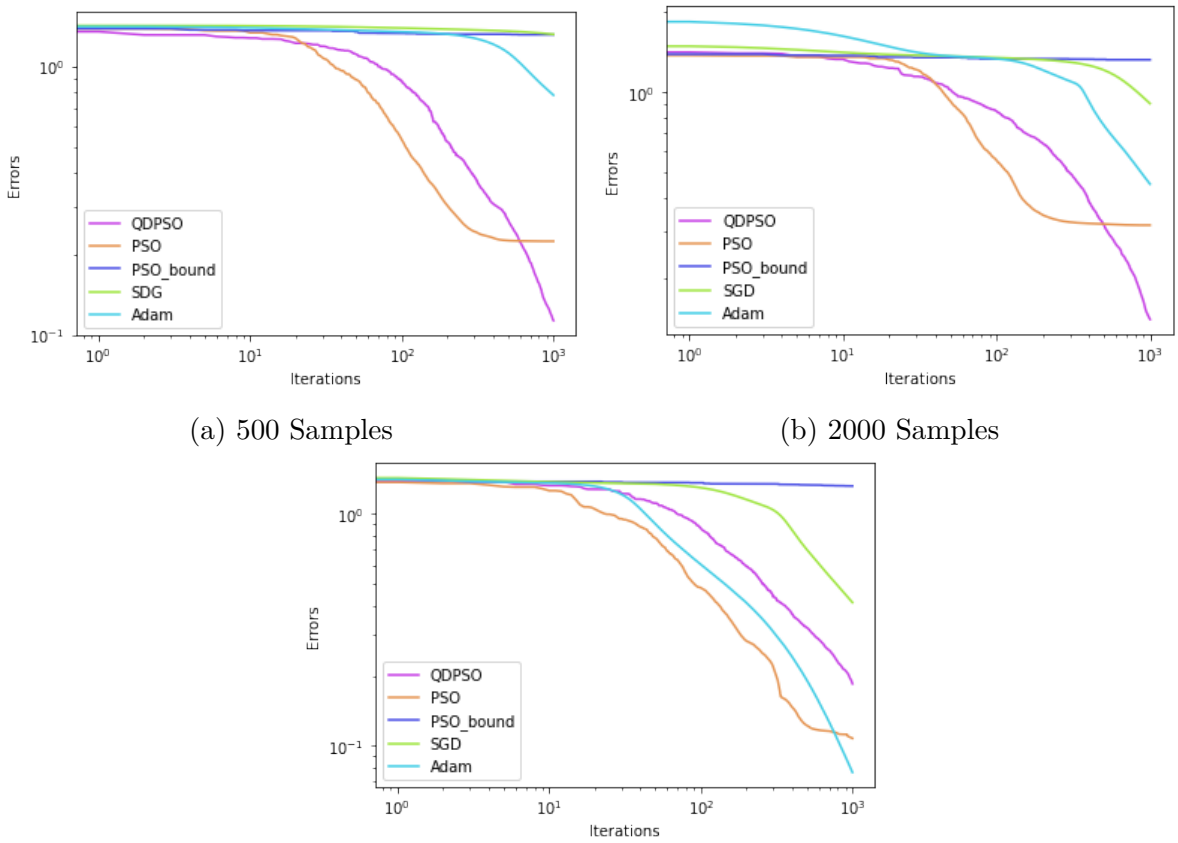
(c) 10000 Samples

Figure 5.9: The loss curve of the circle dataset uses metaheuristic-based and gradient-based optimizers with normalization, 3 classes, and 1000 iterations. Furthermore, (a) shows the loss curves with 500 samples, (b) with 2000 samples, and (c) with 10000 samples.

Finally, L-BFGS could not make a graph because the MLPClassifier framework does not allow it.

In 2 classes, the PSO curve has a faster convergence than QDPSO, Adam, SGD, and PSO_bound curves with 500, 2000, and 1000 samples; see Figure 5.8. However, the QDPSO curve has a better smoothness than Adam, PSO, SGD, and PSO_bound curves, as shown in Figures 5.8a, 5.8b, and 5.8c. On the other hand, the PSO_bound curve is worse in convergence and behavior than the classifiers proposed in this work. Furthermore, it stalls out in search during the training in the 3 cases. In the same way, the QDPSO and PSO curves stall in the last iterations for all cases.

Figure 5.9 shows that the PSO curve converges better than QDPSO, Adam, SGD, and PSO_bound curves with 3 classes and 500, 2000, and 10000 samples. Furthermore, the QDPSO and PSO curves are smoothness than Adam, SGD, and PSO_bound curves during



(a) 500 Samples

(b) 2000 Samples

(c) 10000 Samples

Figure 5.10: The loss curve of the circle dataset uses metaheuristic-based and gradient-based optimizers with normalization, 4 classes, and 1000 iterations. Furthermore, (a) shows the loss curves with 500 samples, (b) with 2000 samples, and (c) with 10000 samples.

the training. Thus, QDPSO and PSO curves have the best behavior; see Figures 5.9a, 5.9b, and 5.9c. On the other hand, the PSO_bound curves is worse behavior and convergence than all classifiers proposed in this work. In addition, with 2000 samples, the PSO curve tends to stagnate in the last iterations, as shown in Figure 5.9b.

With 4 classes, the QDPSO curve converges better than PSO, Adam, SGD, and PSO_bound curves during the training using 500 and 2000 samples; see Figures 5.10a and 5.10b. Moreover, the Adam curve converges better than QDPSO, PSO, SGD, and PSO_bound curves with 1000 samples, as shown in figure 5.10c. In addition, Figure 5.10 depicts that the QDPSO curve has the best behavior in the 3 cases. Conversely, the PSO_bound curve is worse in convergence and behavior than all classifiers described in this project for all cases. Besides, the PSO curve stalled in the last iterations; see Figure 5.10.

