# Global Optimization Techniques Based on Swarm-intelligent and Gradient-free Algorithms

Futong Li

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the
Master of Applied Science degree

School of Electrical and Computer Engineering
Faculty of Engineering
University of Ottawa

**Abstract**

The need for solving nonlinear optimization problems is pervasive in many fields. Particle swarm optimization, advantageous with the simple underlying implementation logic, and simultaneous perturbation stochastic approximation, which is famous for its saving in the computational power with the gradient-free attribute, are two solutions that deserve attention. Many researchers have exploited their merits in widely challenging applications. However, there is a known fact that both of them suffer from a severe drawback, non-effectively converging to the global best solution, because of the local "traps" spreading on the searching space. In this article, we propose two approaches to remedy this issue by combined their advantages.

In the first algorithm, the gradient information helps optimize half of the particles at the initialization stage and then further updates the global best position. If the global best position is located in one of the local optima, the searching surface's additional gradient estimation can help it jump out. The second algorithm expands the implementation of the gradient information to all the particles in the swarm to obtain the optimized personal best position. Both have to obey the rule created for updating the particle(s); that is, the solution found after employing the gradient information to the particle(s) has to perform more optimally.

In this work, the experiments include five cases. The three previous methods with a similar theoretical basis and the two basic algorithms take participants in all five. The experimental results prove that the proposed two algorithms effectively improved the basic algorithms and even outperformed the previously designed three algorithms in some scenarios.

# Table of Contents

# List of Tables

# List of Figures

# Glossary

**SPSA** Simultaneous Perturbation Stochastic Approximation

**PSO** Particle Swarm Optimization

**SAD PSO** Stochastic-approximation-driven PSO

**SPPSO** Simultaneous Perturbation Particle Swarm Optimization

**SPSA-PSO** Serial integration of the two single methods is termed as SPSA–PSO

**RHN** Restricted Hopfield Network

# Nomenclature

| | |
|---|---|
| $\Delta_k$ | Perturbation vector |
| $\omega$ | Decreasing linear inertia weight |
| $c_1$ | Cognitive velocity parameter |
| $c_2$ | Social velocity parameter |
| $i$ | Particle's number |
| $lr$ | Particles' learning rate |
| $M$ | Particles' number |
| $N$ | Particles' movements number |
| $P$ | Particles' local optimization iteration |
| $p_g(t)$ | Best swarm position |
| $p_i(t)$ | Best individual particle position |
| $r_1, r_2$ | Random numbers between 0 and 1 |

$t$ Time Step t

$v_i(t)$ Particle velocity at time step t

$V_{max}$ Bound of particles' velocity (here equals $X_{max}$)

$x_i(t)$ Particle position at time step t

$X_{max}$ Bound of particles' position

$y_g^{best}$ Best swarm error

$y_g^{best}(t)$ Best swarm error

$y_i(t)$ Output of the objective function

$y_i(t)$ Result of objective function evaluation

$y_i^{best}$ Best particle error for particle $i$

$y_i^{best}(t)$ Best particle error

# Chapter 1

# Introduction

## 1.1 Problems and Motivation

Optimization problems gradually become one of the most urgent-to-be-solved issues in Engineering, mathematics, computer science, and economics, which are the problems of finding the relatively best solution from multiple legit solutions (mentioned in [1], [3], etc.). Researchers face two varieties of problems in the optimizing field — linear and nonlinear — depending on whether the relationship between the objective function and its variables is linear or not. While linear optimization problems can easily find a single best solution, nonlinear counterparts are in the opposite scenario because of the local optima. Nonlinearities could yield many local minima (or maxima), and the best solution, commonly called the global optima, is the most minimal (or maximal) result among the local solutions. That is the reason why nonlinear optimization problems are intrinsically more difficult to solve. However, many real-world optimization problems are nonlinear, and solving them is the basis of various analyses. Jumping out of those local optima to find

the global optima is a necessity. Many researchers improve existing well-known algorithms, such as the approaches included in Evolutionary Techniques and Swarm Intelligence.

Local optimal solutions could be so harmful that they damage analysis, businesses, and decision-making. It is not easy to find the global optima for nonlinear optimization problems without stuck in local optima. Not only that, but the limitation of computational power is also a headache. Calculations directly relying on the gradient information could lead to a massive calculation load and extra difficulty for computation. Therefore, it is helpful to use a computationally friendly strategy for complicated nonlinear optimization systems.

## 1.2 Thesis Objective

The thesis's main objective is to use innovative methods to solve nonlinear problems in different fields, e.g., data classification and associative memory. They are two classical fields involved in nonlinear problems, which attract many researchers to study.

Two novel approaches are proposed based on some previous works, which are swarm-strategy combining with gradient-free algorithms. This work compares with the standard algorithms and the earlier improvements to draw a relatively thorough picture of how the proposed methods work. Besides, the analysis of the results reasonably explains the optimal performance of the proposed algorithms.

This work's key point of view is to make use of the swarm-intelligent and gradient-free characteristics of two basic algorithms to provide a better strategy for detecting the global best solution of some nonlinear optimization problems.

## 1.3 Thesis Methodology

Trapped by the local minimum or maximum (depends on the type of optimization problem) is a situation that all the researchers are reluctant to face. For dealing with nonlinear optimization problems, several algorithms have been presented, such as Genetic Algorithm (GA), PSO, and Differential Evolution (DE). According to the comparison result from [10], the PSO shows its ability to be a qualified candidate for nonlinear optimization problems solver. The PSO, developed by Kennedy and Eberhart, simulates "the movement of a flock of birds" intended for social behavior. Within the scope of a specific search space (typically a uniform distribution area), a population (called a swarm) of candidate solutions (called particles) "fly" and "gather" to the position leading to the best solution. Because the initial position is distributed, in the desired situation, all over the search space, the particles are more likely to fly over some undesirable local optima and finally find the global optima. More importantly, it is meta-heuristics. Therefore it does not need any preset assumption and does not take the gradient for the objective function into account. From those two aspects, the PSO is not only suitable to be the solution of classic differentiable problems but also to be that of some non-differentiable problems. However, no configuration can guarantee that the PSO will return the optimal result in any search space. Particles might converge prematurely and get trapped in a local optimum due to the lack of fine-tuned parameter configuration.

Considering the computational cost, the direct calculation of gradient from the differential of the objective function occupies amounts of computing power. One of the most widely-used algorithms has to be mentioned, which is Gradients Descent (GD). Like most gradient-based algorithms, however, it is dragged by the "vanishing gradient" issue, which means that the gradients will be extremely close to zero and seem to "disappear" after

a bunch of iterations in deep neural networks. Its drawback drastically prevents the parameters from changing themselves and further stops the training procedure unexpectedly. Besides, the probability of trapping into local optima when applying GD for optimization may also be increased, especially for the problems involved in a large number of unknown parameters or defined by the multi-modal objective functions. SPSA, proposed by Spall in 1992, is a suitable scheme for optimizing objective functions properly. It is a recursive optimization algorithm that does not rely on the direct gradient (derivative) of the objective functions. Instead, it is built upon approximation to the gradient formed from only two measurements of the functions [24]. Therefore, it is good at solving high-dimensional optimization problems. The second "S" in the standard SPSA stands for stochasticity, which means a randomly generated perturbation vector simultaneously varies these two measurements for all dimensions. Through this gradient-information-free algorithm, a huge amount of mathematical calculations of derivatives are skipped. For those systems that lack computation powers, the standard SPSA helps boost the optimizing capability. Many studies have already proven that the standard SPSA is very efficient and effective in solving complex system optimization problems. However, the computational advantage of the standard SPSA cannot cover its disadvantage — no guarantee for the global convergence. The standard SPSA is a somewhat different version of Stochastic Approximation (SA). In the previous work done by other researchers, the theory developed for the standard SPSA does not fully consider the global convergence, even though the standard SA supports the global convergence. [17] Since the different representations of gradient approximation (i.e., using two measurements), the existing global convergence theory for usual SA (e.g., FDSA) is not useful for the standard SPSA.

To eliminate, at least alleviate, the demerits of the above-mentioned algorithms, they should be adjusted. Either the standard SPSA and PSO itself has limitations to solve

the global optimization problems. Several researchers have demonstrated their thoughts of combing the standard SPSA and the PSO in different ways, stated in [11], [23], and [15]. The authors of [11] applied the standard SPSA to the procedure of obtaining the global best position for the basic PSO. [23] stacked those two algorithms together (i.e., inputting the optimal parameters after the standard SPSA's optimization step to the PSO for the final optimal outcome). [15] added the gradient information into the velocity's updating.

After considering the structure of the basic PSO, the fact that every personal best position could be the potential global best position is obvious. Within the PSO algorithm, the movement of the particles in the swarm dominates by the global best particle, so they are worthwhile to be selected.

In this thesis, one of the PSO variants is improved by applying the standard SPSA to optimize the global best position of the warm and the personal best position of each particle. Instead of maintaining the fitness of the PSO by itself alone, the gradient information estimating from the objective function by the standard SPSA is adding to the PSO, which will be helpful to converge to the global optimum.

## 1.4 Thesis Contributions

In short, the contributions of the thesis are as follows:

- Use swarm-strategy and gradient-free methods to design the proposed algorithms. The swarm-strategy method stands for the PSO algorithm, and the gradient-free method is the standard SPSA algorithm. Exploit the advantages of each algorithm and try to avoid their weakness by combining them is the underlying idea.

- Apply the proposed algorithms to find the global optimum without or with a lower probability stuck into a local optimum. Use the standard SPSA to provide the gradient information and explore the local searching space around the particles' position in the swarm.
- Compare the standard SPSA and PSO with the proposed algorithms. Also, simulate the previous improvements to observe the merits of the proposed algorithms.
- Improves the accuracy of convergence without scarifying too much computational power of systems. Because both the standard SPSA and the PSO are computationally friendly approaches, even using them together will not occupy significant energy, mitigate the problem about reaching out the best solution when implementing them solely.

The improved algorithms are promising approaches for finding the global solution to optimization problems.

## 1.5 Thesis Outline

The paper is organized as follows. In Chapter 2, detailed descriptions of the standard SPSA and the basic PSO is provided, as well as the previous designs of similar concept (i.e., SAD PSO, SPPSO, SPSA-PSO). The classic Multi-layer Perceptron and a new recurrent neural network called Restricted Hopfield Network, RHN, are introduced as well. Those neural networks are used in the experiments for testing the proposed algorithms. In Chapter 3, a detailed description of the workflow and essential programming parameters' settings of the proposed algorithms is described. Experiments for some scenarios are presented in Chapter 4 and Chapter 5. Conclusions and future work are described in Chapter 6.

# Chapter 2

# Related Works

## 2.1 Basic Optimization Algorithms

### 2.1.1 Particle Swarm Optimization

Particle swarm optimization (PSO) is an algorithm that graphically mimics how social animals behave within a flock, such as birds and fish. They seem to unintentionally choreograph and regroup from a loose and random distribution to a formation centered around the most optimal position. Kennedy, Eberhart, and Shi attributed the concept was transformed into a straightforward optimization algorithm [7].

There are a bunch of elements in the PSO. Individual "birds" are called particles, which are "flown" through hyper-dimensional search space. They simulate the procedure of searching for potential solutions to optimization problems. A swarm is constructed by all the particles, an analogy of a flock. Each particle has its own position-changing pace and but has an overall direction. Eventually, the particles in the entire swarm will gather

at the "successful region" in the hyper-space. One of the key parts of the PSO algorithm is how to update the position of every particle. Updates to the position of particles within the searching space are based on the experience, or knowledge, of individuals to emulate the success of other individuals [7]. Every particle can not only be itself but also might be one of the neighbors for others (depends on whether the algorithm is implemented as the global best PSO or as the local best PSO). Regardless, the relationship between particles in a swarm is obviously mutual-influenced. In addition to the changing behavior of particles, the objective function is also essential. Basically, the objective function is the problem waiting to be optimized and is the calculation formula for the next searching step along with the time series.

Two steps roughly do the positional updates: first, updating velocity information; second, updating position information. Every particle owns velocity and position. In general, adding velocity element to position helps the particles move in the searching space, i.e.

$$x_i(t+1) = x_i(t) + v_i(t+1),\ x_i(0) \sim U(x_{min}, x_{max}) \tag{2.1}$$

where $x_i(t)$ is the current position of particle $i$ (i.e., $i$ is the number of particles) in the search space at time $t$, $x_i(t+1)$ represents the next position, and $v_i(t+1)$ is the velocity for updating $x_i(t)$ to $x_i(t+1)$. The initial distribution of the swarm is important, because the result should be the global optima. Initialize the particles using uniform distribution is an approach to ensure they can cover the entire search space. In the paper, we use Golort Initialization, and I will introduce it in the following part. The velocity component $v_i$ is a vector containing velocity information of all particles in the swarm.The updating formula for the velocity is:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1[p_i(t) - x_{ij}(t)] + c_2 r_2[p_g(t) - x_{ij}(t)] \tag{2.2}$$

where $i$ maintain the same meaning with the previous equation (2.1), and $j$ represents the number of dimensions that the search space holds, $v_{ij}(t)$ and $v_{ij}(t+1)$ are the velocity of particle $i$ at time $t$ and the next time step $t+1$ respectively. $c_1$, $c_2$, $r_1$ and $r_2$ are parameters for velocity scaling. $c_1$ and $c_2$ are constants which could be positive or negative to accelerate or decelerate the particle's movement. $r_1$ and $r_2$ are the numbers following the uniform distribution (i.e., $r_{1j}(t), r_{2j}(t) \sim U(0,1)$) to combine stochasticity into the PSO. $p_i(t)$ and $p_g(t)$ are the personal best position for particle $i$ and the global best position found from the current swarm respectively. The global best PSO (abbreviated as the gbest PSO) is used here for searching the global solution, and that means every particle act as a candidate of other particles' neighbour.

**Algorithm 2.1** The gbest PSO

Initialization

1: Set constants $N$, $c_1$, $c_2$

2: Randomly initialize particle positions $x_i(0) \in D$ in $\mathbb{R}^M$ for $i = 1, 2, ..., M$

3: Randomly initialize particle velocities $v_i(0) \in [0, V_{max}]$ for $i = 1, 2, ..., M$

4: Set $t = 1$

Optimization

1: **while** $t \leqslant N$ **do**

2: Evaluate function value $y_i(t)$ using design space coordinates $x_i(t)$

3: **if** $y_i(t) \leqslant y_i^{best}(t)$ **then**

4: $y_i^{best}(t) = y_i(t),\ p_i(t) = x_i(t)$

5: **end if**

6: **if** $f_i(t) \leqslant y_g^{best}(t)$ **then**

7: $y_g^{best}(t) = y_i(t),\ p_g(t) = x_i(t)$

8: **end if**

9: If stopping condition is satisfied, terminate

10: Update all particles velocities $v_i(t)$

11: Update all particles positions $x_i(t)$

12: Increment $t$

13: **end while**

From equation 2.2, the calculation of velocity involves three terms: the previous velocity $v_{ij}(t)$, cognitive component $c_1 r_1[p_i(t) - x_{ij}(t)]$, and social component $c_2 r_2[p_g(t) - x_{ij}(t)]$. The name of each component somehow clearly implies the function of itself. The previous velocity is also named as the inertia component that plays a role in recording the moving

direction of the previous step for the particle. The cognitive component is a tie between the current step and the step generating the best performance, ensuring individuals can be pulled back to the most satisfying place they have ever been. In the gbest PSO, the social component measures the performance among the neighborhood and controls all the particles moving close to the global optima. The convergent performance of the basic PSO has also been discussed in [2], [22], and [5].

### 2.1.2 Basic Variations for PSO

Despite the fact that the basic PSO has solved many optimization problems, it is not enough. The research community has developed some adjustments for the basic algorithm. There is a term, "exploration-exploitation trade-off" [7], which concludes the algorithm's job. On the one hand, exploration stands for the ability that how many different regions of the search space could be explored by particles. On the other hand, exploitation can shrink the searching range for the swarm to the place near the optima. It is significant for PSO to balance those contradictory abilities.

First, controlling the velocity increasing or decreasing within a moderate range. The technique is called "velocity clamping". Unfortunately, the velocity easily explodes to enormous values, which means particles have big steps, easily jump over the best position, and even leave the search space. For eliminating explosion, the velocity of each particle is restricted by a specific value, which is the maximum number that the velocity could reach. If the velocity is beyond the specified maximum velocity, replace the current velocity with it during the calculation. The velocity is checked before the next movement, so a proper step is ensured. A common practice for setting the maximum values in corresponding

dimensions is to compute from the position. That is,

$$V_{max,j} = \delta(x_{max,j} - x_{min,j}),\ \delta \in (0, 1] \tag{2.3}$$

Second, instead of using the previous time steps velocity, applying an inertia weight to that component. The inertia weight is highly related to the previous step and similar to a controller of the momentum of particles. Although the constant inertia weight is considered, decreasing the inertia weight linearly performs better. A big value benefits exploration and a small value promotes exploitation. When implementing the linear inertia weight with the gbest PSO, it is as follows:

$$v_{ij}(t+1) = \omega(t)v_{ij}(t) + c_1 r_1[p_i(t) - x_{ij}(t)] + c_2 r_2[p_g(t) - x_{ij}(t)] \tag{2.4}$$

$$\omega(t+1) = \alpha\omega(t),\ \alpha \in (0, 1] \tag{2.5}$$

where $\alpha$ adds a linear factor to the velocity updating. Since $\alpha$ is a number between 0 and 1, the inertia weight drops gradually, and the movement of particles becomes slower, which means the aim of the swarm switch from exploration to exploitation reasonably. For linear decreasing, usually, the inertia decreases from 0.9 to 0.4. Multiple modifications have been proposed in the past decades. In this paper, the two improvements mentioned above for the standard PSO is considered.

### 2.1.3 Simultaneous Perturbation Stochastic Approximation

In the field of solving optimization problems, a computational-friendly method has been developed by James C. Spall in 1998, called Simultaneous Perturbation Stochastic Approximation (SPSA). This algorithm is well-known for its power and relative ease of implementation. Unlike the gradient-based method, the gradient-free SPSA only requires two

measurements of the objective function no matter how dimensionally complex the problem is, which saves massive calculations. Thus, SPSA is suitable to be a solution to addressing high-dimensional optimization problems.

Since the SPSA is a relatively mature algorithm, a lot of studies and improvements have been proposed. For instance, the consideration of enhanced convergence ( [4] and [32]), the idea of Constrained optimization ( [28] and [20]), the though of optimal choice of $\Delta k$ distribution ( [21] and [9]), and the application as a global optimization solution ( [18] and [19]) have been developed.