5.3.2 Discussion of Increasing Classes and Samples

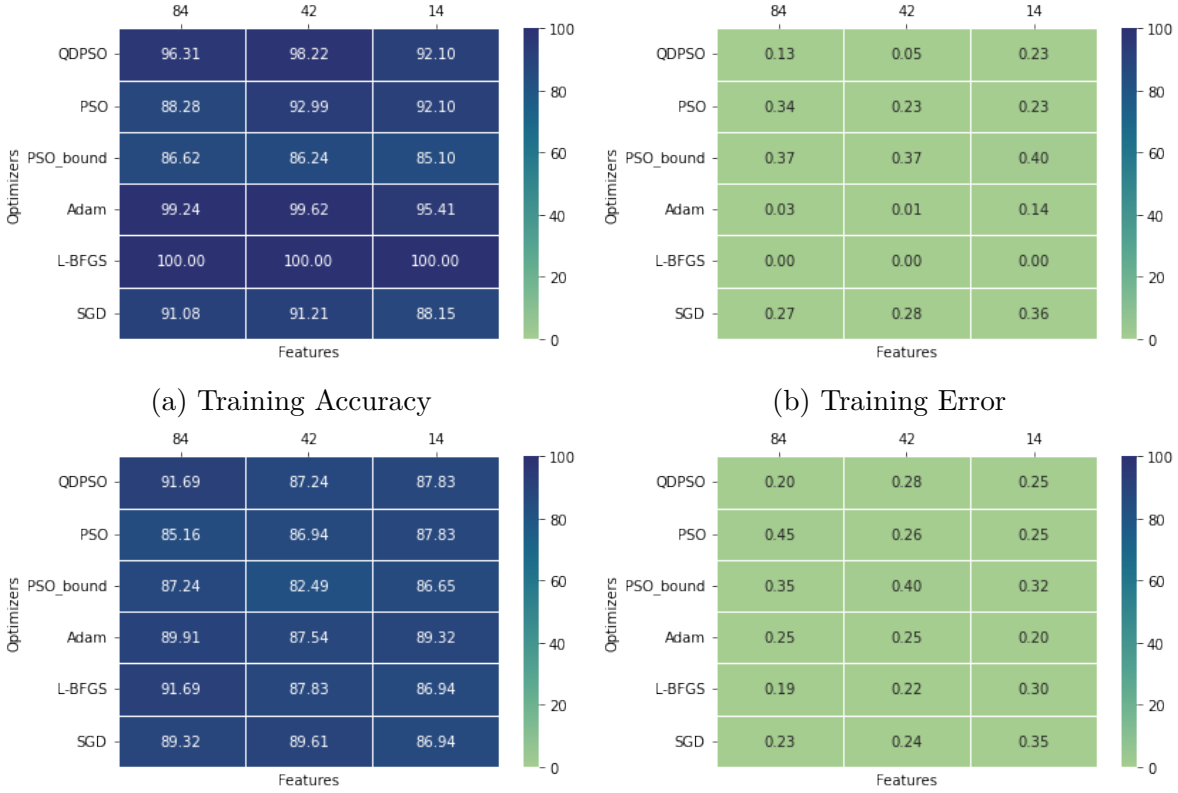
The results illustrated in Tables 5.7 and 5.8 are coherent with the loss cost graphics described in Figures 5.8, 5.9, and 5.10. Furthermore, The QDPSO classifier is the best for the metaheuristics-based optimizers, while L-BFGS is for the gradient-based optimizers in all cases, as shown in Tables 5.7 and 5.8. On the other hand, the QDPSO curve has the best smoothness in the 3 cases with different samples; see Figures 5.8, 5.9, and 5.10. Then, with 3 classes, Figure 5.9 depicts that the PSO curve has a smooth curve using 500, 2000, and 300 samples. Finally, with 2 and 4 classes, QDPSO has a faster convergence using 500, 2000, and 10000. Besides, with different classes, PSO convergences quickly. However, it was stagnant in the last iterations; see Figures 5.8, 5.9b, and 5.10. Similarly, Figure 5.8 shows QDPSO stalls with different samples and 2 classes. In addition, Adam has the best convergence with 4 classes and 10000 samples, as shown in Figure 5.10c. Then, PSO_bound was the worse classifier in all cases and optimizers.

5.4 Performance of Image Classification

This section describes the image classification problem with the proposed model. Figure 5.11 shows the experimental results of the MCW dataset using different features (we use Isomap to reduce the characteristics of the MCW dataset and data normalization) and optimizers. Furthermore, it shows the mean square error and accuracy of the experiments. Finally, we analyze the image classification results.

5.4.1 The result of the metaheuristic-based optimizers

In the training phases with 84 and 42 features, QDPSO (96.31 and 98.22) is better than PSO (88.28 and 92.99) and PSO_bound (86.62 and 86.24); see Figures 5.11a and 5.11b. However, with 14 features, QDPSO and PSO are better than PSO_bound; 92.10, 92.10, and 85.10, respectively. Moreover, with 14 features in the testing stage, QDPSO and PSO are better than PSO_bound; correspondingly, 87.83, 87.83, and 86.65, see Figures 5.11c and 5.11d. Nonetheless, with 84 and 42 characteristics, QDPSO (91.69 and 87.24) is better than PSO (85.16 and 86.94) and PSO_bound (87.24 and 82.49).



(a) Training Accuracy

(b) Training Error

(c) Testing Accuracy

(d) Testing Error

Figure 5.11: Heatmaps depicting the performance of each optimizer with 84, 42, and 14 features using 1000 iterations and MCW dataset. Where (a) shows the accuracy of training in percent, (b) illustrates the mean squared error during the training, (c) sees the testing accuracy in percent, and (d) represents the mean squared error during the testing.

5.4.2 The result of the gradient-based optimizers

In the training phase, with 84, 42, and 14 characteristics, L-BFGS (100, 100, and 100) is better than Adam (99.24, 99.62, and 95.41) and SGD (91.08, 91.21, and 88.15); see Figures 5.11a and 5.11b. Furthermore, in the testing stage with 84 features, L-BFGS is better than Adam and SGD; correspondingly, 91.69, 89.91, and 89.32, see Figures 5.11c and 5.11d. However, SGD is better than L-BFGS and Adam, with 42 characteristics; 89.61, 87,83, and 87.54, appropriately. Then, with 14 features, Adam is better than L-BFGS and SGD; 89.32, 86.94, and 86.94, respectively.

5.4.3 The result of the metaheuristic-based and gradient-based optimizers

In the training phase, with 84, 42, and 14 features, L-BFGS (100, 100, and 100) is better than Adam (99.24, 99.62, and 95.41), QDPSO (96.31, 98.22, and 92.10), PSO (88.28, 92.99, and 92.10), SGD (91.08, 91.21, and 88.15), and PSO_bound (86.62, 86.24, and 85.10); see Figure 5.11. Furthermore, QDPSO (91.69, 87.24, and 87.83) and L-BFGS (91.69, 87.83, and 86.94) are better than Adam (89.91, 87.54, and 89.32), SGD (89.32, 89.61, and 86.94), PSO (85.16, 86.94, and 87.83), and PSO_bound (87.24, 82.49, and 86.65) in the testing stage with 84, 42, and 14 characteristics, respectively.

5.4.4 The metrics of the metaheuristic-based optimizers

Table 5.9 describes the best, average, worst, and deviation standards of the loss costs of the MCW dataset. Moreover, we use normalization, 10 training, and 1000 iterations with different characteristics and optimizers.

Optimizers	Features	Worst	Avg	Best	Std
QDPSO	84	5.49E-01	2.64E-01	9.91E-02	1.49E-01
	42	1.30E-01	6.56E-02	4.78E-02	2.24E-02
	14	1.41E-01	1.28E-01	1.15E-01	7.76E-03
PSO	84	**1.37**	1.11	3.27E-01	2.89E-01
	42	2.49E-01	2.14E-01	1.73E-01	2.19E-02
	14	2.70E-01	2.54E-01	2.23E-01	1.32E-02
PSO_bound	84	6.13E-01	5.29E-01	4.17E-01	6.07E-02
	42	**4.80E-01**	4.14E-01	3.86E-01	2.54E-02
	14	**6.99E-01**	5.00E-01	4.30E-01	8.13E-02
Adam	84	3.97E-02	3.86E-02	3.72E-02	6.97E-04
	42	1.79E-02	1.66E-02	1.56E-02	6.99E-04
	14	1.51E-01	1.40E-01	1.32E-01	6.94E-03
L-BFGS	84	3.94E-03	**2.88E-03**	**2.46E-03**	**4.31E-04**
	42	1.77E-03	**1.55E-03**	**1.34E-03**	**1.26E-04**
	14	2.87E-03	**2.29E-03**	**1.95E-03**	**2.54E-04**
SGD	84	2.84E-01	2.81E-01	2.78E-01	1.85E-03
	42	2.67E-01	2.64E-01	2.60E-01	2.18E-03
	14	3.25E-01	3.23E-01	3.21E-01	1.41E-03

Table 5.9: The best, worst, average, and standard deviation of the loss cost of the MCW dataset for the metaheuristic-based and gradient-based optimizers.

Table 5.9 shows that QDPSO is better than PSO and PSO_bound in the best and

average loss costs with 84, 42, and 14 features. Furthermore, in the deviation standard of the loss cost, QDPSO (2.24E-02 and 7.76E-03) is better than PSO (2.19E-02 and 1.32E-02) and PSO_bound (2.54E-02 and 8.13E-02) with 42 and 14 characteristics. However, with 84 features, the PSO_bound is better than QDPSO and PSO; correspondingly, 6.07E-02, 1.49E-01, and 2.89E-01. On the other hand, in the worst loss cost, PSO is worse than PSO_bound and QDPSO, with 84 features; 1.37, 6.13E-01, and 5,49E-01, appropriately. Besides, with 42 and 14 characteristics, PSO_bound (4.80E-01 and 6.99E-01) is worse than PSO (2.49E-01 and 2.70E-01) and QDPSO (1.30E-01 and 1.41E-01).

5.4.5 The metrics of the gradient-based optimizers

L-BFGS is better than Adam and SGD in the best and average loss costs with 84, 42, and 14 features; see Table 5.9. Moreover, L-BFGS (4.31E-04, 1.26E-04, and 2.54E-04) has the best deviation standard of loss cost. Further, the worst loss cost, SGD (2.84E-01, 2.67E-01, and 3.25E-01) is worse than Adam (3.97E-02, 1.79E-02, and 1.51E-01) and L-BFGS (3.94E-03, 1.77E-03, and 2.87E-03) with 84, 42, and 14 characteristics.

5.4.6 The metrics of the metaheuristic-based and gradient-based optimizers

L-BFGS is better than Adam, SGD, QDPSO, PSO, and PSO_bound in the best and average loss costs with 84, 42, and 14 features; see Table 5.9. Moreover, L-BFGS (4.31E-04, 1.26E-04, and 2.54E-04) carries the best deviation standard of loss cost. Conversely, in the worst loss cost, PSO is worse than PSO_bound, QDPSO, SGD, Adam, and L-BFGS with 84 characteristics; 1.37, 6.13E-01, 5.49E-01, 2.84E-01, 3.97E-02, and 3.94E-03, respectively. Besides, with 42 and 14 features, PSO_bound (4.80E-01 and 6.99E-01) is worse than PSO (2.49E-01 and 2.70E-01), SGD (2.67E-01 and 3.25E-01), QDPSO (1.30E-01 and 1.41E-01), Adam (1.79E-02 and 1.51E-01), and L-BFGS (1.77E-03 and 2.87E-03).

Table 5.10 describes the precision, recall, and f1 score of the MCW dataset for the metaheuristic-based and gradient-based optimizers. Furthermore, we use the best training, 1000 iterations, and dataset normalization. On the other hand, the precision score with 84, 42, and 14 features are 9.19E-01, 8.74E-01, and 8.97E-01 for QDPSO, 8.56E-01, 8.73E-01,

Optimizers	Features	Presicion	Recall	F1 score
QDPSO	84	9.19E-01	9.17E-01	9.17E-01
	42	8.74E-01	8.72E-01	8.73E-01
	14	8.97E-01	8.93E-01	8.94E-01
PSO	84	8.56E-01	8.52E-01	8.51E-01
	42	8.73E-01	8.69E-01	8.70E-01
	14	8.84E-01	8.78E-01	8.79E-01
PSO_bound	84	8.56E-01	8.52E-01	8.51E-01
	42	8.73E-01	8.69E-01	8.70E-01
	14	8.84E-01	8.78E-01	8.79E-01
Adam	84	9.02E-01	8.99E-01	9.00E-01
	42	8.79E-01	8.75E-01	8.76E-01
	14	8.96E-01	8.93E-01	8.93E-01
L-BFGS	84	9.18E-01	9.17E-01	9.17E-01
	42	8.81E-01	8.78E-01	8.79E-01
	14	8.75E-01	8.69E-01	8.71E-01
SGD	84	8.96E-01	8.93E-01	8.94E-01
	42	9.00E-01	8.96E-01	8.97E-01
	14	8.73E-01	8.69E-01	8.70E-01

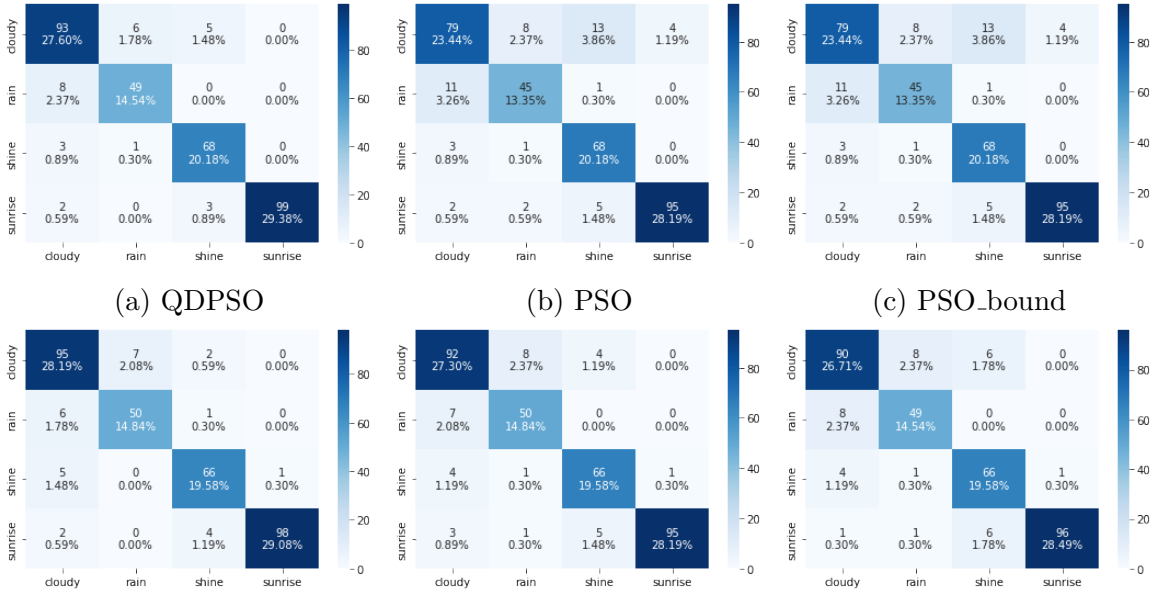
Table 5.10: The precision, recall, and f1 score of the MCW dataset for the metaheuristic-based and gradient-based optimizers.

and 8.84E-01 for PSO, 8.56E-01, 8.73E-01, and 8.84E-01 for PSO_bound, 9.02E-01, 8.79E-01, and 8.96E-01 for Adam, 9.18E-01, 8.81E-01, and 8.75E-01 for L-BFGS, and 8.96E-01, 9.00E-01, and 8.73E-01 for the SGD. Besides, the recall score is 9.17E-01, 8.72E-01, and 8.93E-01 for QDPSO, 8.52E-01, 8.69E-01, and 8.78E-01 for PSO, 8.52E-01, 8.69E-01, and 8.78E-01 PSO_bound, 8.99E-01, 8.75E-01, and 8.93E-01 for Adam, 9.17E-01, 8.78E-01, and 8.69E-01 for L-BFGS, and 8.93E-01, 8.96E-01, and 8.69E-01 for SGD. Finally, the f1 score is 9.17E-01, 8.73E-01, and 8.94E-01 for QDPSO, 8.51E-01, 8.70E-01, and 8.79E-01 for PSO, 8.51E-01, 8.70E-01, and 8.79E-01 for PSO_bound, 9.00E-01, 8.76E-01, and 8.93E-01 for Adam, 9.17E-01, 8.79E-01, and 8.71E-01 for L-BFGS, and 8.94E-01, 8.97E-01, and 8.70E-01 for SGD.

5.4.7 Confusion Matrix and Loss Cost Curve

Figures 5.12, 5.13, and 5.14 describe the confusion matrix of the MCW dataset with 84, 42, and 14 features, respectively. Moreover, Figures a, b, and c illustrate the metaheuristic-based optimizers, while Figures d, e, and f are the gradient-based optimizers. Further, the

confusion matrix uses the best training, dataset normalized, and 1000 iterations.