The standard SPSA algorithm can be summarized into six steps.That are,

**Algorithm 2.2** The Standard SPSA

Initialization

1: $a, c, A, \alpha, \gamma$

2: iteration number $n$

3: $k = 1$

Optimization

1: **while** $k \leqslant P$ **do**

2: $a_k = \frac{a}{(k+A)^\alpha}$

3: $c_k = \frac{c}{k^\gamma}$

4: $\Delta_k = random.randint(0, 2) * 2 - 1$ ▷ Bernoulli distribution

5: $\theta_+ = \theta + c_k * \Delta_k$

6: $\theta_- = \theta - c_k * \Delta_k$

7: $y_+ = y(\theta_+)$ ▷ $y(*)$ is the obejective function

8: $y_- = y(\theta_-)$

9: $\Delta_g = (y_+ - y_-)/(2 * ck * \Delta_k)$

10: $\theta = \theta - a_k * \Delta_g$

11: Increment $k$

12: **end while**

Step 1 Initialization. It is crucial to choose a set of suitable values for coefficients $a, c, A, \alpha, \gamma$. $a_k$ and $c_k$ are nonnegative scalar gain coefficients, calculated according to the following equations:

$$\begin{aligned} a_k &= \frac{a}{(k+1+A)^\alpha} \\ c_k &= \frac{c}{(k+1)^\gamma} \end{aligned} \tag{2.6}$$

An effectively valid practice is $\alpha = 0.602,\ \gamma = 0.101$. $a, A, c$ are problem-related hyper-parameters. Normally, $a, A$ can be set together, where $A = 0.1 \times iter_{max}$ (i.e. ten percent of the optimization iterations) and $a_0 = \frac{a}{(k+1+A)^{0.602}}$.

Step 2 Generating the simultaneous perturbation vector $\Delta_k$. $\Delta_k$ is a random p-dimensional vector following zero-mean probability distribution. As Spall saying, a Bernoulli distribution can yield desired results in some scenarios.

Step 3 Evaluating the objective function $y(\hat{\theta}_k)$. The standard SPSA is a two-sided gradient approximation, therefore $y(\hat{\theta}_k + c_k\Delta_k), y(\hat{\theta}_k - c_k\Delta_k)$ are used.

Step 4 Calculating gradient approximation as follows:

$$\Delta g(\hat{\theta}_k) = \frac{y(\hat{\theta}_k + c_k\Delta_k) - y(\hat{\theta}_k - c_k\Delta_k)}{2c_k} \begin{bmatrix} \Delta_{k1}^{-1} \\ \Delta_{k2}^{-1} \\ \cdot \\ \cdot \\ \cdot \\ \Delta_{kp}^{-1} \end{bmatrix} \tag{2.7}$$

, where $c_k, \Delta_k$ are chosen are generated before.

Step 5 Updating the current estimate $\hat{\theta}_k$ to a more optimal $\hat{\theta}_{k+1}$, using

$$\hat{\theta}_{k+1} = \hat{\theta}_k - a_k\Delta g_k(\hat{\theta}_k) \tag{2.8}$$

after all the preparation and calculation.

Step 6 If the termination condition is satisfied or $k$ touches the maximum number of iterations, then stop the optimization process; otherwise, $k = k + 1$, and return to the second step to generate a new simultaneous perturbation vector.

### 2.1.4 The Features of SPSA

Comparing to the algorithms that directly rely on the gradient information associated with the objective function on every dimension, SPSA does not use the detailed relationship between the parameters waiting for adjusted and the objective function waiting for minimized (or maximized). Computing the relationship between them accurately and efficiently is one of the main difficulties since the gradient information is not always available in some situations, e.g., multi-layer perception (MLP) and nonlinear feedback controller design. For instance, the gradient descent (GD), a famous gradient-based approach for backward propagation, has the "vanishing gradient problem" when implemented to the deep MLP. Because of the gradient knowledge loss, the parameters cannot be updated correctly, and the neural network could even be prevented from further training. In contrast, the gradient approximation approaches to tackle the problems with a rough picture of input-output relationships. This feature speeds up the convergence and makes training procedure easier.

When dealing with real-world problems, many factors have to be taken into account for deciding which algorithm is the most optimal candidate that could be applied under the current situation. A simple summary of the factors is:

1) whether the gradient information of the input-output relationship of the system is easy to be obtained;

2) whether enough computational power is allowed to achieve the overall optimization goal;

3) whether the practical convergence rate can be reached based on the finite samples.

After a thorough consideration of the above-mentioned three aspects, better schemes for solving the problem could be put forward. In general, if direct gradient information is trustworthy and ready for calculation, using the gradient-based methods is probably advantageous; if the full knowledge of the required relationship is lacking, the gradient-free methods (e.g., SPSA) are legit.

Furthermore, among the gradient-free algorithms, SPSA needs fewer calculation efforts. The finite difference stochastic approximation (FDSA) is one of the classical stochastic optimization approaches. For FDSA, the estimate of the gradient at iteration $k$ is calculated by $\Delta g(\hat{\theta}_k) = \frac{y(\hat{\theta}_k + c_k e) - y(\hat{\theta}_k - c_k e)}{2c_k}$, $e = \begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ . & . & & . \\ . & . & & . \\ 0 & 0 & ... & 1 \end{bmatrix}$. The number of measurements involved is equal to the number of dimensions of the problem. Contrarily, SPSA uses the same perturbation vector to be the numerator in all components, which means SPSA achieves magnificent savings in the evaluation. It proves that SPSA is more efficient than some classical stochastic approximation methods in reasonable general conditions.

## 2.2 Improvements of the Basic Optimization Algorithms

As the previous discussion, Both the standard SPSA and the basic PSO (even the PSO variants) cannot search for the global optimum. Therefore, researchers put forward some methods that take advantage of each algorithm's merit and combine them. Integrating the local searching ability and the cost-effective computation of the SPSA with the almost

full-coverage-distribution searching space of the PSO for achieving a better solution.

### 2.2.1 SPSA-PSO

SPSA-PSO, in [23], is one of the improvements of the combination of the standard SPSA and the basic PSO. Before moving all the particles in the swarm as the flock, the standard SPSA optimizes part of them (i.e., the standard SPSA helps partial swarm re-locate at a better starting position). Then utilizes the social behavior among all particles to find an optimal position as the what the particles do in the PSO. As they stated, the aim of SPSA-PSO is to create an ideal swarm at the initialization and perform the PSO more efficiently.

The number of the optimized particles should be tuned as well. The initialization of the swarm can be derived from the following forms:

$$\begin{aligned} Init_{SPSA} &= [X_{SPSA,0}, X_{SPSA,1}, X_{SPSA,2}, ..., X_{SPSA,i}] \\ Init_{rndm} &= [X_{rndm,0}, X_{rndm,1}, X_{rndm,2}, ..., X_{rndm,S-i}] \\ Init_{swarm} &= Init_{SPSA} + Init_{rndm} \end{aligned} \tag{2.9}$$

,where $i$ is the number of particles that initialized by the standard SPSA, $S$ is the total number of particles in the desired swarm, and the rest $S - i$ particles naturally maintain the randomness for the swarm. $Init_{swarm}$ varies a lot due to the ratio of the number of random-generated particles and the number of SPSA-based particles. The ratio, $m$, is represented as $m = \frac{i}{S}$ which is the result of the number of particles determined by the standard SPSA divided by the total number of particles in the swarm. They revealed that $m = 0.5$ yields the best solution with a higher probability.

### 2.2.2 SPPSO

SPPSO involves the gradient information in the updating procedure of the PSO proposed in [15]. The gradient information is computed by the simultaneous perturbation method. The authors designed two algorithms in which the only difference is whether add the momentum of each particle or not (i.e., whether to add the velocity from the previous step or not).

When updating each particle, first applying the simultaneous perturbation procedure for obtaining the gradient information $\Delta g$, and then add this information to the update for the current particle's velocity. Notice that the gradient information they used here is only the single-side version (i.e., only the $f(x+c)$ is calculated), not the two measurements used in the standard SPSA. Therefore, the gradient information at the point is as follows: $\Delta g_i = \frac{f(x_i+c_i)-f(x_i)}{c_i}, (i = 1, 2, ..., n)$. The rest of the algorithm keeps the same with the basic PSO.

The updating procedure of particles' velocity demonstrates as follows:

$$\begin{aligned} v_i(t+1) &= c_1 r_1[y_i(t) - x_i(t)] + c_2 r_2[\hat{y_i}(t) - x_i(t)] - \Delta\hat{g} \\ &\textit{or} \\ v_i(t+1) &= \omega v_i(t) + c_1 r_1[y_i(t) - x_i(t)] + c_2 r_2[\hat{y_i}(t) - x_i(t)] - \Delta\hat{g} \\ &\textit{and} \\ \Delta\hat{g} &= \frac{y(x_i(t) + c_k) - y(x(t))}{c_k} \end{aligned} \tag{2.10}$$

(all the notations in these equations are the same as the previous definitions).

### 2.2.3 SAD PSO

SAD PSO improves the combination of the standard SPSA and the PSO by plugging the standard SPSA into the updating of the global best position. The authors in [11] also developed two algorithms, and the discrepancy of them is whether judging the performance of the global best position before and after optimized by the standard SPSA (i.e., whether applying the updating rule for replacing the global best position). The second approach only updated the global best position if and only if it is situated in a better position with a lower error value.

They referenced the works done by the researchers of the SPPSO but mentioned the weakness of their design. In the SPPSO, the gradient information was inserted into updating every particle's velocity, intentionally trying to boost each particle's searching ability. However, it could bring the overall fitness noise and confuse all the particles not moving towards the global best position (i.e., incorrectly directional moving appears). Besides, they also adopted the FGBF operation of [12] and substituted it with an artificial global best particle. This artificial global best particle takes the current global best particle place only when it satisfies a more optimal condition.

The initialization procedure of the works in [11] maintain the same way with the basic PSO, but the optimizing steps are different and shown in the following descriptions:

**Algorithm 2.3** The SAD PSO algorithm [11]

Optimization

**while** $t \leftarrow 1$ *to* $N$ **do**
  **while** $i \leftarrow 1$ *to* $M$ **do**
    Evaluate objective function $y_i(t)$
    **if** $y_i(t) \leq y_i^{best}$ **then**
      $y_i^{best} = y_i(t),\ p_i(t) = x_i(t)$
    **end if**
    **if** $y_i^{best} \leq y_g^{best}$ **then**
      $y_g^{best} = y_i^{best},\ p_g(t) = p_i(t)$
    **end if**
    Increment $i$
  **end while**
  $\theta = p_g(t)$
  Generate a perturbation vector $\Delta_t$
  $\theta_- = \theta - 2 * c_t * \Delta_t$
  $y = y(\theta),\ y_- = y(\theta_-)$
  $\Delta_g = (y - y_-)/(2 * c_t * \Delta_t)$
  $\theta = \theta - a_t * \Delta_g$
  Obtain the optimized global best position $temp_{p_g(t)} = \theta$
  **if** $temp_{y_g^{best}} < y_g^{best}$ **then**
    $p_{g^{best}} = temp_{p_g(t)}$
    $y_{g^{best}} = temp_{y_g^{best}}$
  **end if**
  **while** $i \leftarrow 1$ *to* $M$ **do**
    Velocity and position updating procedure in the basic PSO
    Increment $i$
  **end while**
  Increment $t$
**end while**

## 2.3 Related Neural Networks

### 2.3.1 Multilayer Perceptron

Multilayer perceptron (MLP) is the simplest kind of feed-forward artificial neural network, a classical way to solve nonlinear problems. The concept of MLP is that aggregating some nodes (i.e., processing units in layers) to compute some complex functions linearly and nonlinearly.

Several key components in MLP are the number of inputs, the number of nodes in each hidden layer, the number of output nodes, the number of layers (i.e., the depth of the network), and the activation function between layers. The input data are fitted into the first input layer, and the last output layer is the calculation results of the neural network. All the hidden layers are in the middle of the input and output layers. The value of input nodes and output nodes is known, whereas the value of hidden nodes is mysterious. MLP is a fully-connected network, which means every node in the hidden and output layers is connected to every node in its previous layer, and no connection between the nodes in the same layer. Therefore the input information is propagated in one direction (i.e., from the input layer to the output layer). Besides, activation functions are essential, which distinguish MLP from a linear perceptron. Without activation functions for layers (the output layer is exclusive), MLP will lose the nonlinearity and act as a linear model.

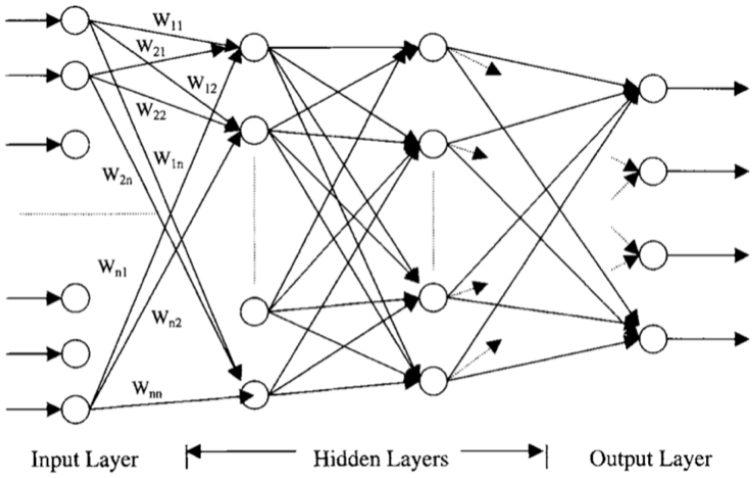



Figure 2.1: MLP architecture [6]

Mathematically, the calculation of the -hidden layer MLP (the input and output layers does not count) can be vectorized as:

$$\begin{aligned} H^1 &= f(W^1 X + b^1) \\ H^2 &= f(W^2 H^1 + b^2) \\ H^l &= f(W^l H^{l-1} + b^l) \\ \hat{Y} &= f(W^o H^l + b^o) \end{aligned} \tag{2.11}$$

, where $X$ is input matrix and $Y$ is output matrix. $H^l$ is the output from the $l$th layer, which contains the same rows and columns with the input matrix. $W^l$ and $b^l$ represents the connection weight matrix and the bias vector for the $l$th layer. $f(*)$ is activation functions, and it could be different from layer to layer. One of the most commonly used activation

functions is the sigmoid function, i.e.,

$$f(*) = \frac{1}{1 + \exp(-\alpha x)}. \tag{2.12}$$

Linear problems are commonly existing in the real world; however, linear models are less expressive power than nonlinear counterparts. Because of the nonlinear activation functions of each layer, MLP is more powerful.

When considering the learning law, the weights of nodes plays an essential role in interconnecting processing units. If one unit is activated by another unit or both units are fired, the weights of them are supposed to be increased; if two units act reversely, the weights of them would be decreased, like:

$$\begin{aligned} W(t+1) &= W(t) + \Delta W \\ \Delta W &= \alpha(Y - \hat{Y})X \end{aligned} \tag{2.13}$$

, where $X$ represents the input values for all the layers generally, $Y$ is the desired output, and $\hat{Y}$ is the actual output of the network. $\alpha$ is a hyper-parameter controlling the learning rate. Conventional MLP with backward propagation (back-prop) applies gradient descent (GD) to evaluate the objective function, then adjust the weights and biases to make the actual output closer to the desired output. Here, no need to repeat how GD works in the back-prop.

### 2.3.2 Hopfield Network

In various domains, neural networks have become preferred solutions nowadays, which can be used in function approximation, classification, data processing, and so forth. Recurrent neural networks (RNN), can exhibit temporal dynamic behavior and have additional stored

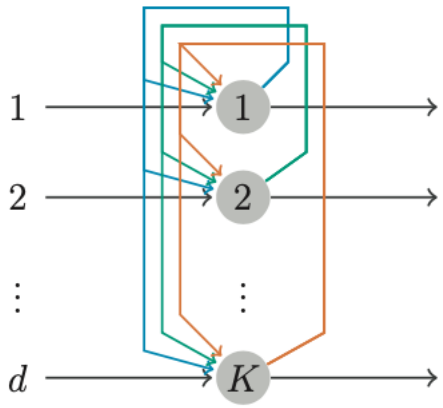



Figure 2.2: Visualization of Hopfield Network with K nodes matching d features

states. The core of a Hopfield Network is a model that can recover data (i.e., the patterns in the memory) after being fed with a distorted version of the same data (i.e., the data for testing its ability to reconstruct).

Similar to MLP, Hopfield Network can also be described as a network of nodes (or units, or neurons) connected by links. Each node has two states at any time in the time series, which could be 0 and 1, or -1 and +1. The assembly of neurons is fully connected, although they do not have self-loops shown in [29] Chapter 6. If K nodes is initialized in a Hopfield Network, $K * (K - 1)$ interconnected links are leaded to.

To clearly visualize the network, the state of each node can be vectorized, and at each time frame, a vector containing all states can be obtained and will be updated over time. There is a weight matrix $W$ that represents the importance of each link or the strength of the connection. Assume that the element $W_{ij}$ in the weight matrix indicates the link from node $i$ to node $j$. The link between nodes needs to be identical in whichever direction in the graph, shown in matrix form: $W_{ij} = W_{ji}$. Therefore, the weight matrix should be symmetric and zero-diagonal (even though this is unrealistic in the real world), which is

also the premise for the Hopfield Network to be stable. The state $S_i$ of node $i$ is decided by the following rule:

$$S_i = \begin{cases} -1 \ or \ 0 & \sum_j W_{ij} S_j \geqslant \theta_i \\ +1 & otherwise \end{cases} \tag{2.14}$$

, where $\theta_i$ is a threshold value of the node $i$. Commonly, $\theta_i$ could just be 0.

Hopfield Network uses the Hebbian rule to update the weight matrix, which is defined as "Neurons that fire together, wire together". The higher the value of a $W_{ij}$ weight, the more likely that the connected node $i$ and node $j$ will activate simultaneously [reference], the mathematical form is shown: $W_{ij} = \frac{1}{N} x_i x_j$ , where $x$ is the input pattern, and $N$ is the number of the digits (when input patterns use binary representation).

After initializing the network states with the inputs (i.e.m state vectors), the network will be evolved with the Hebbian laws introduced before, and it will converge to the states to desired values. That is due to the energy function associated with the Hopfield Network's state and denote it with:

$$E = -\frac{1}{2} \sum_{i,j} W_{ij} S_i S_j + \sum_i S_i \tag{2.15}$$

. No matter how the state of a node changes (i.e., the change can be from +1 to -1 or from +1 to -1), the energy function will always decrease, which is monotonically decreasing until reaching its lowest point. In one shot, when the states assume the original value (the uncorrupted value), the energy function will not change anymore.