(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

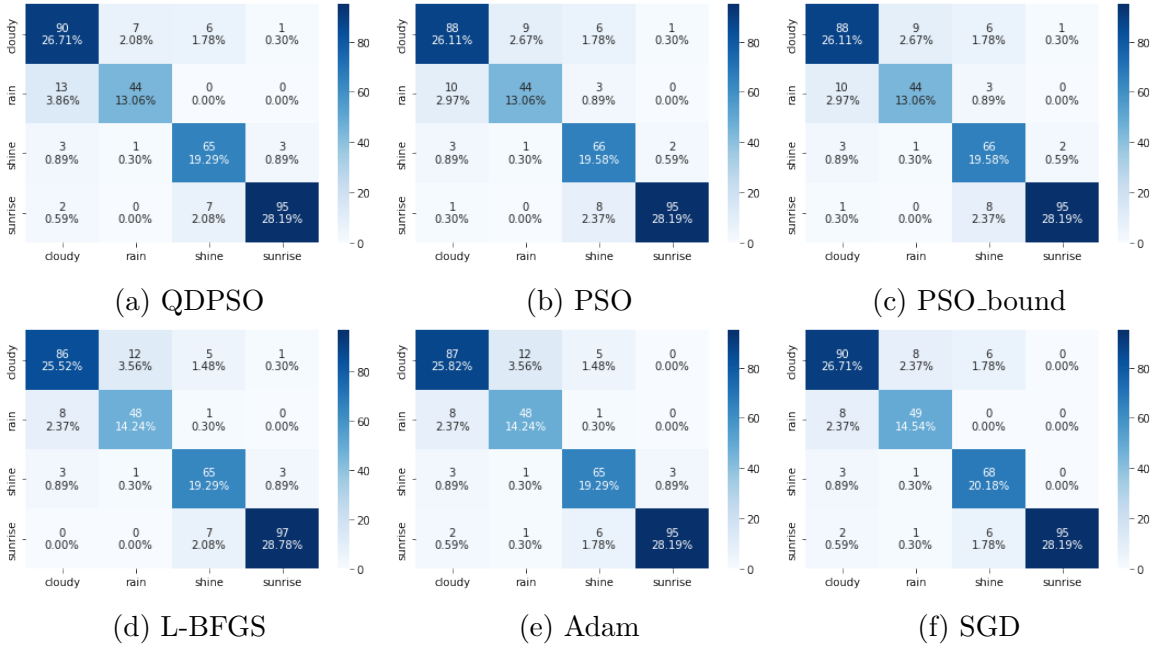
Figure 5.12: The confusion matrix of the MCW dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization, 84 features, and 1000 iterations.

The precision of the QDPSO classifier with 84 features is 91.7%, divided into 27.60% for the cloudy class, 14.54% for the rain class, 20.18% for the shine class, and 29.38% for the sunrise class, as shown in Figure 5.12a. Thus, 8.3% corresponds to the classification errors, where 1.78% think the cloudy class is the rain class and 1.48% is the shine class. In addition, 2.37% say the rain class is the cloudy class. Besides, 0.89% assume the shine class is the cloudy class, and 0.30% is the rain class. Then, 0.59% believe that the sunrise class is the cloudy class, and 0.89% is the shine class. On the other hand, with 42 characteristics, the precision is 87.25%, distributed in 26.71% for the cloudy class, 13.06% for the rain class, 19.29% for the shine class, and 28.19% for the sunrise class, see Figure 5.13a. Thus, 12.75% corresponds to the classification error, where 2.08% assumes the cloudy class is the rain class, 1.78% is the shine class, and 0.30% is the sunrise class. Then, 3.86% think the rain class is the cloudy class. Furthermore, 0.89% say the shine class is the cloudy class, 0.30% is the rain class, and 0.89% is the sunrise class. Besides, 0.59% believe that the sunrise class is the cloudy class, and 2.08% is the shine class. Finally, with 14 features, the precision is 89.32%, divided into 25.82% for the cloudy class, 13.95% for the rain class,

20.47% for the shine class, and 29.08% for the sunrise class, as shown in Figure 5.14a. Thus, 10.68% is the classification error, where 2.97% assumes the cloudy class is the rain class, and 2.08% is the shine class. Further, 2.37% say the rain class is the cloudy class, and 0.59% is the shine class. Then, 0.59% believe that the shine class is the cloudy class, and 0.30% is the rain class. In addition, 0.30% think the sunrise class is the cloudy class, 0.30% is the rain class, and 1.19% is the shine class.

The main diagonal of Figure 5.12b shows the precision of the PSO (85.16%) classifier with 84 characteristics, which is distributed into 23.44% for the cloudy class, 13.35% for the rain class, 20.18% for the shine class, and 28.19% for the sunrise class. Thus, 14.84% is the classification error, where 2.37% determines that the cloudy class is the rain class, 3.86% is the shine class, and 1.19% is the sunrise class. Moreover, 3.26% assume the rain class is the cloudy class, and 0.30% is the shine class. So then, 0.89% think the shine class is the cloudy class, and 0.30% is the rain class. Besides, 0.59% believe that the sunrise class is the cloudy class, 0.59% is the rain class, and 1.48% is the shine class. Conversely, with 42 features, the precision is 86.94%, divided into 26.11% for the cloudy class, 13.06% for the rain class, 19.58% for the shine class, and 28.19% for the sunrise class, see Figure 5.13b. Thus, 13.06% corresponds to the classification errors, where 2.67% think the cloudy class is the rain class, 1.78% is the shine class, and 0.30% is the sunrise class. Besides, 2.97% say the rain class is the cloudy class, and 0.89% is the shine class. So then, 0.89% consider the shine class the cloudy class, 0.30% for the rain class, and 0.59% for the sunrise class. Further, 0.30% believe that the sunrise class is the cloudy class, and 2.37% is the shine class. Finally, with 14 characteristics, the precision is 87.84%, split into 25.82% for the cloudy class, 13.95% for the rain class, 20.18% for the shine class, and 27.89% for the sunrise class, as shown in Figure 5.14b. Thus, 12.16% is the classification error, 2.97% think the cloudy class is the rain class, 1.78% is the shine class, and 0.30% is the sunrise class. Moreover, 2.67% believe that the rain class is the cloudy class, and 0.30% is the shine class. So then, 0.59% say the shine class is the cloudy class, and 0.59% is the rain class. In addition, 0.30% assume that the sunrise class is the cloudy class, 0.30% is the rain class, and 2.37% is the shine class.

With 84 features, the PSO_bound precision is 85.16%, divided into 23.44% for the cloudy class, 13.35% for the rain class, 20.18% for the shine class, and 28.19% for the



(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

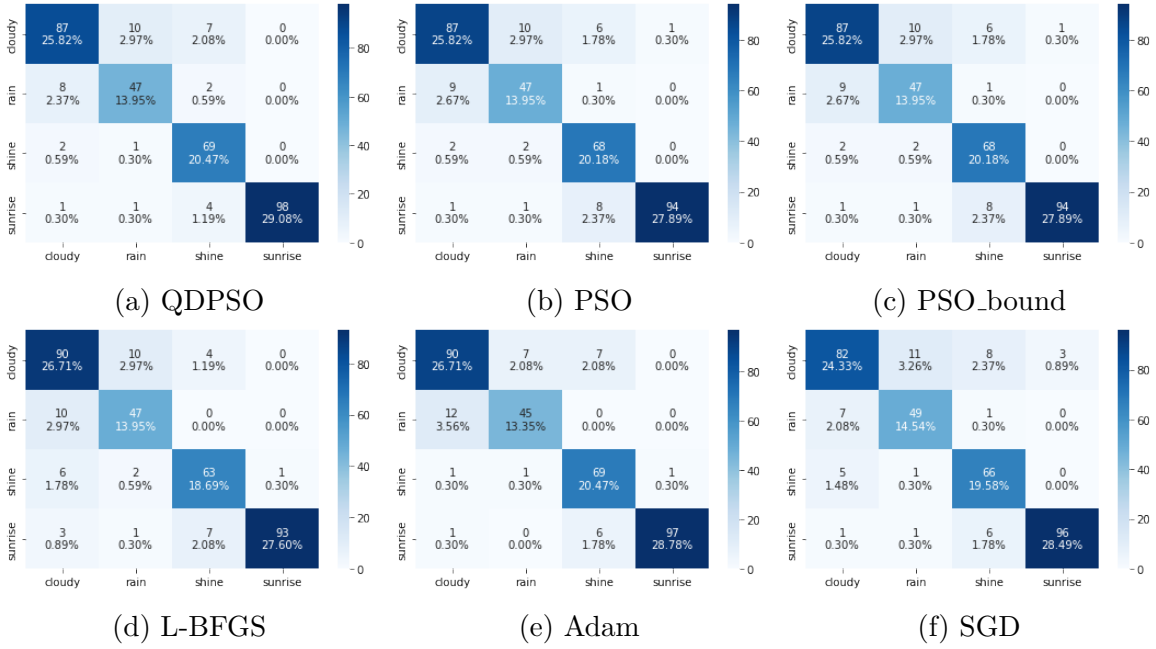
Figure 5.13: The confusion matrix of the MCW dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization, 42 features, and 1000 iterations.