Also, a Hopfield Network is an associative memory. The number of patterns stored in the network (i.e., the storage capacity of this associative memory) and the number of neurons maintain a linear relationship. By applying the Hebbian rule, the estimate of the capacity is not larger than 0.15K ( K is the number of nodes in the network), which is a relatively small amount.

### 2.3.3 Restricted Hopfield Network

A widely-used RNN is the Hopfield network; however, there are some limitations when apply it to tackle real-world problems. Derived from the standard Hopfield network, Restricted Hopfield Network can extend the stored memory capacity by being trained.

Hopfield network serves as an "associative (content-addressable) memory system", "optimization engine", etc. However, all the standard Hopfield network nodes only take two different states, conventionally either "-1 & 1" or "0 & 1", depending on the pre-set value of the threshold and whether the current non-binary value of each node exceeds their threshold or not. Additionally, this network can only be programmed to memorize some patterns instead of "learning patterns". Therefore, the volume of its memory capacity is about 0.15 times the number of neuron units in the network, which is small. Even though the Hopfield network is one of the most popular legit networks, it cannot be the candidate to solve the problem in a bunch of situations because of these limitations.

The introduction of Restricted Hopfield Network (RHN) is one of the improved alternatives to the Hopfield network [31]. RHN is not only expanded the memory capacity but also turned the discrete neural network into an analog neural network. Plus, it is also proposed for solving the trainable ability problem of the standard Hopfield network. The combination of the Hopfield network and Bidirectional Associative Memory (BAM) [16] is a rough representation of RHN. The backbone of RHN is two fully connected and no-intra-layer-connection layers. Specifically, the visible layer inputs and outputs the data and results, respectively, and one hidden layer was treated as enhancing this network, which conceals each node's state in this layer.

The energy function was derived by Yeap [28] using Lyapunov's Direct Method to prove that the network is stable. Like the other network, the following energy function is valid

if and only if weights are symmetric.

$$E = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{L}\omega_{ij}^{H}v_{i}^{H}V_{j}^{V} - \frac{1}{2}\sum_{j=1}^{L}\sum_{i=1}^{M}\omega_{ji}^{V}V_{j}^{H}V_{i}V - \sum_{i=1}^{M}v_{i}^{H}\theta_{i}^{H} - \sum_{i=1}^{L}v_{i}^{V}\theta_{i}^{V} \tag{2.16}$$

Table 2.1: Parameters for RHN

| Parameters | Description |
|---|---|
| $u_i^H$ | Sum of all inputs to the hidden nodes |
| $\omega_{ij}^H$ | Weight connecting the output of visible node j to the input of hidden node $i$ |
| $V_j^V$ | Output of visible node $j$ |
| $\theta_i^H$ | Threshold of hidden node $i$ |
| $V_i^H$ | Output of hidden node $i$ |
| $u_i^V$ | Sum of all inputs to the output nodes |
| $\omega_{ij}^V$ | Weight connecting the output of hidden node j to the input of visible node $i$ |
| $V_j^H$ | Output of hidden node $j$ |
| $\theta_i^H$ | Threshold of output node $i$ |
| $V_i^V$ | Output of output node $i$ |
| $I_j$ | Initial input of the visible node $j$ |
| $f(*)$ | Activation function of nodes |

The updates of the network follows the differential equations below:

$$\begin{aligned}\frac{du_i^H}{dt} &= \sum_{j=1}^{L}\omega_{ij}^{H}V_j^V + \theta_i^H \\ V_i^H &= f(u_i^H)\end{aligned} \tag{2.17}$$

$$\frac{du_i^V}{dt} = \sum_{j=1}^{M} \omega_{ij}^V V_j^H + \theta_i^V$$
$$V_i^V = f(u_i^V) \tag{2.18}$$

, where the first one describes the forward path from the visible layer to the hidden layer, and the second one shows the backward path. For the initial condition, the outputs of the visible nodes are the inputs, and the hidden nodes are zero, i.e.,

$$V_j^V(0) = I_j,\ V_i^H(0) = 0 \tag{2.19}$$

The activation function of the nodes, including hidden and visible nodes, is a sigmoid function. The output of the nodes, therefore, is a positive value smaller than 1. The digital form of the visible nodes can be obtained by applying threshold units to the analog output of visible nodes.

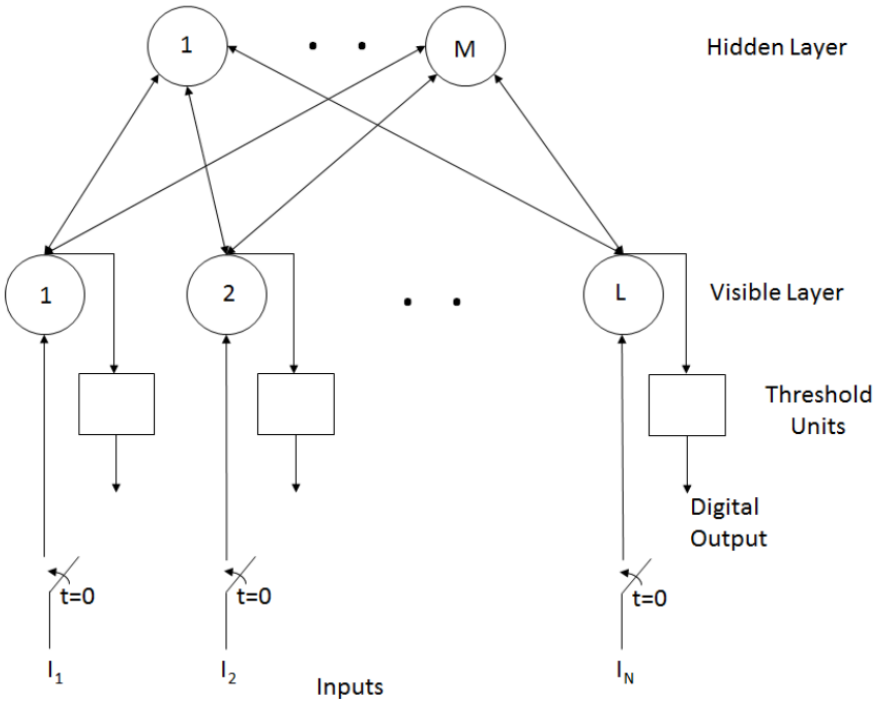



Figure 2.3: Restricted Hopfield Network (RHN) [31]

Intuitively, the inputs for RHN, just like several "balls", bounce between the visible and hidden layers, and the shape of "balls" is transformed with weights and biases. Finally, because of its continuous property, the sigmoid or hyperbolic tangent function helps extract the digital results.

Even though the forward propagation of RHN is similar to MLP, the difference between them is the key to the improvement. The way to train the RHN is in the direction from the visible layer to the hidden layer, and from the hidden layer to the visible layer, then to the hidden layer, followed by a final propagation to the visible one (which can be assumed to contain 5 layers), which can be stated as "one feedback" involved, and it's no longer MLP. The number of "layers" in the RHN can be as many as we desire, depending on how many

feedback(s) you want. Hence the RHN could become a real deep network. The training of this network by using Simultaneous perturbation stochastic approximation is better than using backward propagation.

## 2.4 Related Dataset used in the Experiment

### 2.4.1 XOR problem and dataset

Exclusive or, also called XOR, is one of the classic problems solved by artificial neural network. It is a classification problem and one for which the expected outputs are known in advance. Therefore, a supervised learning approach is appropriate to be used.

| input 1 | input 2 | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 2.2: XOR problem

XOR appears to be a very simple problem, because the data is quite simple, shown in table 2.2. However, Minksy and Papert (1969) showed that this was a big problem for neural network architectures back into the 1960s, known as perceptrons. Initially, researchers failed to achieve a good by using a linear propagation function result, which distinguish incorrectly over half testing cases. Then, they realized that a nonlinear activation function (e.g., sigmoid function) helps overcome the limitation and create right decision bound-

aries. Additionally, another important creation to tackle this problem is the utilization of the backward propagation completed by gradient descent, which follows the downward trend of the error surface by obtaining the slope of the concave at each step. [27]

A widely-used MLP for XOR problems is a two three-layer one, which contains a hidden layer with two hidden nodes. However, the difficulty could be increased by using just one hidden node in that hidden layer, which is included in the experiment of this work.

### 2.4.2 iris dataset

The Iris flower data set, also known as Fisher's Iris data set, is a multivariate dataset introduced by the British statistician, eugenicist, and biologist Ronald Fisher in his 1936 paper [13]. The one used in the experiment includes three classes, which are "setosa", "versicolor", and "virginica" (i.e., they are also named "Category 0", "Category 1", and "Category 2"), and each class contains 50 samples with four dimensions, which are sepal length, sepal width, petal length, and petal width. All the data in the dataset are real and positive values.

The iris dataset is famous as an entry level dataset for testing the machine's classification performance, which can be access straightforwardly by loading the machine learning package Scikit-learn (shorten for "sklearn") in Python.

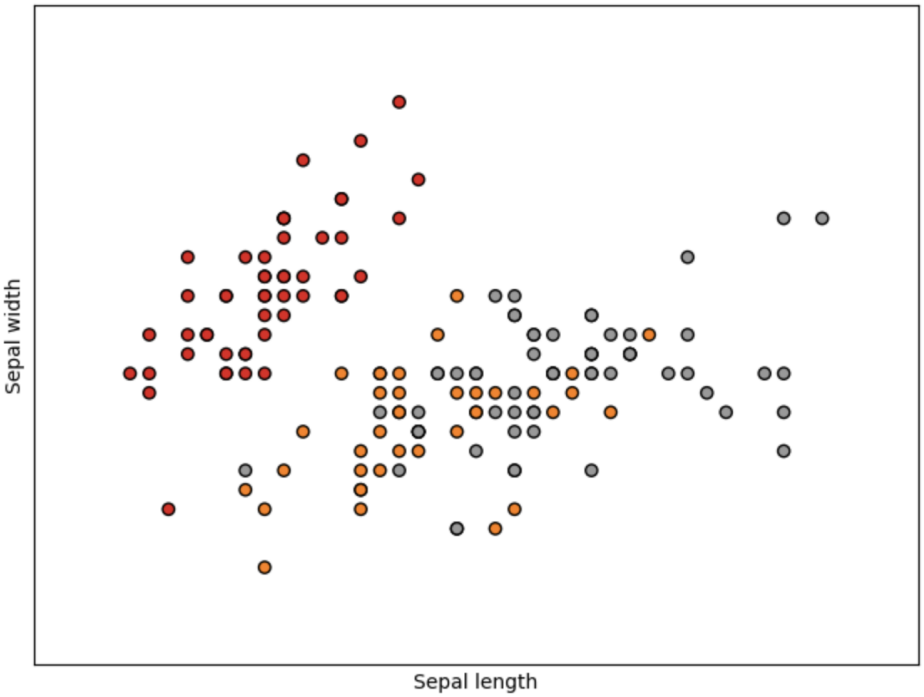



Figure 2.4: The visualization of iris dataset

### 2.4.3 Patterns for testing associative memory

For displaying the results from implementing the associative memory, some images formed by binary-representation matrices are prepared to be tested. Each image consists of 5 * 7 binary pixels, so there are 35 elements in every input vector, which are the original data in the memory. In order to observe whether the network working along with the ready-to-be-test algorithms can reconstruct the distorted data, five noisy pixels are randomly chosen from the original data (i.e., the hamming distance between the distorted image and the original image is 5).

The bigger the patterns are, the difficult the task is. Therefore, three cases (with a

graduate growth of the memory capacity) are used in the experiment. The 4-pattern case is to memorize the A, U , T, and S characters; the 10-pattern case uses the number from zero to nine; the 26-pattern case is represented by the alphabet.

Figure 2.5: The original data for reconstruction

# Chapter 3

# Improved aGBPSO & aPBPSO Algorithms

## 3.1 Underlying Theory

Based on many researchers' works in the field of mathematics, science, and engineering, either the gradient-information-related SPSA algorithm or the gradient-free PSO algorithm has a vital disadvantage — the problem of converging to the global optima.

The standard SPSA optimization method suffers from the lack of proof for global convergence. As the previous works ( [8], [14], [25] etc.) stated, the global convergence does exist in classical stochastic approximation (SA) algorithm, e.g., the finite difference SA (FDSA). However, the standard SPSA is not on the same page. Applying the standard SPSA to the calculation of the objective function evaluation can reduce a considerable amount of computation efforts, especially if the computation has to be simulated on the entire hardware system due to the usage of only two measurements (the detailed description

is in Chapter 2). The standard SPSA losses the concrete mathematical proof for achieving the global optima, resulting from its computational efficiency advantage. In every recursion of the FDSA, noise is injected by the calculation of the standard gradient. According to [8] and [14], the standard SPSA, contrarily, cannot take advantage of the existing global convergence theory of standard stochastic algorithms.

[17], in 2008, established two theorems with and without injected noise, showing the standard SPSA's ability to find the global solution of the objective function. The two developed theorems mathematically aim to help the standard SPSA holds a solid background that shows the ability to be a global optimizer. Following their descriptions, the two theorems are different from each other, distinguished by whether a noise is intentionally injected or not. But in essence, both of them are based on noise injection. They considered that SPSA has already been injected noise, called "effective noise," without manually injecting additive noise. Therefore, if we give up the noise-based theory, the global optimization of SPSA could still be questionable.

A significant number of experiments have shown that some implementations have undergone the problem of falling into the local optima when working with the standard SPSA. Therefore, the standard SPSA has been known as a tool for converging to a local minimum (or maximum) of the objective function in many cases. How to exploit its virtue but avoid its weakness is worthy of studying.

The basic PSO is another well-known algorithm for solving high-dimensional global optimization problems with the non-differentiable fitness function. Despite this, the basic PSO is also demerited by the convergence issue. (i.e., no general convergence theory applicable to practical problems)

[30] proved a close relationship between the velocity of particles in the swarm and

convergence of the optimization problem addressed by the basic PSO. They stated that the premature convergence (i.e., the results get trapped into the local optimal point) results from the decrease of particles' velocity in the searching space, which drops down the fitness momentum of the entire swarm. Their analysis with matrix implies that the velocity of the particles will finally be stable or extremely close to zero with the convergent PSO as desired.

Additionally, the structure of the basic PSO determines that the direction of updating the global best position could overwhelm the fitness of other particles, which means the global best particle is able to direct the entire swarm and hence to cause the yielding of some similar particle's position with deficiency of diversity among particles. As mentioned in [11], the questionable updating equation is the factor leading to this circumstance.

Tuning the parameters properly to balance the ability of exploitation and exploration for the swarm is critical. Those parameters include the number of the particles (i.e., the size of the swarm), the number of the epoch, the values of cognitive constant and social constant for velocity, the value of inertia weight, the learning rate, and so on, which vary from problems to problems, meanwhile, definitely complicate the optimization process. [26] also uttered the importance of initializing the position of particles. A good sampling position can lead to better performance for jumping out of the local minimum (or maximum).

Even though many useful variants have been developed to improve the basic PSO performance, the discovery of this algorithm will continue.

Obviously, implemented either the standard SPSA algorithm or the basic PSO algorithm alone for solving the nonlinear global optimization problems individually has a severe drawback that is no guarantee for the global optima.

## 3.2 The Previous Works on Improvements

Based on the previous thoughts, combing the standard SPSA with the variant of PSO could be interesting. [23], [15], and [11] have proposed some ideas of the combination option.

[23] stacked two algorithms together, with the standard SPSA first followed with the basic PSO. More specifically, this means partial particles are initialized by the standard SPSA and feeding them together with the rest randomly generated particles to the basic PSO for optimizing further. A more optimal initial swarm can be generated by applying the SPSA to some random-generated particles. To avoid overly disturbing the randomness of the basic PSO, only a part of the particles initialized by the basic SPSA.

There is an essential measurement of this algorithm: the ratio of the number of copies of the solution determined by SPSA with the total initial swarm size. They tested this ratio by using the specific objective function defined by themselves. The ratio varied from 0.2 to 0.8 with 0.1 as the increment while the other parameters are constant. The experiment revealed that the most suitable solution was initializing half the swarm by the standard SPSA.

Two algorithms in [15] applied simultaneous perturbation to calculate the gradient information. The authors added this information to the velocity updating procedure at every optimization iteration of the basic PSO, which intent to improving the ability of the swarm particles' exploration. They considered that the basic PSO updates the estimators (e.g., particles' position) only based on macro information (i.e., the information about particles' movements). Therefore the local information, like gradients for each step, is absent, contributing to direct the swarm to the global optimum. Meanwhile. They did not want to ruin the derivative-free property of the basic PSO, so the simultaneous perturbation method is chosen.

More elegant algorithms add the standard SPSA to the PSO only when the standard SPSA can guide the particles towards a better global searching result were proposed in [11]. The authors of thought that the global best position updating equation is non-efficient because two velocity components — social components and cognitive components — are forced to be quiet. And if the global best position is fallen to the local optimum, all the particles cannot survive. In the papers they researched, there is one shown that creating an "artificial global best particle" (aGB) at each iteration could be helpful for guiding the global best position to a better fitness direction. [11] twisted the originally proposed aGB. When the aGB performs better than the global best obtained from the movement of the particles in the basic PSO (i.e., leads to a smaller error associated with the objective function), the global best position will be replaced by the aGB. They aimed to solve the poor global position update equation, meanwhile keep other things in the PSO integrally as the conventional way.

## 3.3 The Previous Works' Drawbacks

The drawbacks of the previous works, all three algorithms (i.e., the SPSA-PSO, SPPSO, SAD PSO) are discussed as follows.

The SPSA-PSO only optimizes the swarm at the initialization but does not change anything in the basic PSO. Therefore, the problems with only applying the PSO still exits. Besides, in our experiments, [23]'s parameters setting is useless.

The algorithms in [15], from their perspective, can exploit the local information therefore the global convergence has a better tendency by integrating the local gradients of the particles and swarm behavior (i.e., the original swarm's fitness) together. However, [11] pointed out that [15] could mess the fitness of PSO particles up because of the insertion of

gradient information at each iteration. By implemented [15]'s methods, the results prove that the related statement in [11] is correct. The SPPSO works in some cases but not in all conditions.

It does not means that the SAD PSO in [11] is perfect. The thought of only optimizing the global best position is feasible, but the issue is how they calculated the gradient information. Instead of using the standard SPSA, they chose the one-sided variant of it (i.e., they set $\theta+ = p_g(t)$, and naturally $\theta- = p_g(t) - 2*ck*\Delta k$). It helps save the computational power relatively but scarifies the accuracy somehow. Another detail that worth considering is the iteration of operating the standard SPSA. In every epoch of the PSO, the current epoch's number is shared to be the number of the current iteration of the standard SPSA. That means, if the current epoch is 100, the iteration of the standard SPSA at this time is also 100. This action could lead to a slow convergence at the beginning.