sunrise class, as shown in Figure 5.12c. Thus, 14.84% corresponds to the classification error, where 2.37% consider the cloudy class as the rain class, 3.86% is the shine class, and 1.19% is the sunrise class. In addition, 3.26% think the rain class is the cloudy class, and 0.30% is the shine class. So then, 0.89% say the shine class is the cloudy class, and 0.30% is the rain class. Besides, 0.59% believe that the sunrise class is the cloudy class, 0.59% is the rain class, and 1.48% is the shine class. On the other hand, with 42 characteristics, the precision is 86.94%, divided into 26.11% for the cloudy class, 13.06% for the rain class, 19.58% for the shine class, and 28.19% for the sunrise class, see Figure 5.13c. Thus, 13.06% corresponds to the classification errors, where 2.67% believe that the cloudy class is the rain class, 1.78% is the shine class, and 0.30% is the sunrise class. Further, 2.97% assume the rain class is cloudy, and 0.89% is shiny. So then, 0.89% consider the shine class the cloudy class, 0.30% for the rain class, and 0.59% for the sunrise class. Besides, 0.30% say the sunrise class is the cloudy class, and 2.37% is the shine class. Finally, with 14 characteristics, the precision is 87.84%, distributed into 25.82% for the cloudy class, 13.95% for the rain class, 20.18% for the shine class, and 27.89% for the sunrise class, as shown in Figure 5.14c. Thus, 12.16% is the classification error, 2.97% think the cloudy

class is the rain class, 1.78% is the shine class, and 0.30% is the sunrise class. In addition, 2.67% consider the rain class as the cloudy class and 0.30% as the shine class. So then, 0.59% think the shine class is the cloudy class, and 0.59% is the rain class. Moreover, 0.30% believe that the sunrise class is the cloudy class, 0.30% is the rain class, and 2.37% is the shine class.

The precision of the L-BFGS classifier with 84 features is 91.69%, split into 28.19% for the cloudy class, 14.84% for the rain class, 19.58% for the shine class, and 29.08% for the sunrise class; see Figure 5.12d. Thus the classification error is 8.31%, where 2.08% say the cloudy class is the rain class, and 0.59% is the shine class. Besides, 1.78% think the rain class is the cloudy class, and 0.30% is the shine class. Furthermore, 1.48% assume the shine class is the cloudy class, and 0.30% is the sunrise class. So then, 0.59% believe that the sunrise class is cloudy, and 1.19% is shining. Conversely, with 42 characteristics, the precision is 87.83%, divided into 25.52% for the cloudy class, 14.24% for the rain class, 19.29% for the shine class, and 28.78% for the sunrise class; as shown in Figure 5.13d. Thus, 12.17% correlates with the classification error, where 3.56% estimate the cloudy class is the rain class, 1.48% is the chine class, and 0.30% is the sunrise class. Moreover, 2.37% infer that the rain class is the cloudy class, and 0.30% is the shine class. So then, 0.89% say the shine class is the cloudy class, 0.30% is the rain class, and 0.89% is the sunrise class. Further, 2.08% think the sunrise class is the shine class. Finally, with 14 features, the precision is 86.95%, distributed in 26.71% for the cloudy class, 13.95% for the rain class, 18.69% for the shine class, and 27.60% for the sunrise class; see Figure 5.14d. Thus, the classification error is 13.05%, where 2.97% assumes the cloudy class is the rain class, and 1.19% is the shine class. Furthermore, 2.97% infer that the rain class is the cloudy class. In addition, 1.78% say the shine class is the cloudy class, 0.59% is the rain class, and 0.30% is the sunrise class. Moreover, 0.89% believe that the sunrise class is the cloudy class, 0.30% is the rain class, and 2.08% is the shine class.

With the 84 characteristics, the Adam precision is 89.91%, divided into 27.30% for the cloudy class, 14.84% for the rain class, 19.58% for the shine class, and 28.19% for the sunrise class, as shown in Figure 5.12e. Thus, 10.09% corresponds to the classification error, where 2.37% infer that the cloudy class is the rain class, and 1.19% is the shine class. In addition, 2.08% say the rain class is the cloudy class. Furthermore, 1.19% assume



(a) QDPSO (b) PSO (c) PSO_bound

(d) L-BFGS (e) Adam (f) SGD

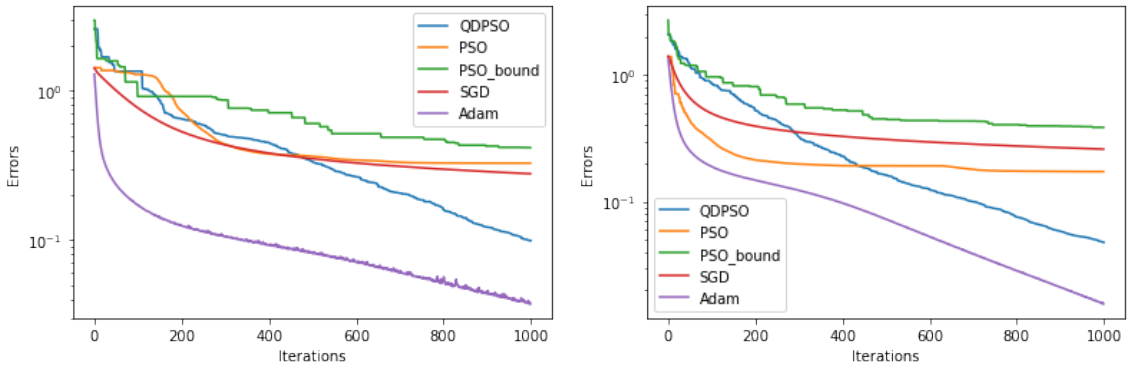
Figure 5.14: The confusion matrix of the MCW dataset uses metaheuristic-based (a, b, and c) and gradient-based (d, e, and f) optimizers with normalization, 14 features, and 1000 iterations.

the shine class is the cloudy class, 0.30% is the rain class, and 0.30% is the sunrise class. So then, 0.89% think the sunrise class is the cloudy class, 0.30% is the rain class, and 1.48% is the shine class. On the other hand, with 42 features, the precision is 87.54%, split into 25.82% for the cloudy class, 14.24% for the rain class, 19.29% for the shine class, and 28.19% for the sunrise class; see Figure 5.13e. Thus, the classification error is 12.46%, where 3.56% say the cloudy class is the rain class, and 1.48% is the shine class. Moreover, 2.37% assume that the rain class is the cloudy class, and 0.30% is the shine class. Besides, 0.89% believe that the shine class is the cloudy class, 0.30% is the rain class, and 0.89% is the sunrise class. So then, 0.59% think the sunrise class is the cloudy class, 0.30% is the rain class, and 1.78% is the shine class. Finally, with 14 characteristics, the precision is 89.31%, distributed into 26.71% for the cloudy class, 13.35% for the rain class, 20.47% for the shine class, and 28.78% for the sunrise class, as shown in Figure 5.14e. Thus, 10.69% is the classification error, where 2.08% infer that the cloudy class is the rain class, and 2.08% is the shine class. Further, 3.56% say the rain class is the cloudy class. Besides, 0.30% think the shine class is the cloudy class, 0.30% is the rain class, and 0.30% is the sunrise class. In addition, 0.30% believe that the sunrise class is the cloudy class, and 1.78% is

the shine class.