The improved algorithms have verified the problem, at the same time, trying to design better strategies.

## 3.4 The Proposed aGBPSO & aPBPSO Algorithms

In this work, two improved algorithms are proposed. One is the combination of [23] and [11], and another one uses the basic SPSA at each iteration to optimize every particle's position. The PSO plays a role similar to the global searching technique, and the standard SPSA is more like a deep searching technique. For example, if you want to search the steepest mountain on the planet, first you can search the five continents, and then explore each continent to trace the result.

The full name of the aGBPSO is artificial global best particle swarm optimization,

and that of the aPBPSO is artificial personal best particle swarm optimization. The implication of "artificial" is that intentionally using the gradient information to improve the optimization result. Here, the gradient information is derived from applying the standard SPSA.

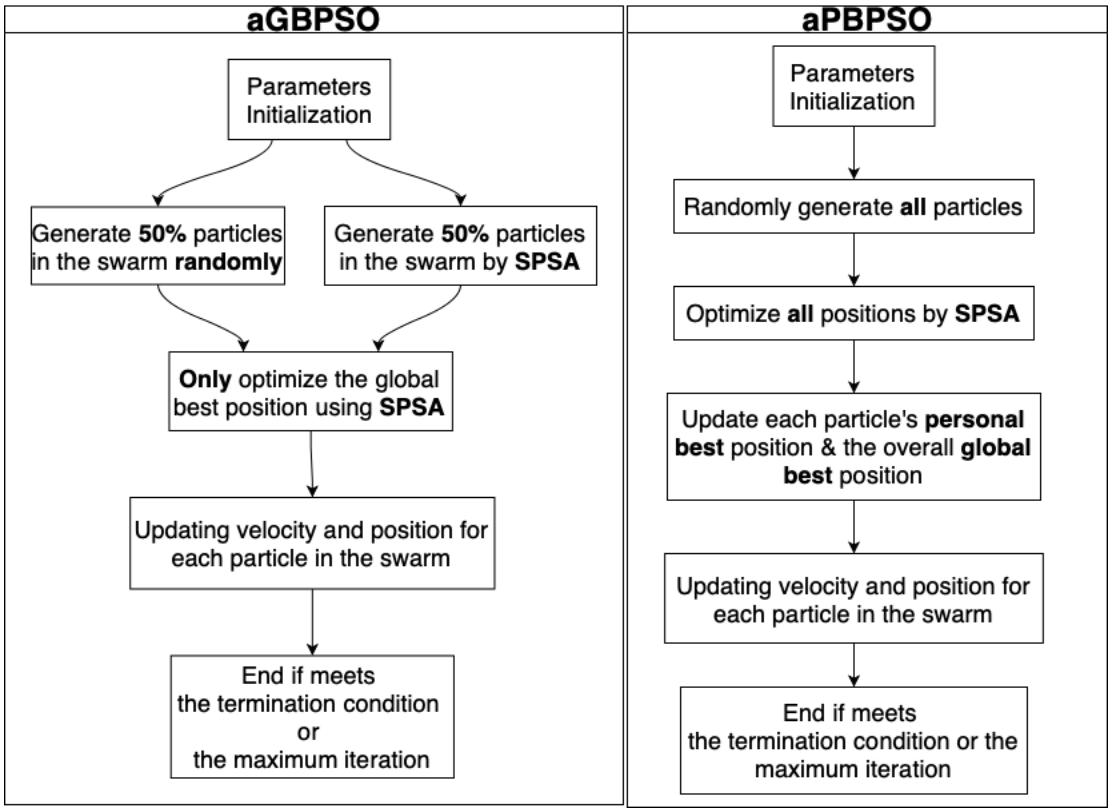



Figure 3.1: Simple aGBPSO and sPBPSO workflow

### 3.4.1 The aGBPSO approach: improving the previous works

The first proposed aGBPSO algorithm combines the ideas of [23] and [11]. In this approach, optimizing half of the swarm using the standard SPSA first at the initialization, then the

Table 3.1: Parameters in the proposed aGBPSO and aPBPSO algorithms

| Parameters | Description |
|---|---|
| $N$ | Optimization iteration |
| $M$ | Particle number |
| $P$ | Epoch of perturbation vector's optimization |
| $Deltak$ | Perturbation vector |
| $x_i(t)$ | Particle position at time step t |
| $v_i(t)$ | Particle velocity at time step t |
| $p_i(t)$ | Best individual particle position |
| $p_g(t)$ | Best swarm position |
| $y_i(t)$ | Result of objective function evaluation |
| $y_i^{best}$ | Best particle error for particle $i$ |
| $y_g^{best}$ | Best swarm error |
| $c_1$ | Cognitive velocity parameter |
| $c_2$ | Social velocity parameter |
| $r_1, r_2$ | Random numbers between 0 and 1 |
| $\omega$ | Decreasing linear inertia weight |
| $lr$ | Particles' learning rate |
| $X_{max}$ | Bound of particles' position |
| $V_{max}$ | Bound of particles' velocity (here equals $X_{max}$) |

global best particle is optimized by the standard SPSA at each iteration.

As the research results shown in [23], applying the standard SPSA to 50% particles in the swarm and leaving the rest 50% particles as their random state before implementing the PSO can generate the best consequence overall. In order to keep the generosity of the algorithm, the parameters of the standard SPSA should be set for suitable to solve many optimization problems, and the variables in the standard SPSA calculation should be integrated appropriately into the PSO structure (do not disturb the random fitness of the particles). After testing the standard SPSA with various sets of parameters (including the parameters setting in [15], [23], [11], and [16]), the most optimal set of parameters is chosen and illustrated in the tables.

Although the standard SPSA does well in finding the local optima within a specific searching scope, it cannot guarantee a better result. The notation $gbest_{spsa}$ is assumed to represent the result of the global best particle optimized by the standard SPSA. If $gbest_{spsa}$ does not make progress compared to the current global best position at the current iteration, the current global best will not be replaced by $gbest_{spsa}$ (i.e., the global best will stay in the best position before implementing the calculation of the standard SPSA to it).

**Algorithm 3.1** The Proposed aGBPSO

Initialize the optimization network or system
Set $N, M, P, \alpha, \gamma, a, c, bounds, c_1, c_2, \omega, lr$
**while** $i \leftarrow 1$ *to* $M$ **do**
    Generate $i$ particle's position randomly
    **if** $i\%2 == 0$ **then**
        Generate a perturbation vector Deltak
        Calculate $x_i(0)_+, x_i(0)_-$
        Calculate $y_i(t)_+ - y_i(t)_-)$
        Obtain the current particle position $x_i(0)$ based on algorithm 2.2
                ▷ Algorithm 2.2 is the standard SPSA algorithm
    **end if**
    Generate particle velocity $v_i(0)$
    Calculate the best particle error $y_i^{best}$ from objective function
    $i = i + 1$
**end while**
Set the best swarm position to the current estimate parameters
Set the best swarm error $y_g^{best} = y_0^{best}$

Optimization

**while** $t \leftarrow 1$ *to* $N$ **do**
    **while** $i \leftarrow 1$ *to* $M$ **do**
        Evaluate objective function $y_i(t)$
        **if** $y_i(t) \leq y_i^{best}$ **then**
            $y_i^{best} = y_i(t),\ p_i(t) = x_i(t)$

6: **end if**

7: **if** $y_i^{best} \leq y_g^{best}$ **then**

8: $y_g^{best} = y_i^{best},\ p_g(t) = p_i(t)$

9: **end if**

10: $i = i + 1$

11: **end while**

12: Obtain the optimized global best position $temp_{p_g(t)}$ based on 2.2

13: **if** $temp_{y_g^{best}} < y_g^{best}$ **then**

14: $p_g^{best} = temp_{p_g(t)}$

15: $y_g^{best} = temp_{y_g^{best}}$

16: **end if**

17: **while** $i \leftarrow 1$ *to* $M$ **do**

18: $r_1 \in [0, 1], r_2 \in [0, 1]$

19: Calculate cognitive velocity: $c_1 r_1 (p_i(t) - x_i(t))$

20: Calculate social velocity: $c_2 r_2 (p_g(t) - x_i(t))$

21: $V_{new} = w(t) v_i(t) + c_1 r_1 (p_i(t) - x_i(t)) + c_2 r_2 (p_g(t) - x_i(t))$

22: **if** $V_{new} > V_{max}$ **then**

23: $V_{new} = V_{max}$

24: **end if**

25: Calculate new position: $x_i(t + 1) = lr * x_i(t) + v_i(t + 1)$

26: $\omega(t + 1) = (\omega(0) - \omega(N)) \frac{(N-t)}{N} + \omega(N)$

27: $i = i + 1$

28: **end while**

29: $t = t + 1$

30: **end while**

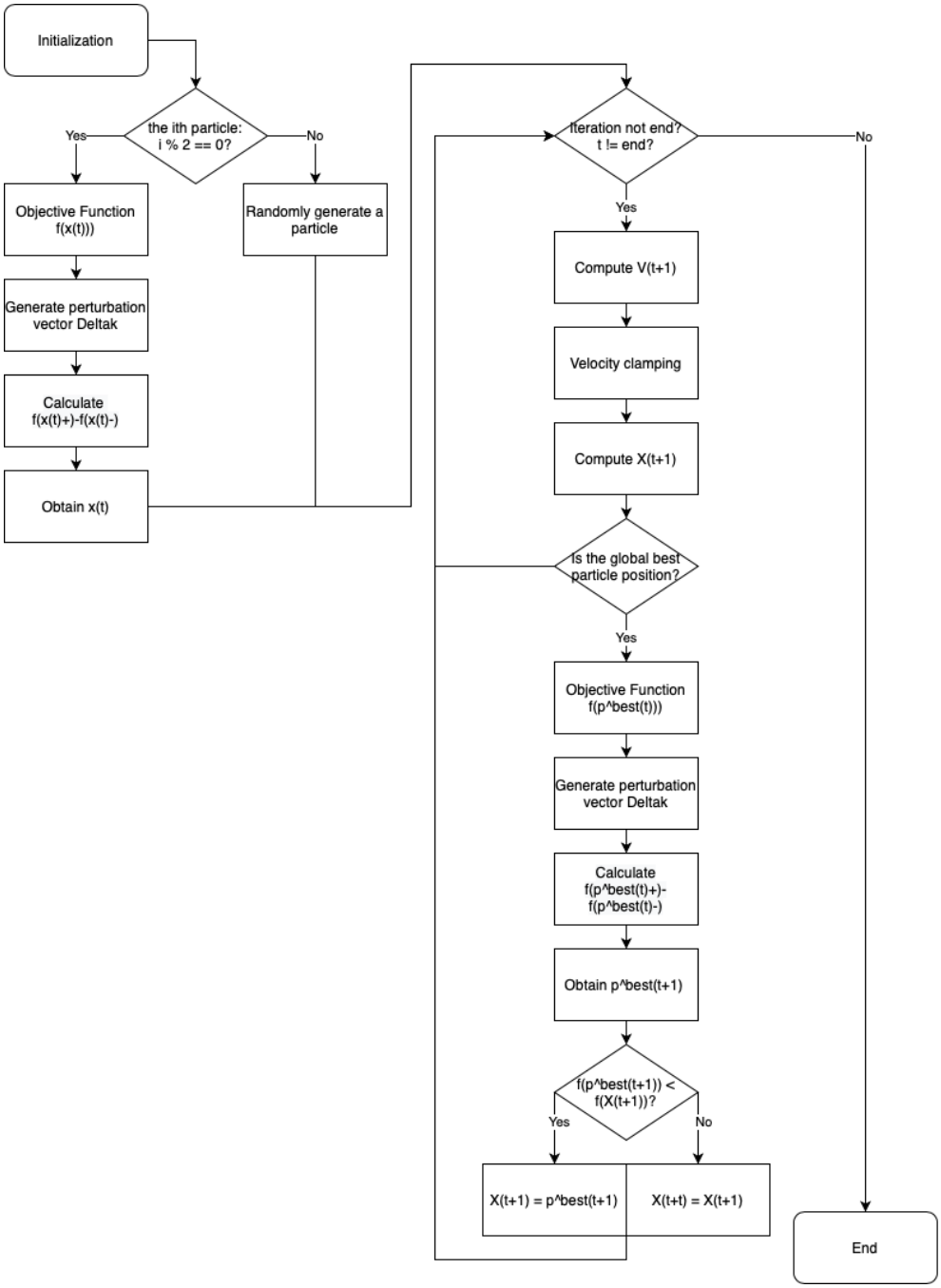



Figure 3.2: aGBPSO detailed workflow diagram

### 3.4.2 The aPBPSO approach: improving searching ability

The second proposed aPBPSO is the algorithm that plugs the standard SPSA into the position of all the particles (i.e., not only into the global best position) in the PSO variant at each iteration. The variant is the linear inertia weight PSO with velocity clamping. The differences between aGBPSO and aPBPSO need to explain from two aspects.

On the one hand, all the particles in the swarm are staying in their random state without further optimization by the SPSA in the aPBPSO. On the other hand, the aPBPSO expands the influence of gradient information (i.e., the aPB) to every particle at each iteration. Simultaneously, aGBPSO only uses the artificial global best (aGB) calculation for the global best particle.

The standard SPSA is implemented to every particle's position to "locally" search for a relatively better personal solution than the current one during the whole optimization time. Specifically, it is improved by adding "the local searching tool" (i.e., the standard SPSA) into each iteration of the PSO to enhance the searching ability of the algorithm.

Theoretically, the aPBPSO is more likely to find the global optima because of the thoroughly deeper optimizing procedure. The bigger the target searching scope is, the better the result could be yield. But based on [11]'s study, the opposite scenario could appear. That is, excessive use of gradient information leads to particles getting lost and unable to find the optimal direction

The general rule for updating the particles here is still the same as what is formulated for aPBPSO: if the position of the particle is optimal after optimized by the standard SPSA, then its position will be replaced; otherwise, the current position of it will be maintained.

The aGBPSO and the aPBPSO share the same parameters in the experiments since they are basically derived from the same theories. Plus, it is more straightforward to

compare the results obtained from both of them.

**Algorithm 3.2** The Proposed aPBPSO

Initialize the optimization network or system
Set $N, M, P, \alpha, \gamma, a, c, bounds, c_1, c_2, \omega, lr$
**while** $i \leftarrow 1$ *to* $M$ **do**
    Generate $i$ particle's position randomly
    Generate particle velocity $v_i(0)$
    Calculate the best particle error $y_i^{best}$ from objective function
    $i = i + 1$
**end while**
Set the best swarm position to the current estimate parameters
Set the best swarm error $y_g^{best} = y_0^{best}$

Optimization

**while** $t \leftarrow 1$ *to* $N$ **do**
    **while** $i \leftarrow 1$ *to* $M$ **do**
        Generate a perturbation vector $\Delta k$
        Calculate $x_i(t)_+, x_i(t)_-$
        Calculate $y_i(t)_+, y_i(t)_-$
        Obtain the optimized global best position $temp_{x_i(t)}$ based on algorithm 2.2
        **if** $temp_{y_i(t)} < y_i^{best}$ **then**
            $x_i(t) = temp_{x_i(t)}$
            $y_i(t) = temp_{y_i(t)}$
        **end if**
        Evaluate objective function $y_i(t)$

12: **if** $y_i(t) \leq y_i^{best}$ **then**

13: $y_i^{best} = y_i(t),\ p_i(t) = x_i(t)$

14: **end if**

15: **if** $y_i^{best} \leq y_g^{best}$ **then**

16: $y_g^{best} = y_i^{best},\ p_g(t) = p_i(t)$

17: **end if**

18: $i = i + 1$

19: **end while**

20: **while** $i \leftarrow 1$ *to* $M$ **do**

21: $r_1 \in [0, 1]$

22: $r_2 \in [0, 1]$

23: Calculate cognitive velocity: $c_1 r_1 (p_i(t) - x_i(t))$

24: Calculate social velocity: $c_2 r_2 (p_g(t) - x_i(t))$

25: $V_{new} = w(t)v_i(t) + c_1 r_1 (p_i(t) - x_i(t)) + c_2 r_2 (p_g(t) - x_i(t))$

26: **if** $V_{new} > V_{max}$ **then**

27: $V_{new} = V_{max}$

28: **end if**

29: Calculate new position: $x_i(t + 1) = lr * x_i(t) + v_i(t + 1)$

30: $\omega(t + 1) = (\omega(0) - \omega(N))\frac{(N-t)}{N} + \omega(N)$

31: $i = i + 1$

32: **end while**

33: $t = t + 1$

34: **end while**

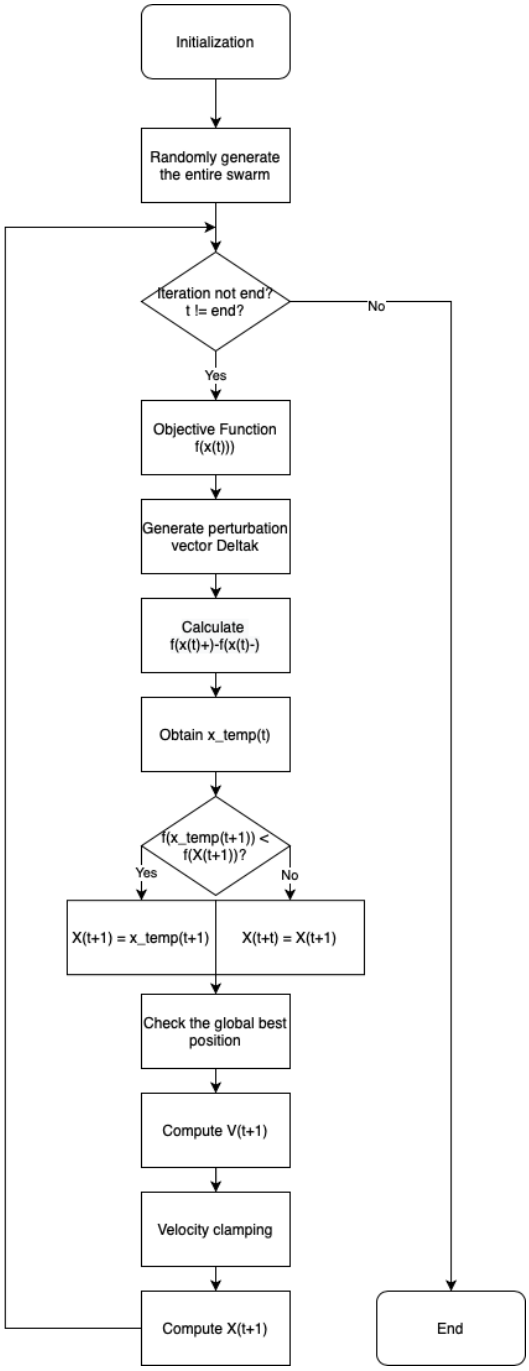



Figure 3.3: aPBPSO detailed workflow diagram

# Chapter 4

# Experimental Results of MLP Implementation

In this chapter, the proposed aGDPSO and aPBPSO algorithm's performance is compared with that of the SAD PSO, the SPPSO, the SPSA-PSO, the standard SPSA, and the PSO methods using different parameter values (i.e., suitable parameters' values for each own algorithm). The above listed three algorithms (the SAD PSO, the SPPSO, the SPSA-PSO) are developed by the author of [11], [15], and [23], respectively.