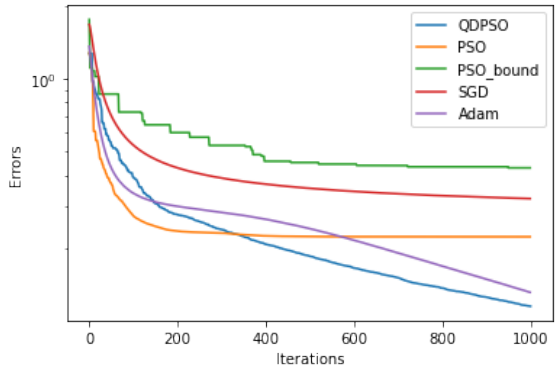
In Figure 16, the SGD classifier has 89.32% precision with 84 features, split into 26.71% for the cloudy class, 14.54% for the rain class, 19.58% for the shine class, and 28.49% for the sunrise class; see Figure 5.12f. Thus, 10.68% correlates to the error classification, where 2.37% think the cloudy class is the rain class, and 1.78% is the shine class. Moreover, 2.08% say the rain class is the cloudy class. In addition, 1.19% believe that the shine class is the cloudy class, 0.30% is the rain class, and 0.30% is the sunrise class. So then, 0.30% assume the sunrise class is the cloudy class, 30% is the rain class, and 1.78% is the shine class. Conversely, with 42 characteristics, the precision is 89.62%, divided into 26.71% for the cloudy class, 14.54% for the rain class, 20.18% for the shine class, and 28.19% for the sunrise class; as shown in Figure 5.13f. Thus, the classification error is 10.38%, where 2.37% say the cloudy class is the rain class, and 1.78% is the shine class. Besides, 2.37% think the rain class is the cloudy class. Furthermore, 0.89% assume the shine class is the cloudy class, and 0.30% is the rain class. In addition, 0.59% believe that the sunrise class is the cloudy class, 0.30% is the rain class, and 1.78% is the shine class. Finally, with 14 features, the precision is 86.94%, decided into 24.33% for the cloudy class, 14.54% for the rain class, 19.58% for the shine class, and 28.49% for the sunrise class; see Figure 5.14f. Thus, 13.06 corresponds to the classification error, where 3.26% assumes the cloudy class is the rain class, 2.37% is the shine class, and 0.89% is the sunrise class. Further, 2.08% say the rain class is the cloudy class, and 0.30% is the shine class. So then, 1.48% think the shine class is the cloudy class, and 0.30% is the rain class. Moreover, 0.30% believe that the sunrise class is the cloudy class, 0.30% is the rain class, and 1.78% is the shine class.

Figure 5.15 describes the convergence of the loss cost for the optimizers proposed in this work. Furthermore, we use the best training, 1000 iterations, and normalization of the MCW dataset with 84, 42, and 14 characteristics. Unfortunately, L-BFGS could not plot because the MLPClassifier framework does not allow it. In metaheuristic-based optimizers, Figures 5.15a, 5.15b, and 5.15c show that the QDPSO curve is better than the PSO and PSO_bound curves. Furthermore, the Adam curve is better than the SGD curve in the gradient-based optimizers. Conversely, in the metaheuristic-based and gradient-based optimizers with 84 and 42 features, the Adam curve is better than QDPSO, PSO,



(a) 84 Features

(b) 42 Features



(c) 14 Features

Figure 5.15: The loss curve of the MCW dataset uses metaheuristic-based and gradient-based optimizers with normalization, different characteristics, and 1000 iterations.

SGD, and PSO_bound curves. However, with 14 features, the QDPSO curve is better than Adam, PSO, SGD, and PSO_bound.

5.4.8 Discussion of Image Classification

The f1 score and deviation standard in Tables 5.10 and 5.9 show that the metaheuristic-based and gradient-based classifiers handle the MCW dataset. Thus, Figure 5.11 shows that QDPSO and L-BFGS are better results than PSO, PSO_bound, Adam, and SGD with 84 features. Moreover, Adam has the best results in the test phase with 14 characteristics (features proposed). Nevertheless, QDPSO and PSO have results virtually close to the best. Besides, the deviation standard reduces considerably with the characteristic proposed in this work, as shown in Table 5.9. Conversely, L-BFGS has the best results in the training phases. However, QDPSO has the best results in testing, see Figure 5.11.

In Figure 5.15, the Adam curve has the best behavior and convergence than the opti-

mizers proposed in this work with 84 and 42 characteristics. Nevertheless, with 14 features, the QDPSO curve is the best in convergence and behavior, see Figure 5.15c. Furthermore, the curve of the QDPSO is the best in metaheuristic-based optimizers and all cases. On the other hand, the PSO_bond curve is worse in 84, 42, and 14 features. Finally, the execution time reduces considerably with our technic for image classification. For example, executing the algorithm with the global features (84 characteristics) takes around 1 week, while with features reduced (14 characteristics) takes 1 day. The reason is the computing of parameters number, which are equal to particle dimensions. Thus, with 84 features, each particle in the swarm has 22432 dimensions, while with 14 features, it has 802 dimensions; see Table 4.4.

Chapter 6

Conclusions

6.1 Conclusions

In this thesis, we proposed training artificial neural networks using particle swarm optimization with quantum delta particle behavior. Thus, we implemented a multi-layer perceptron using QDPSO and analyzed the performance with several benchmark classification datasets and the MCW dataset. Furthermore, we evaluated the behavior by reducing the features and increasing the classes and samples. Finally, we compared several metrics of the classification problem with different optimizers based on gradient and metaheuristics.

The tests evaluated the performance of the gradient-based and metaheuristic-based classifiers according to 4 well-known classification datasets and the MCW dataset. Thus, the tests begin with collecting the required datasets, preprocessing their data, and then running the model. After that, the fitness function is evaluated inside the model proposed. In addition, the QDPSO optimizes the neural network training before obtaining the classification results. Finally, each classifier is evaluated for accuracy, precision, recall, and f1 score in the training and testing phases.

The results show that QDPSO handles the benchmark datasets and the MCW dataset. For example, QDPSO, PSO, and L-BFGS have the best results in the training and testing phases with 100 and 1000 iterations using normalization. Moreover, it has better behavior and convergence with the balanced dataset, such as the circle and iris datasets. In the same way, QDPSO converges faster in MCW dataset with our technic proposed, and the behavior is the best of all classifiers. For instance, in the test phase and using the QDPSO classifier,

the accuracy of the MCW dataset was 91.69%, with 84 global extracted characteristics in an execution time of 1 week. In comparison, 14 reduced characteristics were 87.83% in a running time of 1 day.

TSNE is the best reducing algorithm to benchmark datasets with small features and Isomap with extensive characteristics. Conversely, the execution time for image classification reduces considerably because of the computing of the parameter numbers (the dimensions of each particle in the swarm) during the training in the artificial neural networks. For example, the parameter number for global characteristics is 22432, while the characteristics reduced is 802. Furthermore, the reliability of the QDPSO optimizer increased with 14 features (sixth part of global features) in the MCW dataset using Isomap as a reducing algorithm.