Near optimal values of parameter were used in all test cases for the above seven algorithms. In order to cover the various possible optimization systems and their essential stochastic nature, five optimization problems are considered, including XOR problem, data classification problem, and data reconstruction problems. Meanwhile, all the solutions found by the aGBPSO, the aPBPSO, and the other five algorithms are analyzed, respectively.

Examining the solutions and (sum of squared) errors of the objective functions derived

from the above seven optimization strategies, interesting results can be concluded. Optimal solutions obtained by implementing the basic PSO in terms of the value of errors in the optimizing process are undesirable and will not be discussed further. It is easy for the PSO algorithm to get stuck in local optima or skipped over the global optimum. Even though the standard SPSA performs better than the basic PSO, its performance heavily depends on the number of iteration. It shows that increasing optimization iterations could boost the searching ability of the standard SPSA, and better solutions could be achieved after adding the allowable iterations. Among all the previous improvements, the SAD PSO performs more decently than the other two. The SPSA-PSO converges quickly at the beginning but hardly keeps the steep converging trend further. Finally, the experimental results indicate that the proposed aGBPSO and aPBPSO algorithms converge to the global optima more efficiently and effectively while other algorithms are still exploring the searching space.

In the algorithms, many parameters have to set carefully. In general, the two proposed algorithms and three previous-designed algorithms share the same criteria as combining the maximum number of iterations allowed and the cut-off error $10^{-6}$. The allowed maximum number of iterations is the summation of the number of particles' optimization iteration, equal to 1000, and the number of epochs, equivalent to 100 (for all the cases except for the XOR problem). The standard SPSA performs optimization with 100000 iterations, which is the only one that no need to set the number of particles since it is not a swarm optimization scheme. For the basic PSO, the swarm size is fixed at 200; for the rest five approaches (e.g., the aGBPSO, aPBPSO, PSO, SAD PSO, SPPSO, SPSA-PSO), the swarm size varies concerning the problems, ranging from 20 to 50. The inertia weight in them linearly decreases from 0.9 to 0.4. Besides, the recommended values for $\alpha$ and $\gamma$ as 0.602 and 0.101 are used, and A is calculated by $A = \frac{opt\_iter}{10.0} = 100.0$, as well as gain $a$ and gain $c$ are set to 1.0, which are purposefully fixed for the standard SPSA, the proposed two

algorithms, and the previous three algorithms. During each run, the optimizing process could be terminated when the error calculated from the objective function drops below the cut-off error, which is considered the global optima reached. At the same time, there is no pre-terminated setting for the standard SPSA and the PSO.

(Statistical) Analysis of each experiment includes:

- Summarize the sum of squared error (i.e., SSE) for the objective functions at each iteration.
- Evaluate the intermediate SSE generated from the aGBPSO and aPBPSO.
- Demonstrate the results generated from a single run (or multiple runs).

Analyzing from various angles can obtain different conclusive information and relatively robust conclusion. The line graphs of SSE illustrate the trends of convergence at each iteration. It depicts all those seven approaches:

- The aGBPSO is in a blue line.
- The aPBPSO is in an orange line.
- The SAD PSO is in a green line.
- The SPPSO is in a red line.
- The SPSA-PSO is in a purple line.
- The standard SPSA is in a brown line.
- The basic PSO is in a pink line.

The intermediate errors' line plots (i.e., the line chart of the sum of square errors in the intermediate calculation) represent the errors of the SPSA optimization step within the aGBPSO and aPBPSO algorithms. The blue line presents the dropping tendency of the errors which are after the standard SPSA optimizes the current global best particle's position at each iteration in the aGBPSO. The orange line stands for the trend of the average of the errors computed by the position after applying the standard SPSA (no matter the new position is better than the current position or not) to all the particles at each iteration in the aPBPSO.

More details will explain in the following sub-sections in this and the following chapters.

## 4.1 Solving XOR problem

In order to solve the classic XOR problem (described in Chapter two), an artificial neural network is implemented. This MLP is calculated in different ways. Instead of using the standard gradient descent techniques such as the back-prop algorithm, the proposed two algorithms and above-mentioned five algorithms are applied.

### 4.1.1 Two-hidden-nodes application

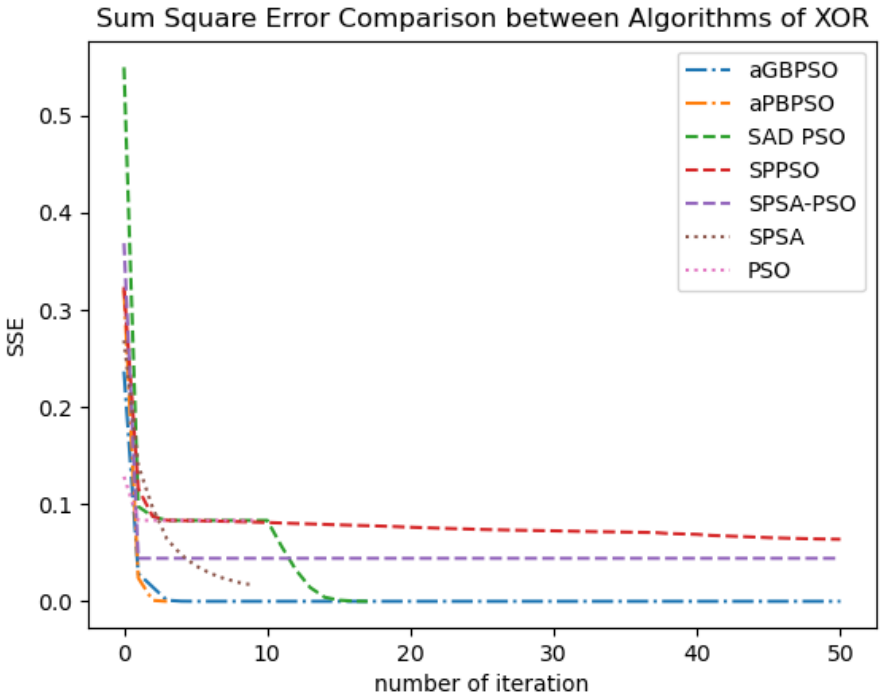
Seven algorithms (the aGBPSO and aPBPSO, the SAD PSO, the SPPSO, the SPSA-PSO, the standard SPSA, and the basic PSO with linear reduction of inertia weight and velocity clamp) are applied to tackle the XOR problem. They use the value of parameters described in the table. A three-layer MLP is implemented as follows: two nodes in the input layer, two hidden nodes in the hidden layers, and one node in the output layer. The detailed parameters setting is shown in table 4.1.

Table 4.1: Parameters setting in the XOR problem

| | Improvements | SPSA | PSO |
|---|---|---|---|
| Layers' dimension of the MLP | 2,2,1 | 2,2,1 | 2,2,1 |
| Number of optimization iteration | 1000 | 10000 | |
| Gain a | 1.0 | 1.0 | |
| Gain c | 1.0 | 1.0 | |
| $\alpha$ | 0.602 | 0.602 | |
| $\gamma$ | 0.101 | 0.602 | |
| Bounds | -10.0 to 10.0 | -10.0 to 10.0 | |
| Number of particles in swarm | 20 | | 200 |
| Number of epochs (with cut-off error 1e-6) | 50 | | 100000 |
| Cognitive c | 1.49445 | | 1.49445 |
| Social c | 1.49445 | | 1.49445 |
| Learning rate | 1.0 | | 1.0 |
| Inertia weight bounds | 0.4 to 0.9 | | 0.4 to 0.9 |

### 4.1.2 Error analysis

The diagram Figure 4.1 below shows the sum square error (SSE) of three algorithms. Overall, the SSE calculated from the objective function using the MLP with all the seven algorithms has declined, which shows a convergent tendency. The aGBPSO, the aPBPSO, and the SAD PSO, and the standard SPSA drop down the SSE close to zero after around 15 iterations. The red line, the SPPSO, hardly decreases the error after the second iteration and keeps the error value about 0.1, which is the highest value. The SSE shown by the purple line representing the SPSA-PSO's performance converges at the first iteration but no improvement after that. When taking a close look, the aGBPSO and aPBPSO have incredible converging speeds and final SSE values. The blue line and orange line start with moderate error values and reach the bottom of the line chart only within one iteration. Moreover, the epoch number of the aGBPSO is set to 50. The algorithm reduces the SSE blow the cut-off error only at the third iteration (i.e., the x-axis of the line chart displays the number of iteration, and the orange line exists only before three), which means it can find the global optimum in approximately three iterations. Although the aPBPSO did not end early, the lowest SSE still proves that both the aGBPSO and the aPBPSO perform optimally in this case. Besides, the standard SPSA achieves an optimal results, which means that the fixed parameters (i.e., $a, c, \alpha, \gamma, A$) chosen for all the algorithms are appropriate since they are sharing the same setting. Therefore, an improvement made by the proposed algorithms can be clearly verified. In this situation with an extremely small dataset, the aPBPSO slightly performs better than the aGBPSO.



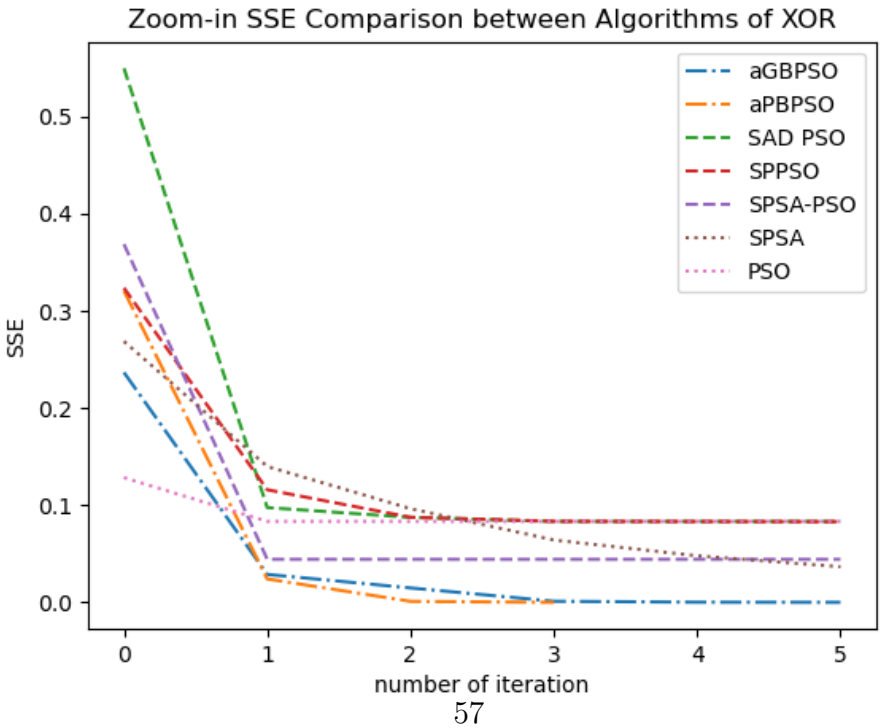






Figure 4.1: SSE of the algorithms analyzed at each iteration in the XOR problem

In figure 4.2, compared to the downward trend of the orange line, there is a bump of the blue line, but the value is stable around zero after the second iteration. It shows, despite the standard SPSA cannot guarantee the best solution always, the searching range of the particles can be moved closer to the global best optimum.

One would wonder why the aPBPSO stops early, but the intermediate error is not below the cut-off error. The reason is that the intermediate SSE of the aPBPSO is an average value from all the particles, and there would be one personal best position with the personal best error that dropped below the cut-off error.

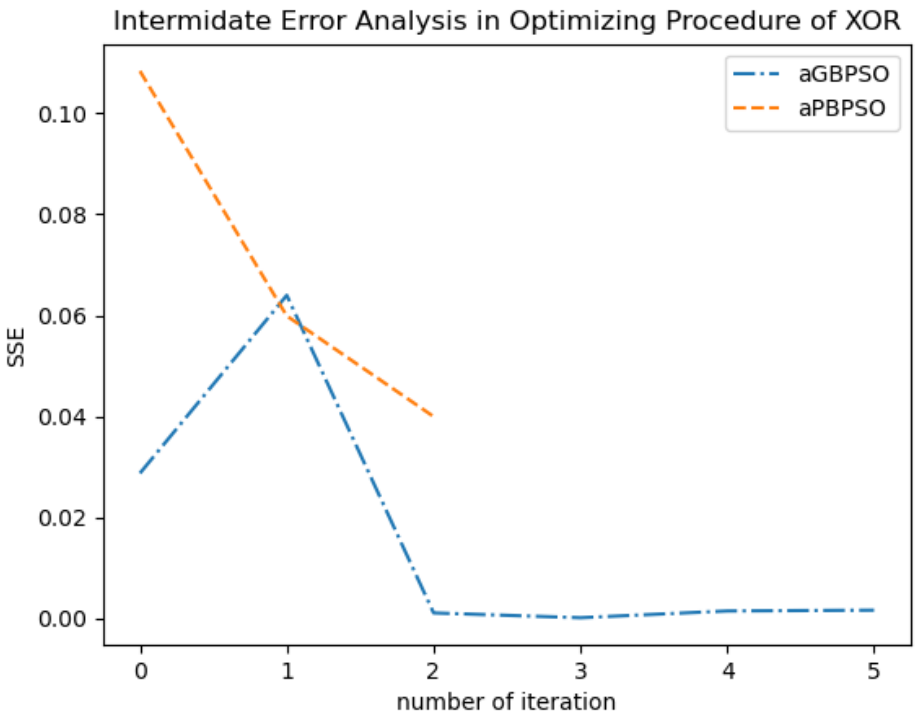



Figure 4.2: Intermediate SSE of the algorithms analyzed at each iteration in the XOR problem

### 4.1.3 Statistical analysis

All six algorithms (i.e., all seven except the PSO) can properly solve the XOR problem. In order to test their performance solidly, all the algorithms are tested over 100 runs, and the first order and second-order statistics (mean and standard deviation) of the results are analyzed. In several trials, the aGBPSO fails to solve the XOR problem, but the overall performance is still acceptable. Meanwhile, the MLP working with the PBPSO acts as an XOR logic gate perfectly. The results of 100 runs for each algorithm are as figure 4.3 and the means and standard deviations are in the table table 4.2:

Table 4.2: Mean and standard deviation of the 100 times running in the XOR problem

| | Mean | Std |
|---|---|---|
| aGBPSO | 0.93 | 0.255 |
| aPBPSO | 1.0 | 0.0 |
| SAD PSO | 1.0 | 0.0 |
| SPPSO | 1.0 | 0.0 |
| SPSA-PSO | 1.0 | 0.0 |
| SPSA | 0.87 | 0.336 |
| PSO | 0.0 | 0.0 |

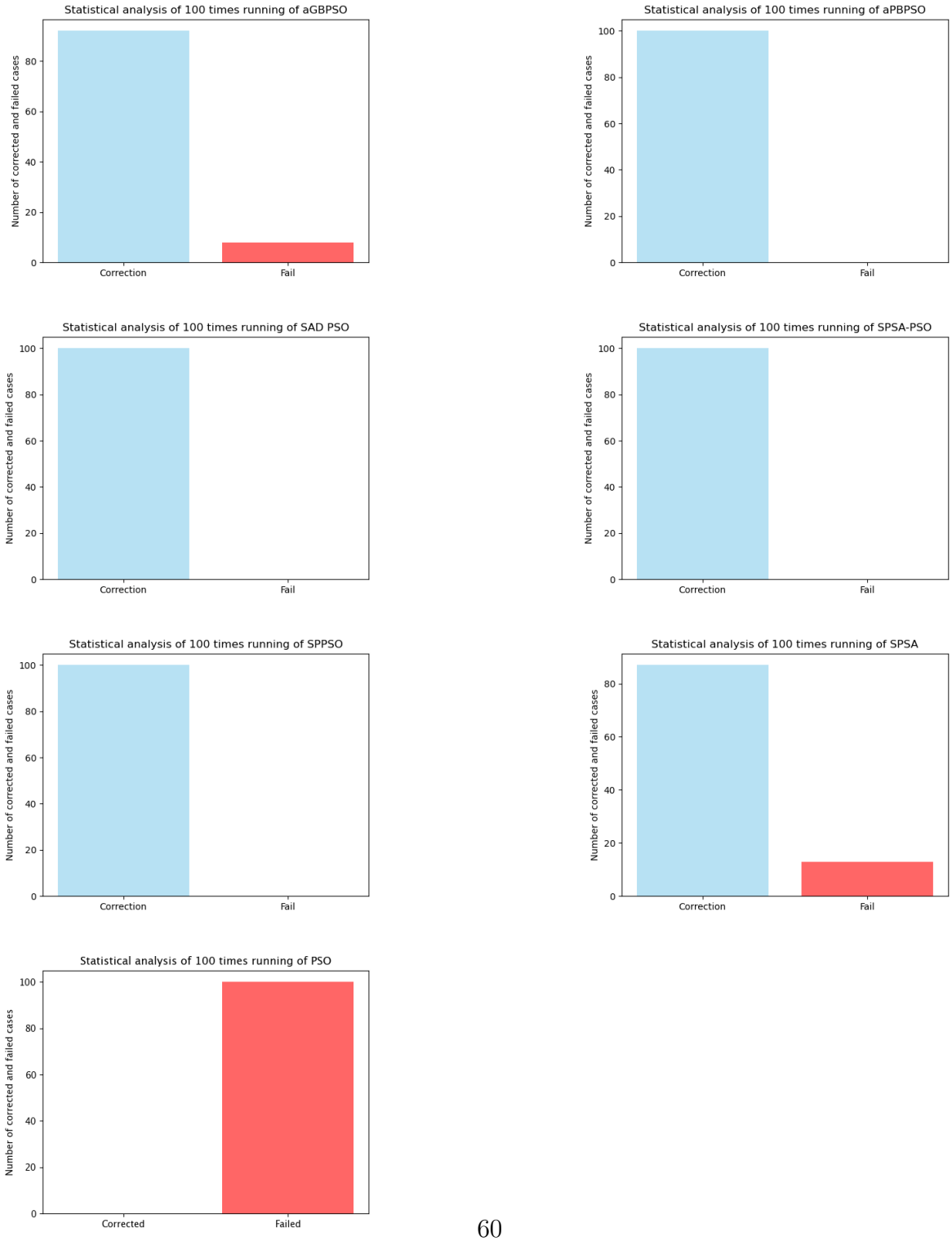





Figure 4.3: Statistical analysis of the 100 times running in the XOR problem

## 4.2 Data classification

### 4.2.1 Iris flower dataset

The Iris ower data set, also known as Fisher's Iris data set, is a multivariate dataset introduced by the British statistician, eugenicist, and biologist Ronald Fisher in his 1936 paper. [13] Since there are totally 150 samples, the ratio of the training dataset and the testing dataset is 5:1, which means there are 120 samples randomly selected to be in the training dataset, and the rest 30 samples consist of the testing dataset. For the testing dataset, "setosa" (Category 0) owns 11 samples, "versicolor" (Category 1) owns 13 samples, and "virginica" (Category 2) owns 6 samples.