In the increasing samples and classes, the QDPSO and PSO curves of the loss cost converge faster with 2 and 3 classes using different samples. Moreover, they stagnate in the last iterations with 2 classes and the PSO curve with 3 classes using 2000 samples. With 4 classes, the QDPSO converges faster with 500 and 2000 samples. Nonetheless, the Adam curve is the best with 10000 samples. On the other hand, the QDPSO curve is very smooth during the training with 2, 3, and 4 classes using different samples.

Finally, the model and optimizer proposed in this thesis for training artificial neural networks have excellent results in the testing and training phases with the benchmark and MCW datasets. Furthermore, the behavior and convergence of the QDPSO are remarkable during the training for the datasets used in this thesis. In addition, the exploration and exploration stay balanced in the proposed optimizer, making the QDPSO classifier finds the best parameters of the multi-layer perceptron with the fewest resources.

6.2 Recommendations

This section provides some recommendations based on the issues and constraints identified in our model proposal.

- In the literature review, we saw that the output of activation functions is between 0 and 1 or -1 to 1, which allowed us to use this range as a limit for swarm initialization. However, this thesis uses a PSO with (PSO) and without (PSO_bound) limit where

the PSO has a better result than PSO_bound. Thus, we recommend not using bound in the QDPSO optimizer.

- The benchmark datasets were divided into 80% for training and 20% for testing, as shown in Table 4.2. This distribution allows us to obtain excellent results in the training and testing stages. However, we recommend a robust distribution with a validation phase. Thus, the split of the benchmark datasets could be 70% for training, 20% for testing, and 10% for validation.

- The MCW dataset has 1125 images divided into four categories: sunrise with 300 images, cloudy with 215 images, rainy with 253 images, and sunshine with 357 images, as described in Table 4.3. Thus, we recommend using a technic to balance the dataset, such as the correct evaluation metrics, resampling the training set, K-folder cross-validation, clustering the abundant class, or using data augmentation for imbalanced datasets.

- In the algorithm implementation, we use MLPClassifier from sklearn, which has Adam, SGD, and L-BFGS as classifiers. MLPClassifier allowed faster and easy implementation. However, this framework doe not permit the extraction of the loss cost value of the L-BFGS optimizer during the training. Thus we recommend using another framework that allows us to take out the loss cost value of all classifiers.

- In the proposed model, the execution time and computation cost depend on the number and dimensions of the swarm particles, as described in Equation 3.1. Thus, we recommend using technic to extract features, decrease features using a reducing algorithm, or use another artificial neural network architecture.

6.3 Future Works

This section suggests more studies and applications that might be considered in future works.

- The current project proposed a multi-layer perceptron as an artificial neural network model for classification problems. This model works well with benchmark datasets

with small features. However, without features reduced, it has some problems with significant characteristics, such as the MCW dataset. For this reason, we suggest a convolutional neural network implementation. CNN is an artificial network focused on classification problems. Furthermore, the architecture of CNN could be an excellent solution for datasets with significant characteristics.

- To extract the global features of the dataset, we used the Color Histogram, Hu Moments, and Haralick Texture, which extract the quantified color, shape, and texture. These technics reduce the computation cost in the classifiers based on metaheuristics. Moreover, they increase their accuracy and reduce classification errors. Thus, we suggest researching more technics to extract the global characteristics and make a comparison between them. Besides, we suggest using batch size and cross-validation to generate a faster and more robust model.

- The subsection of increasing sample and classes shows that the Adam optimizer has better behavior than other optimizers with an expensive sample; see Figure 5.10c. Furthermore, it was better with 84 and 42 features in the image classification subsection, as shown in Figures 5.15a and 5.15b. The Adam classifier performs better in these cases because it makes a stochastic sampling during the training. Thus, we suggest an implementation using this technic.

- The model proposed in this thesis was implemented with the scikit learn. This framework has admirable documentation and tools that allow excellent implementation of ANN. However, it has some limitations, such as taking out the loss cost value of the L-BFGS classifier during the training in the MLPClassifier module. For this reason, we suggest using PyTorch because it allows us the use the tensors and faster implementation with more tools than scikit learn. Furthermore, Tensorflow could be a good option too.

- The parameters in the model proposed are equal to the dimension of each particle in the swarm. Thus the computational cost depends on the number of parameters. In other words, if the number of parameters in the model increases, the computational cost also increases. On the other hand, QDPSO parallelization is possible because of

several implementations in the standard PSO [75]. Moreover, the model proposed in this work only used CPUs. For these reasons, we suggest an implementation using GPUs, parallelization, and different swarms for each layer in the network.

Bibliography

[1] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synthesis lectures on computer vision*, vol. 8, no. 1, pp. 1–207, 2018.

[2] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, 2020.

[3] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. IEEE, 2003, pp. 110–117.

[4] M. F. Ab Aziz, S. A. Mostafa, C. F. M. Foozy, M. A. Mohammed, M. Elhoseny, and A. Z. Abualkishik, "Integrating elman recurrent neural network with particle swarm optimization algorithms for an improved hybrid training of multidisciplinary datasets," *Expert Systems with Applications*, vol. 183, p. 115441, 2021.

[5] X. Liu, Z. Li, Z. Zhang, and G. Zhang, "Coal and gas outbursts prediction based on combination of hybrid feature extraction dwt+ fica–lda and optimized qpso-delm classifier," *The Journal of Supercomputing*, vol. 78, no. 2, pp. 2909–2936, 2022.

[6] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[7] F. Crick, "The recent excitement about neural networks." *Nature*, vol. 337, no. 6203, pp. 129–132, 1989.

[8] Y. Quintero, D. Ardila, E. Camargo, F. Rivas, and J. Aguilar, "Machine learning models for the prediction of the seird variables for the covid-19 pandemic based on a deep dependence analysis of variables," *Computers in Biology and Medicine*, vol. 134, p. 104500, 2021.

[9] D. J. Livingstone, *Artificial neural networks: methods and applications*. Springer, 2008.

[10] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 10 383–10 395. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/75877cb75154206c4e65e76b88a12712-Paper.pdf

[11] A. C. Mater and M. L. Coote, "Deep learning in chemistry," *Journal of chemical information and modeling*, vol. 59, no. 6, pp. 2545–2559, 2019.

[12] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, "Discovering physical concepts with neural networks," *Physical review letters*, vol. 124, no. 1, p. 010508, 2020.

[13] T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine, "Enforcing analytic constraints in neural networks emulating physical systems," *Physical Review Letters*, vol. 126, no. 9, p. 098302, 2021.

[14] P. G. Brodrick, A. B. Davies, and G. P. Asner, "Uncovering ecological patterns with convolutional neural networks," *Trends in ecology & evolution*, vol. 34, no. 8, pp. 734–745, 2019.

[15] S. Christin, É. Hervet, and N. Lecomte, "Applications for deep learning in ecology," *Methods in Ecology and Evolution*, vol. 10, no. 10, pp. 1632–1644, 2019.

[16] R. Ptucha, F. P. Such, S. Pillai, F. Brockler, V. Singh, and P. Hutkowski, "Intelligent character recognition using fully convolutional neural networks," *Pattern recognition*, vol. 88, pp. 604–613, 2019.

[17] B. R. Kavitha and C. Srimathi, "Benchmarking on offline handwritten tamil character recognition using convolutional neural networks," *Journal of King Saud University-Computer and Information Sciences*, 2019.

[18] G. Yao, T. Lei, and J. Zhong, "A review of convolutional-neural-network-based action recognition," *Pattern Recognition Letters*, vol. 118, pp. 14–22, 2019.

[19] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19 143–19 165, 2019.

[20] A. Celikyilmaz, E. Clark, and J. Gao, "Evaluation of text generation: A survey," *arXiv preprint arXiv:2006.14799*, 2020.