Before feeding the MLP, the iris data should be scaled, since it is useful to scale the input attributes for a model that relies on the magnitude of values. This is done by removing the mean and scaling to unit variance.

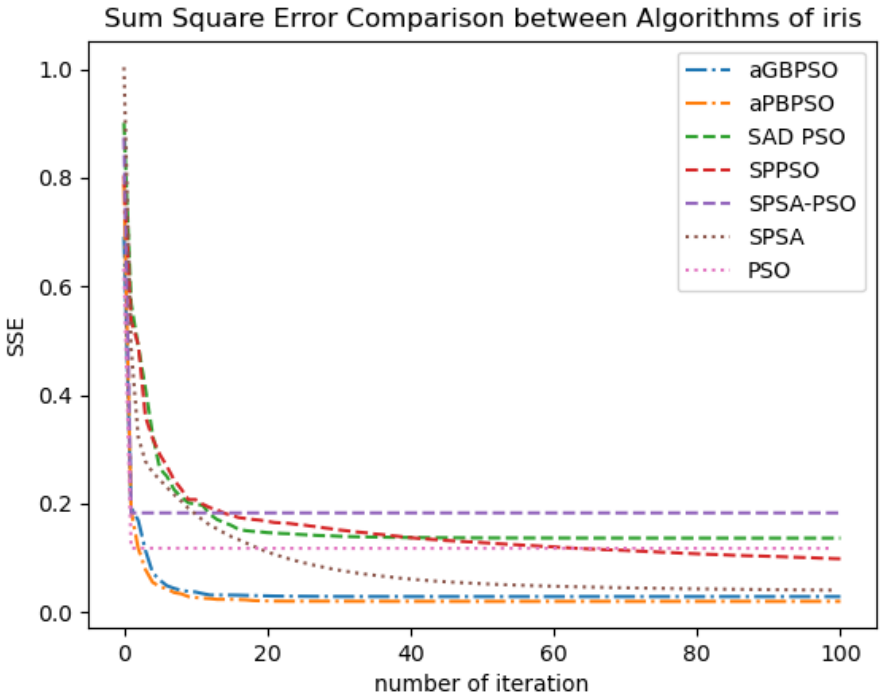
The machines designed to deal with the data classification task are the same among all the methods. A three-layer MLP with four nodes in the input layer, sixteen hidden nodes in the hidden layers, and three nodes in the output layer is implemented. The constant parameters of the seven algorithms remain the same as the setting in the previous XOR experiment.

Table 4.3: Parameters setting in the data classification

| | Modified Algorithms | SPSA | PSO |
|---|---|---|---|
| Layers' dimension of the MLP | 4, 16, 3 | 4, 16, 3 | 4, 16, 3 |
| Number of optimization iteration | 1000 | 10000 | |
| Gain a | 1.0 | 1.0 | |
| Gain c | 1.0 | 1.0 | |
| $\alpha$ | 0.602 | 0.602 | |
| $\gamma$ | 0.101 | 0.602 | |
| Bounds | -10.0 to 10.0 | -10.0 to 10.0 | |
| Number of particles in swarm | 30 | | 200 |
| Number of epochs (with cut-off error 1e-6) | 100 | | 100000 |
| Cognitive c | 1.49445 | | 1.49445 |
| Social c | 1.49445 | | 1.49445 |
| Learning rate | 1.0 | | 1.0 |
| Inertia weight bounds | 0.4 to 0.9 | | 0.4 to 0.9 |

### 4.2.2 Error analysis

The SSE line chart Figure 4.4 compares the convergence condition of the seven mentioned algorithms. The result of this experiment is similar to what is observed in the XOR problem. Because of the larger data size used in this case, the convergence of applying those algorithms slows down somehow, besides the aGBPSO and aPBPSO. Whether looking at lines in the zoom-in graph or in the original graph that has not been zoom-in, it is easy to discover their advantages of converging speed and reduced error value from the two proposed algorithms. Both of them decrease the SSE to the desired level with around five iterations, as shown by the blue line and orange line. All the other algorithms (the SAD PSO, the SPPSO, the SPSA-PSO, the standard SPSA, and the PSO) cannot perform as well as them. Either the convergence speed is too slow, or the error value is relatively high. If considering the error value reduction extent, despite the outcome of the standard SPSA is close to that of the proposed algorithms at the last several iterations, the standard SPSA converges 15 times slower. If considering the convergent speed, the SPSA-PSO and the PSO have a similar speed with the aGBPSO and aPBPSO, but the SSE remains high, especially, the SPSA-PSO yields the biggest error value.



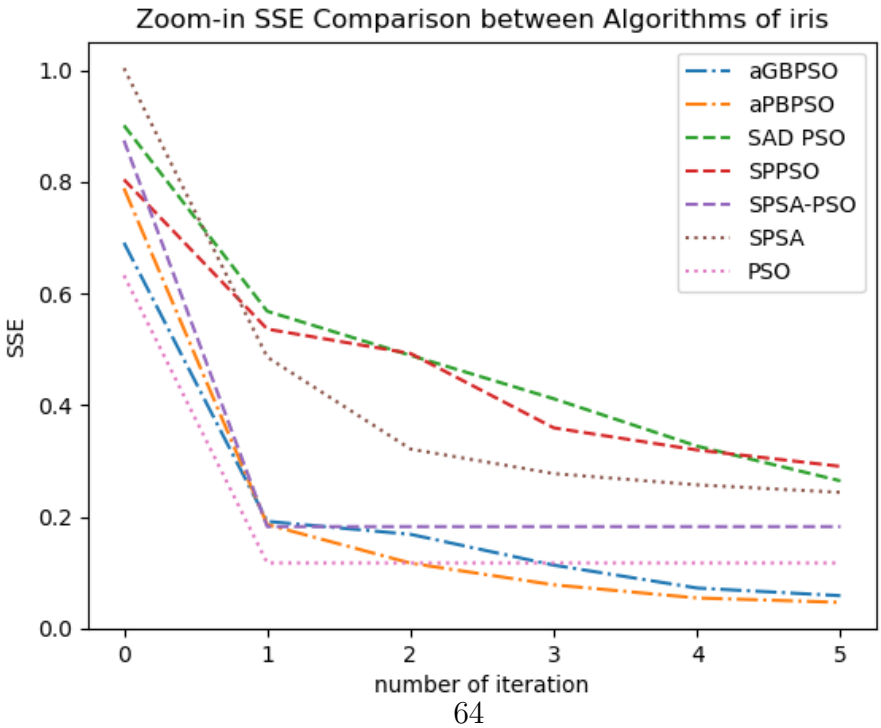






Figure 4.4: SSE of the algorithms analyzed at each iteration in the XOR problem

The two lines in figure 4.5 are similar, which means that the two algorithms perform equally well. Not only that, if checking the final SSE in the previous figure again, the lines of aGBPSO and aPBPSO almost coincide as well. According to the phenomenon, both of the two proposed algorithms are effective.

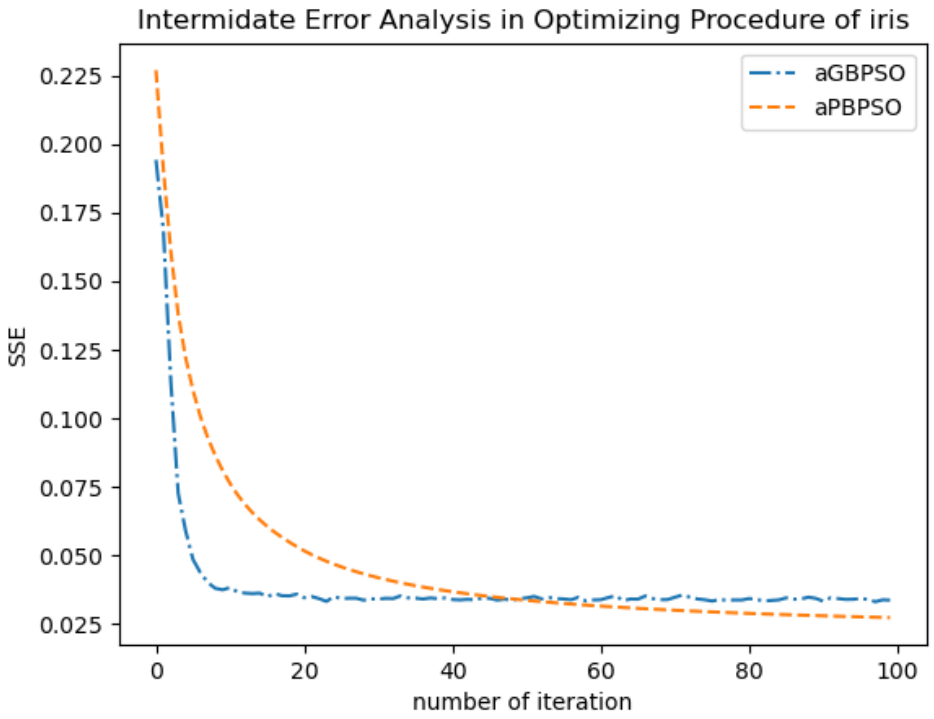



Figure 4.5: Intermediate SSE of the algorithms analyzed at each iteration in the data classification

### 4.2.3 Statistical analysis

The confusion matrix for evaluating a classifier's quality plays a key role in data classification problems, a widely used statistical analysis tool. Here, every algorithm generates a

confusion matrix (illustrated in figure 4.6, figure 4.7, and figure 4.8) based on predicting the samples in the testing dataset.

The number shown on the diagonal's square cells represents how many samples successfully classify to their corresponding label. From the first figure figure 4.6, which contains the confusion matrices generated by implementing the aGBPSO and aPBPSO, incredible results can be found. The machine with the aGBPSO flawlessly achieves the best result, with no misclassified sample at all. Based on the previous analysis of the SSE, a good result should also be detected in the aPBPSO's confusion matrix. There is only one misclassification shown in the diagram of using aPBPSO. Undoubtedly, its performance is already superior among all seven algorithms.

The other two figures figure 4.7 and figure 4.8 are the confusion matrices of the SAD PSO, the SPPSO, the SPSA-PSO, the standard SPSA, and the PSO. The classifier designed using the SPSA-PSO is the worse one with the lowest accuracy, which corresponds to its maximum error value illustrated in the SSE comparison line chart. The other four algorithms solve the data classification problem with a moderately acceptable error and precision value.

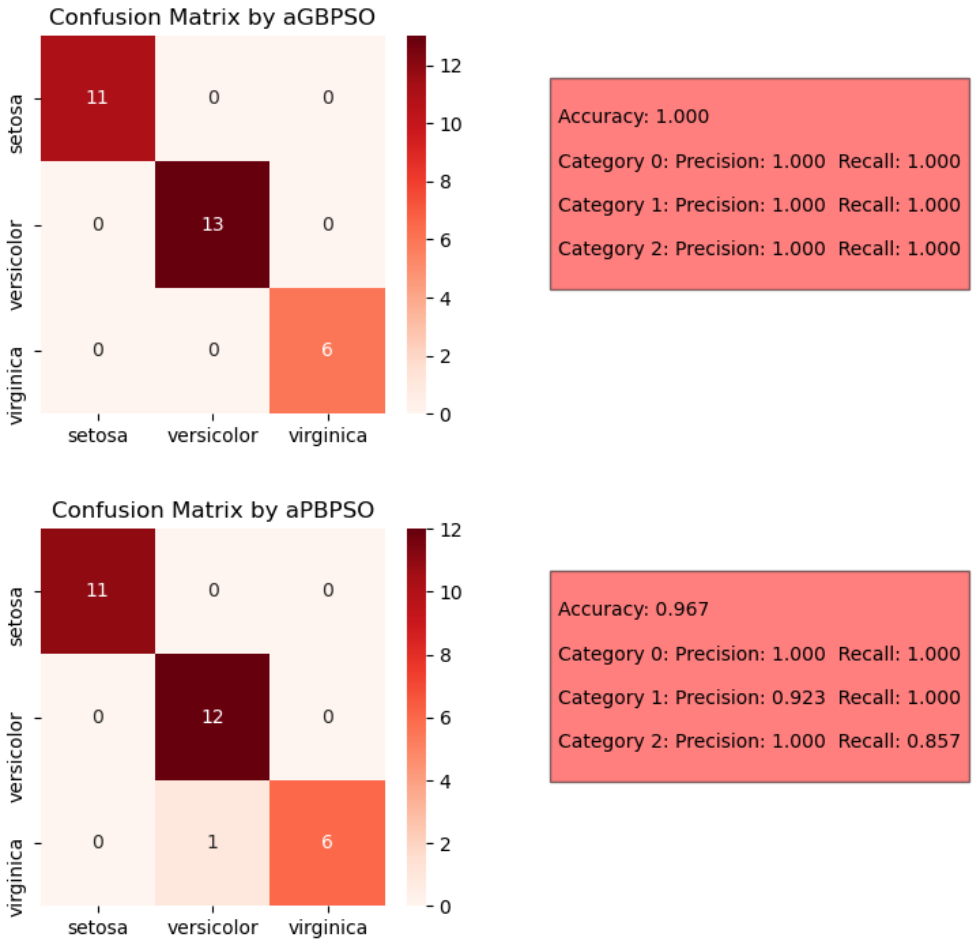



Figure 4.6: Confusion matrix yield from the aGBPSO & aPBPSO in the data classification

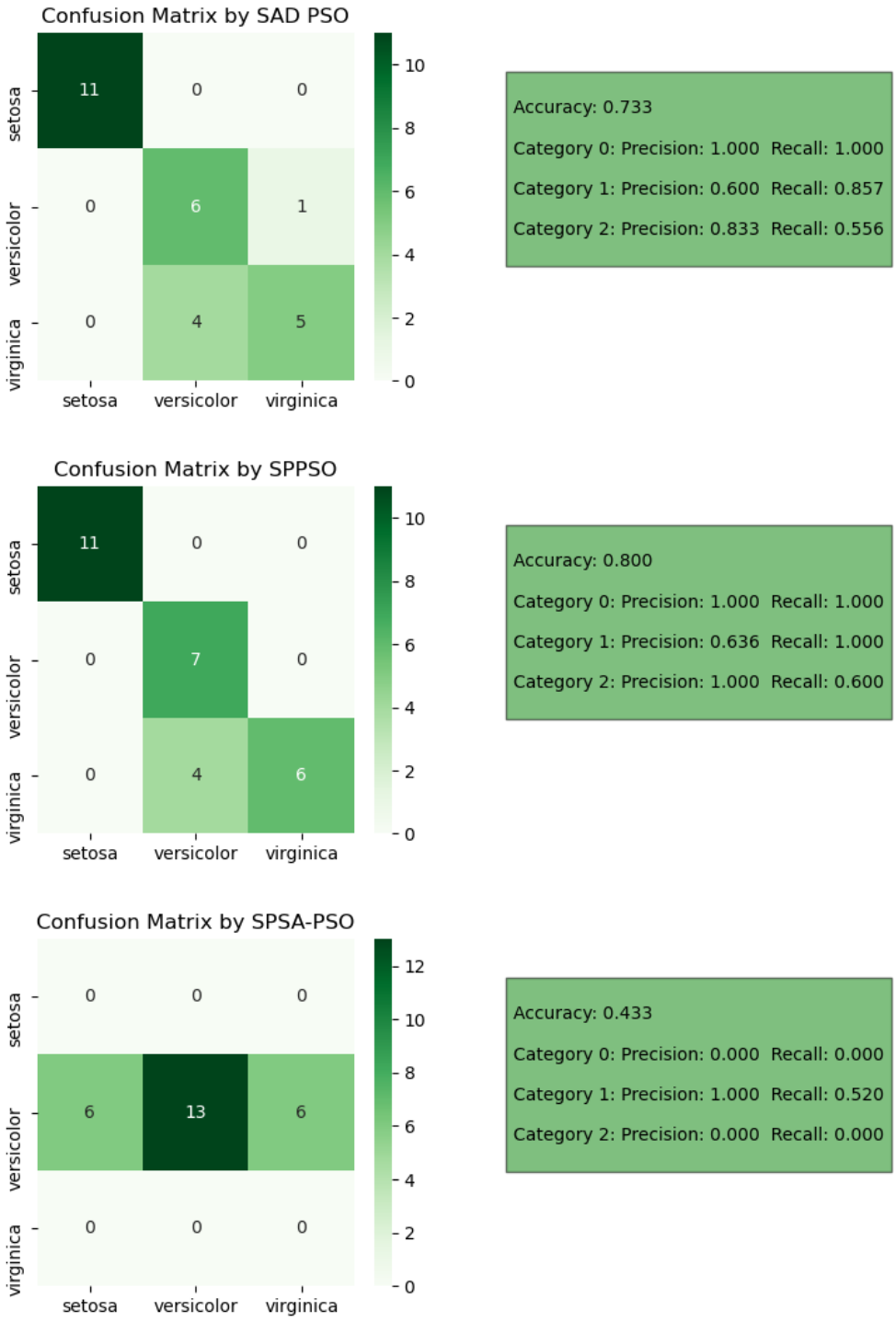



Figure 4.7: Confusion matrix yield from the previous works in the data classification

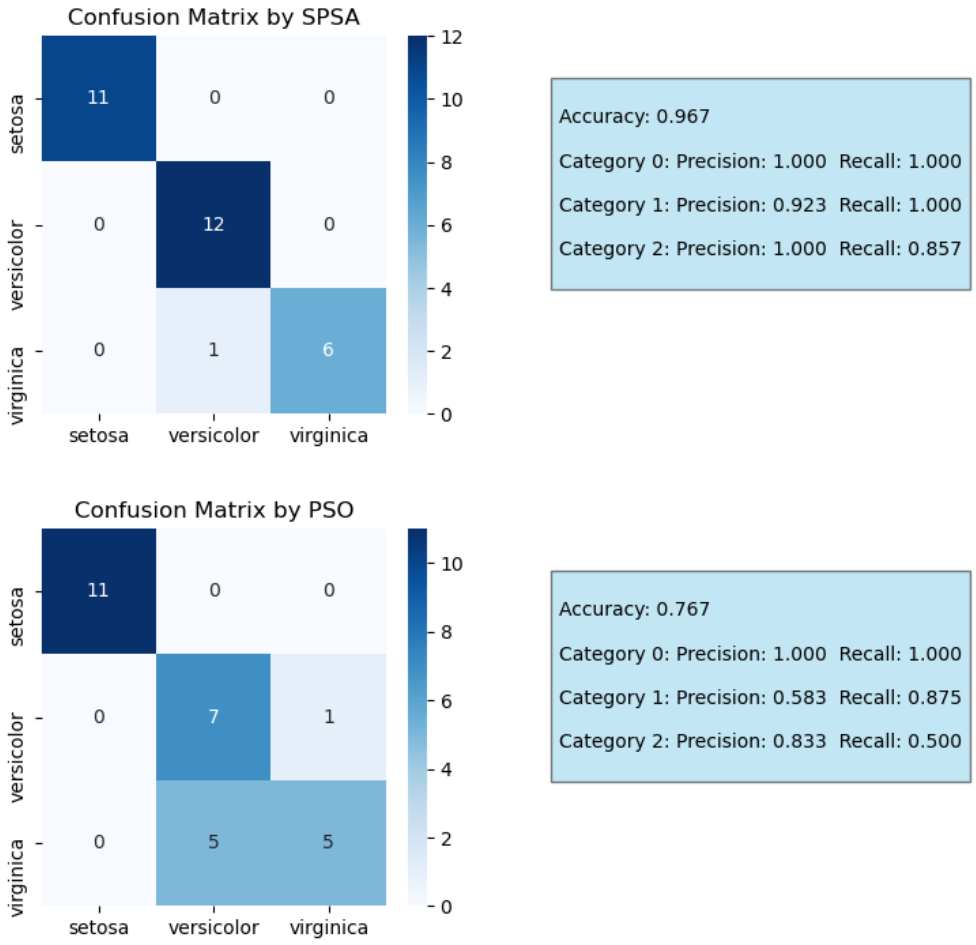



Figure 4.8: Confusion matrix yield from the SPSA & PSO in the data classification

# Chapter 5

# Experiment Results of RHN for Data Reconstruction

An innovative neural network is used here, which is the improved Restricted Hopfield Network (RHN). A detailed description of the RHN is discussed in Chapter two. All the RHN machine includes one feedback, and the nodes‘ number in the hidden layer is fixed to 30 for all the experiments. Three cases are with a different number of patterns, and all the test data are randomly distorted five pixels from the corresponding patterns in the memory.

Table 5.1: Common parameters setting in data reconstruction

| | Modified algorithms | SPSA | PSO |
|---|---|---|---|
| Number of feedback in RHN | 1 | 1 | 1 |
| Number of nodes in hidden layer | 30 | 30 | 30 |
| Gain a | 1.0 | 1.0 | |
| Gain c | 1.0 | 1.0 | |
| $\alpha$ | 0.602 | 0.602 | |
| $\gamma$ | 0.101 | 0.101 | |
| Bounds | -10.0 to 10.0 | -10.0 to 10.0 | -10.0 to 10.0 |
| Cognitive c | 1.49445 | | 1.49445 |
| Social c | 1.49445 | | 1.49445 |
| Learning rate | 1.0 | | 1.0 |
| Inertia weight bounds | 0.4 to 0.9 | | 0.4 to 0.9 |

## 5.1 Parameters' comparison

In chapter two, one of the main drawbacks of gradient descent (GD) is the "vanishing gradient problem". In this experiment, the proposed aGBPSO and aPBPSO show the ability to avoid this vital issue when applied to the high-dimensional machine, since the RHN with one feedback resembles a five-layer MLP.

The constants in the seven approaches almost keep the same as the previously defined values used in the data classification experiment. The only change in the aGBPSO, the aPBPSO, the SPPSO, and the SAD PSO is the size of the swarm. In the 4-patterns case, the swarm size is relatively small, which is only 20 particles. With the number of patterns gradually increase, the swarm size is expanded. In the 10-patterns case and 26-patterns case, the particles are added up to 30 and 50, respectively. The increase of the particles' number is in line with the scientific fact of distribution and probability. The more "birds" in the flock, the better the habitat could be discovered. Identically, the more particles in the swarm, the higher probability of finding the global optimal solution could be achieved. When implementing the SPSA-PSO in the data reconstruction experiment, the number of particles in the swarm is fixed at 200 (i.e., same with the setting of the PSO), and the number of optimization iteration (of the standard SPSA) equals 100000 (i.e., same with the standard SPSA setting). Because using the SPSA-PSO is similar to use the SPSA and PSO separately.

For the SPSA and the PSO, the parameters set for solving all three tasks in the data reconstruction problem are not changed, which is convenient to measure the performance within each case and among the entire experiments.

The selected parameters probably perform better optimization to each problem in general.

*num_birds*: Number of particles in swarm

*num_epochs*: Number of epochs

*opt_iter*: Number of optimization iteration with simultaneous perturbation (with cut-off error 1e-6)

Table 5.2: Swarm size in the 4-patterns data reconstruction

| | aGBPSO | aPBPSO | SAD PSO | SPPSO | SPSA-PSO | SPSA | PSO |
|---|---|---|---|---|---|---|---|
| *num_birds* | 20 | 20 | 20 | 20 | 200 | | 200 |
| *num_epochs* | 100 | 100 | 100 | 100000 | 100000 | | 100000 |
| *opt_iter* | 1000 | 1000 | 1000 | | 1000 | 100000 | |

Table 5.3: Swarm size in the 10-patterns data reconstruction

| | aGBPSO | aPBPSO | SAD PSO | SPPSO | SPSA-PSO | SPSA | PSO |
|---|---|---|---|---|---|---|---|
| *num_birds* | 30 | 30 | 30 | 30 | 200 | | 200 |
| *num_epochs* | 100 | 100 | 100 | 100000 | 100000 | | 100000 |
| *opt_iter* | 1000 | 1000 | 1000 | | 1000 | 100000 | |

Table 5.4: Swarm size in the 26-patterns data reconstruction

| | aGBPSO | aPBPSO | SAD PSO | SPPSO | SPSA-PSO | SPSA | PSO |
|---|---|---|---|---|---|---|---|
| *num_birds* | 50 | 50 | 50 | 50 | 200 | | 200 |
| *num_epochs* | 100 | 100 | 100 | 100000 | 100000 | | 100000 |
| *opt_iter* | 1000 | 1000 | 1000 | | 1000 | 100000 | |

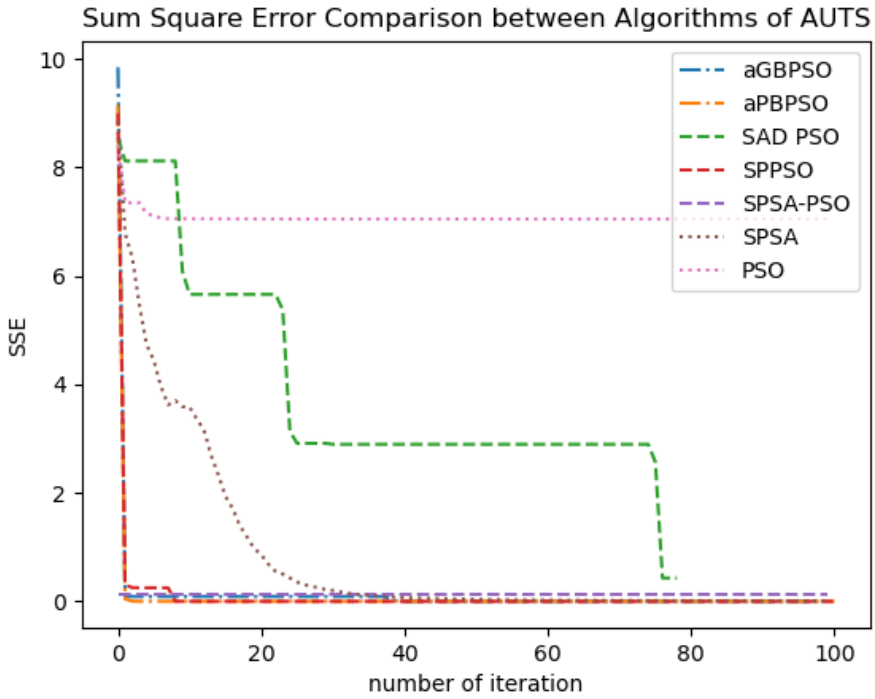
## 5.2 Sum of squared error analysis

The sum of squared error values varies drastically among all seven algorithms in solving data reconstruction problems. Unlike the XOR experiment and the data classification experiment, the proposed aGBPSP and aPBPSO confidently show off their ability to search for the global best optimal solution, which is far better than others, especially in 10-patterns and 26-patterns cases.

The line charts figure 5.1, figure 5.2, and figure 5.3 seem a bit complex to interpret, but taking a careful observation of the proposed two approaches, informative viewpoints are shown. They always drop the SSE to the lowest point, approximately close to zero, but some others (e.g., the PSO, the SPPSO, and the SPSA-PSO) even keep obtaining high error values until the end of the run time. Whether considering the converging speed or the final SSE at each iteration, they are more powerful than the three previous proposed algorithms, the SAD PSO, the SPPSO, and the SPSA-PSO. Again, the standard SPSA can also achieve optimal results, which means that the fixed parameters (i.e., $a, c, \alpha, \gamma, A$)

chosen for all the algorithms are appropriate since they are sharing the same setting.

In the 4-pattern case figure 5.1, the aGBPSO and the aPBPSO perform equivalently, but in the 10-patterns case figure 5.2 and the 26-patterns case figure 5.3, the result of the aGBPSO can be closer than the aPBPSO to the global best optima. The phenomenon is the same as the experiments implementing the MLP (i.e., Because the XOR's number of data input is smaller in comparison with the iris dataset, and both datasets are tested by the machines based on MLP). The aGBPSO algorithm is more suited to more complicated problems. [11] can explain why the aPBPSO is not efficient enough, since plugging gradient information to every particle at every iteration may disturb the fitness of all the particles in the swarm and further impact the directive movement of the entire swarm.



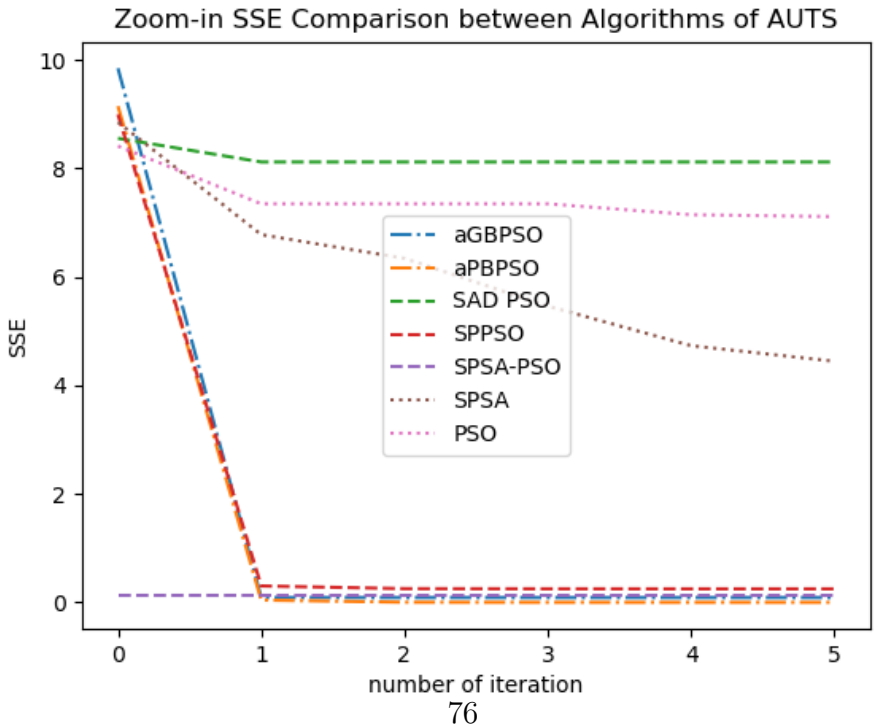
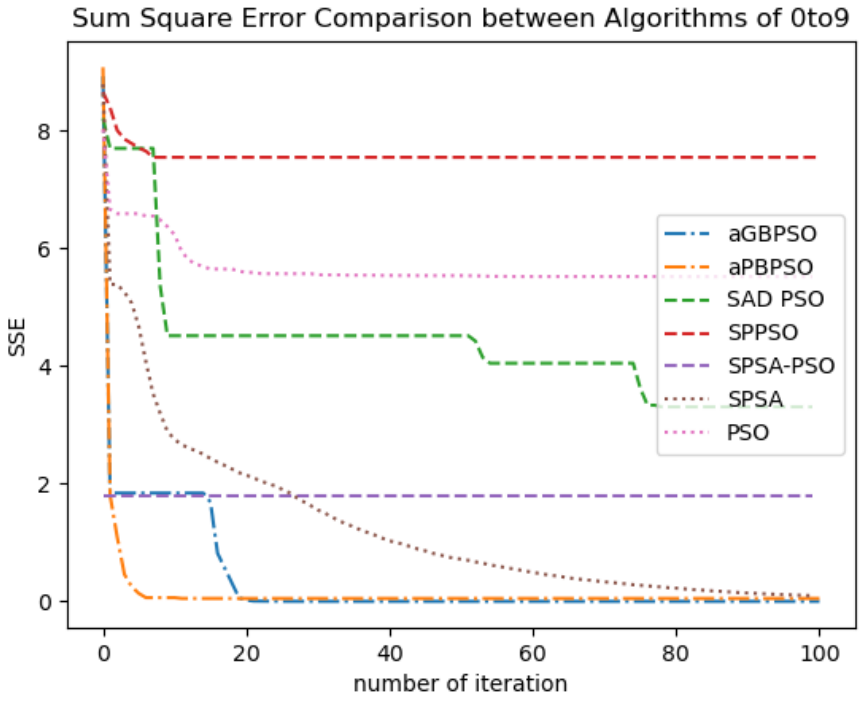






Figure 5.1: SSE of three algorithms analyzed at each iteration in the 4-patterns data reconstruction



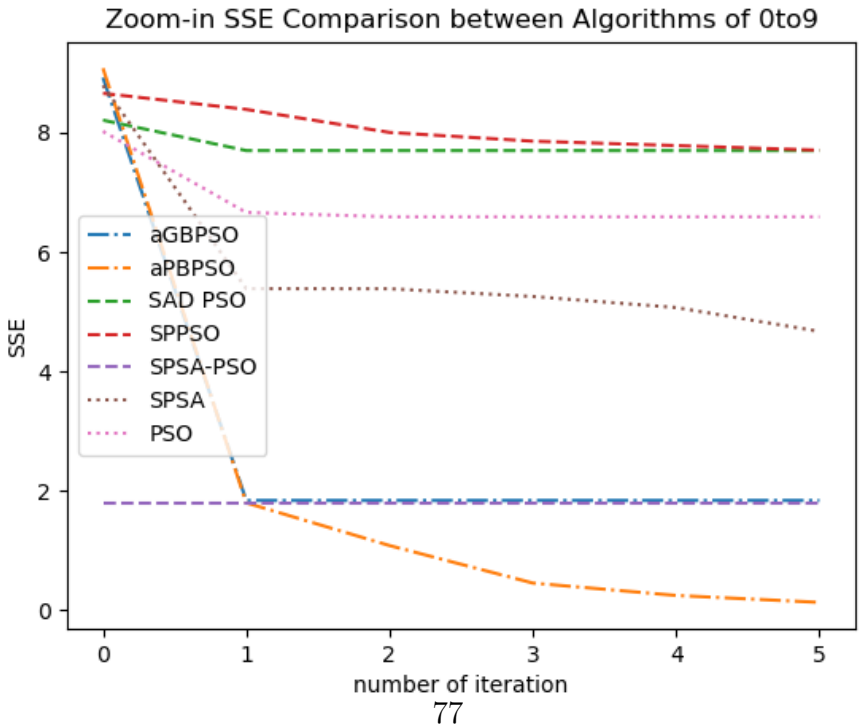
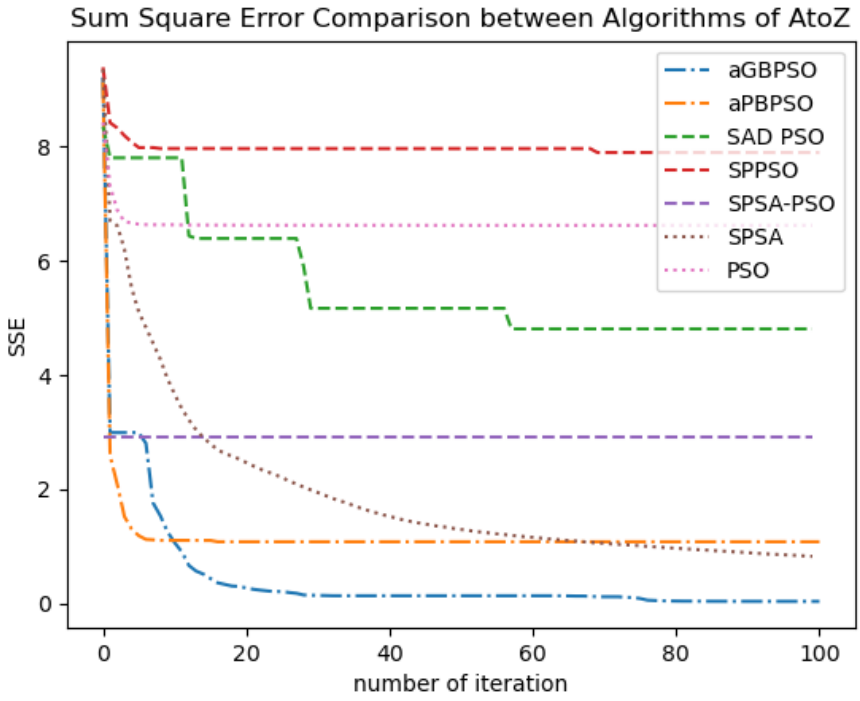