[21] B. Alshemali and J. Kalita, "Improving the reliability of deep neural networks in nlp: A review," *Knowledge-Based Systems*, vol. 191, p. 105210, 2020.

[22] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[23] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 33–47, 2020.

[24] C. L. Srinidhi, O. Ciga, and A. L. Martel, "Deep neural network models for computational histopathology: A survey," *Medical Image Analysis*, vol. 67, p. 101813, 2021.

[25] S. Sivaranjini and C. Sujatha, "Deep learning based diagnosis of parkinson's disease using convolutional neural network," *Multimedia tools and applications*, vol. 79, no. 21, pp. 15 467–15 479, 2020.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[27] K. S. Chahal, M. S. Grover, K. Dey, and R. R. Shah, "A hitchhiker's guide on distributed training of deep neural networks," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 65–76, 2020.

[28] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[29] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[30] E. S. Peer, F. van den Bergh, and A. P. Engelbrecht, "Using neighbourhoods with the guaranteed convergence pso," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. IEEE, 2003, pp. 235–242.

[31] X. Li, "A non-dominated sorting particle swarm optimizer for multiobjective optimization," in *Genetic and evolutionary computation conference*. Springer, 2003, pp. 37–48.

[32] H. Fang, L. Chen, and Z. Shen, "Application of an improved pso algorithm to optimal tuning of pid gains for water turbine governor," *Energy Conversion and Management*, vol. 52, no. 4, pp. 1763–1770, 2011.

[33] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, vol. 1. IEEE, 2004, pp. 325–331.

[34] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[35] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, p. 574, 1959.

[36] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

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

[38] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[39] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Deep learning for hyperspectral image classification: An overview," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 6690–6709, 2019.

[40] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[41] G. Guo and N. Zhang, "A survey on deep learning based face recognition," *Computer vision and image understanding*, vol. 189, p. 102805, 2019.

[42] K. Anilkumar, V. Manoj, and T. Sagi, "A survey on image segmentation of blood and bone marrow smear images with emphasis to automated detection of leukemia," *Biocybernetics and Biomedical Engineering*, vol. 40, no. 4, pp. 1406–1420, 2020.

[43] R. Moradi, R. Berangi, and B. Minaei, "A survey of regularization strategies for deep models," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 3947–3986, 2020.

[44] G. Pandey and U. Ghanekar, "Classification of priors and regularization techniques appurtenant to single image super-resolution," *The Visual Computer*, vol. 36, no. 6, pp. 1291–1304, 2020.

[45] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, no. 1, pp. 1–74, 2021.

[46] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[47] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[50] R. Balamurugan, A. Natarajan, and K. Premalatha, "Stellar-mass black hole optimization for biclustering microarray gene expression data," *Applied Artificial Intelligence*, vol. 29, no. 4, pp. 353–381, 2015.

[51] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, no. 2, pp. 239–287, 2009.

[52] J. Zhao, J. Sun, and W. Xu, "A binary quantum-behaved particle swarm optimization algorithm with cooperative approach," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 1, p. 112, 2013.

[53] X. Peng, L. Li, and F.-Y. Wang, "Accelerating minibatch stochastic gradient descent using typicality sampling," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 11, pp. 4649–4659, 2019.

[54] D. Chang, S. Sun, and C. Zhang, "An accelerated linearly convergent stochastic l-bfgs algorithm," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3338–3346, 2019.

[55] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin, "Deep learning on image denoising: An overview," *Neural Networks*, vol. 131, pp. 251–275, 2020.

[56] H. M. Balaha, M. H. Balaha, and H. A. Ali, "Hybrid covid-19 segmentation and recognition framework (hmb-hcf) using deep learning and genetic algorithms," *Artificial Intelligence in Medicine*, vol. 119, p. 102156, 2021.

[57] M. Borhani, "Multi-label log-loss function using l-bfgs for document categorization," *Engineering Applications of Artificial Intelligence*, vol. 91, p. 103623, 2020.

[58] J. Rafati and R. F. Marcia, "Improving l-bfgs initialization for trust-region methods in deep learning," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 501–508.

[59] R. K. Kennedy, T. M. Khoshgoftaar, F. Villanustre, and T. Humphrey, "A parallel and distributed stochastic gradient descent implementation using commodity clusters," *Journal of Big Data*, vol. 6, no. 1, pp. 1–23, 2019.

[60] F. Cheng and H. Shen, "An improved recurrent neural network for radio propagation loss prediction," in *2010 International Conference on Intelligent Computation Technology and Automation*, vol. 1. IEEE, 2010, pp. 579–582.

[61] H.-B. Ly, M. H. Nguyen, and B. T. Pham, "Metaheuristic optimization of levenberg–marquardt-based artificial neural network using particle swarm optimization for prediction of foamed concrete compressive strength," *Neural Computing and Applications*, vol. 33, no. 24, pp. 17 331–17 351, 2021.

[62] R. C. Green II, L. Wang, and M. Alam, "Training neural networks using central force optimization and particle swarm optimization: insights and comparisons," *Expert Systems with Applications*, vol. 39, no. 1, pp. 555–563, 2012.

[63] M. Chanda and M. Biswas, "Plant disease identification and classification using back-propagation neural network with particle swarm optimization," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2019, pp. 1029–1036.

[64] F. E. F. Junior and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation*, vol. 49, pp. 62–74, 2019.

[65] M. G. Abdolrasol, R. Mohamed, M. A. Hannan, A. Q. Al-Shetwi, M. Mansor, and F. Blaabjerg, "Artificial neural network based particle swarm optimization for microgrid optimal energy scheduling," *IEEE Transactions on Power Electronics*, vol. 36, no. 11, pp. 12 151–12 157, 2021.

[66] R. Ma, Y. Liu, X. Lin, and Z. Wang, "Network anomaly detection using rbf neural network with hybrid qpso," in *2008 IEEE International Conference on Networking, Sensing and Control*. IEEE, 2008, pp. 1284–1287.

[67] Y. Fu-guang and Z. Xian-xin, "An improved qdpso training hypersphere one class support vector machine," in *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, vol. 1. IEEE, 2011, pp. 396–400.

[68] Y. Li, J. Xiao, Y. Chen, and L. Jiao, "Evolving deep convolutional neural networks by quantum behaved particle swarm optimization with binary encoding for image classification," *Neurocomputing*, vol. 362, pp. 156–165, 2019.

[69] Z.-k. Feng, W.-j. Niu, Z.-y. Tang, Z.-q. Jiang, Y. Xu, Y. Liu, and H.-r. Zhang, "Monthly runoff time series prediction by variational mode decomposition and support vector machine based on quantum-behaved particle swarm optimization," *Journal of Hydrology*, vol. 583, p. 124627, 2020.

[70] R. Concepcion, E. Dadios, J. Alejandrino, C. H. Mendigoria, H. Aquino, and O. J. Alajas, "Diseased surface assessment of maize cercospora leaf spot using hybrid gaussian quantum-behaved particle swarm and recurrent neural network," in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*. IEEE, 2021, pp. 1–6.

[71] A. Gbeminiyi, "Multi-class weather dataset for image classification," *Mendeley Data*, 2018.

[72] D. Ping Tian *et al.*, "A review on image feature extraction and representation techniques," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 4, pp. 385–396, 2013.

[73] M.-K. Hu, "Visual pattern recognition by moment invariants," *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962.

[74] R. M. Haralick, "Statistical and structural approaches to texture," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 786–804, 1979.

[75] S. Lalwani, H. Sharma, S. C. Satapathy, K. Deep, and J. C. Bansal, "A survey on parallel particle swarm optimization algorithms," *Arabian Journal for Science and Engineering*, vol. 44, no. 4, pp. 2899–2923, 2019.