Figure 5.2: SSE of three algorithms analyzed at each iteration in the 10-patterns data reconstruction



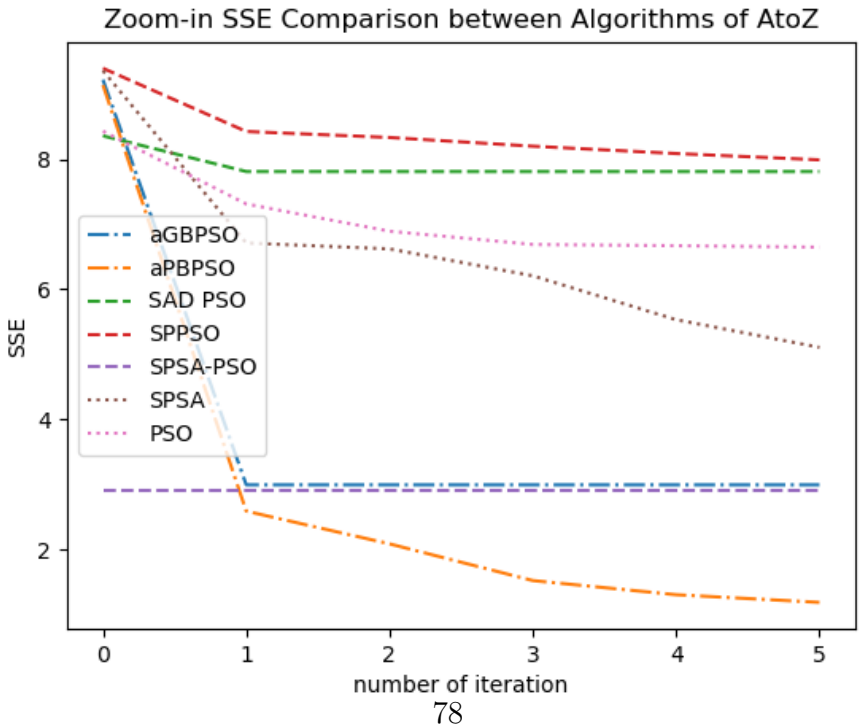






Figure 5.3: SSE of three algorithms analyzed at each iteration in the 26-patterns data reconstruction

### 5.2.1 Intermediate error analysis of aGBPSO & aPBPSO

When analyzing the intermediate SSE of data reconstruction cases shown in figure 5.4, the last paragraph's statement is confirmed deeper. There is a fluctuation at the beginning of the blue line, but with the dataset gradually grows from a smaller number to a large number, the fluctuation become smaller. Furthermore, the intermediate SSE presented by the blue line is much lower than that represented by the orange line in the more complicated cases. That information emphasizes that the aGBPSO is able to respond to difficult optimization problems.

The blue line fluctuation is also an explanation of the drawback of the standard SPSA (i.e., no guarantee to the global convergence). It keeps searching for a better solution around the current best global position. However, it could move potentially further from the best position. After several times of searching the space, the direction pointing to the global best optimum is found.

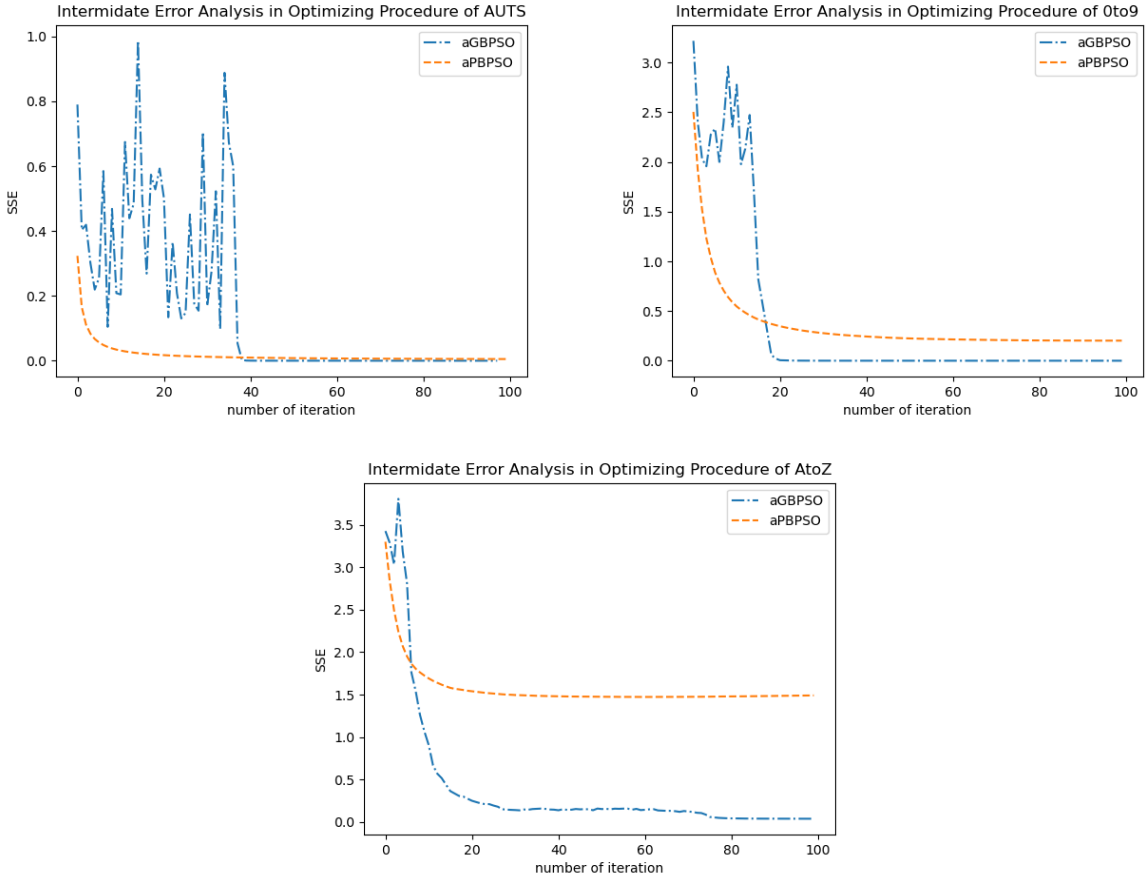



Figure 5.4: Intermediate SSE of three algorithms analyzed at each iteration in data reconstruction

## 5.3 Reconstructed result analysis

The PSO, the SPPSO, and the SPSA-PSO find a solution for the RHN. Hence it does not make sense to continue discussing them. The SAD PSO only works in the simplest 4-pattern case (figure 5.5). The proposed aGBPSO and aPBPSO and the standard SPSA

can convert the distorted data to the original data partially.

For the 26-pattern alphabet case, as shown in figure 5.7, the aGBPSO algorithm successfully reconstructed 19 patterns. Even though 7 patterns failed to the pattern reconstruction partially,, they still can be recognized as the original patterns with minor faults.

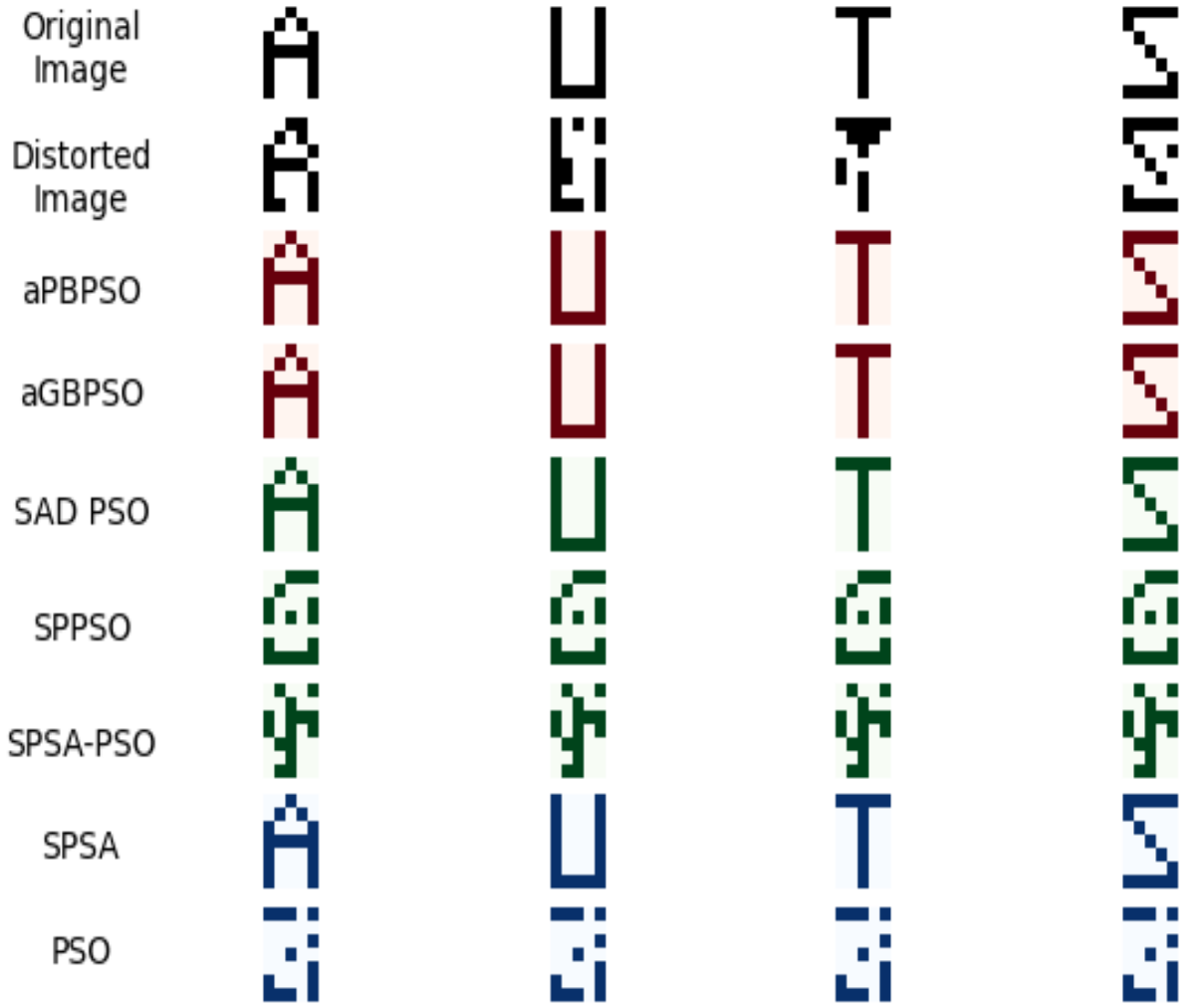



Figure 5.5: Comparison between the original patterns and the results produced by algorithms in the 4-patterns data reconstruction

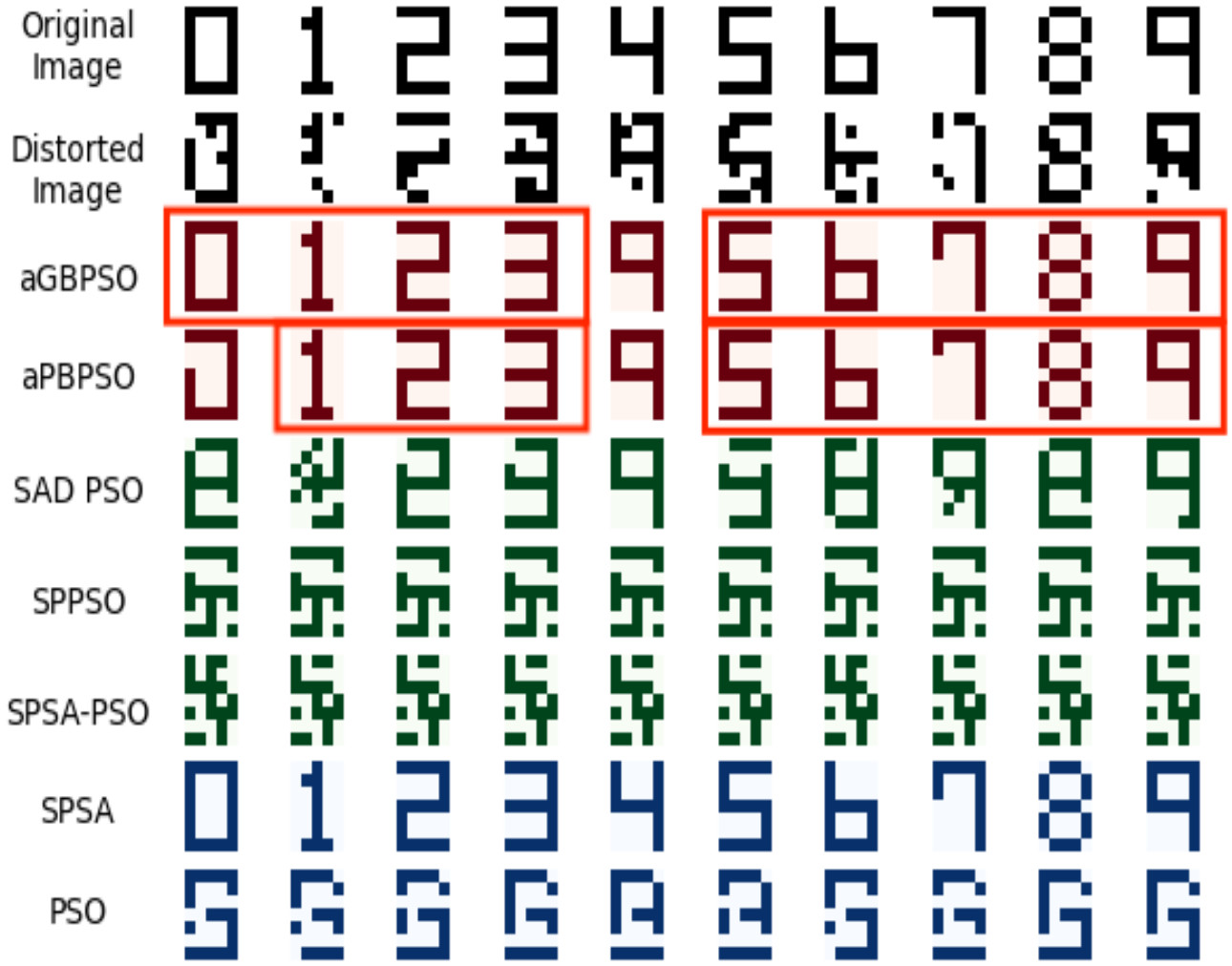



Figure 5.6: Comparison between the original patterns and the results produced by algorithms in the 10-patterns data reconstruction

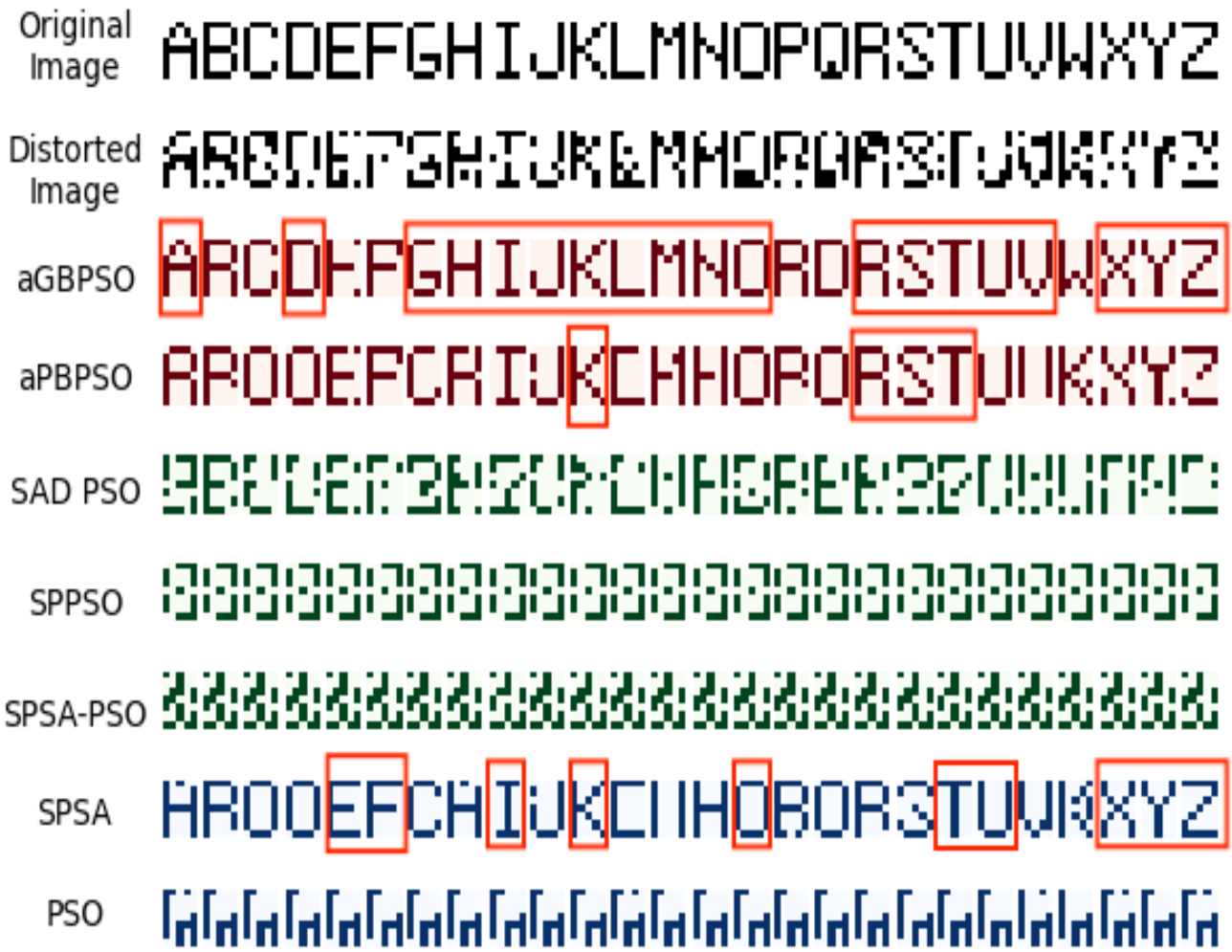



Figure 5.7: Comparison between the original patterns and the results produced by algorithms in the 26-patterns data reconstruction

## 5.4 Statistical Analysis for 4-patterns case

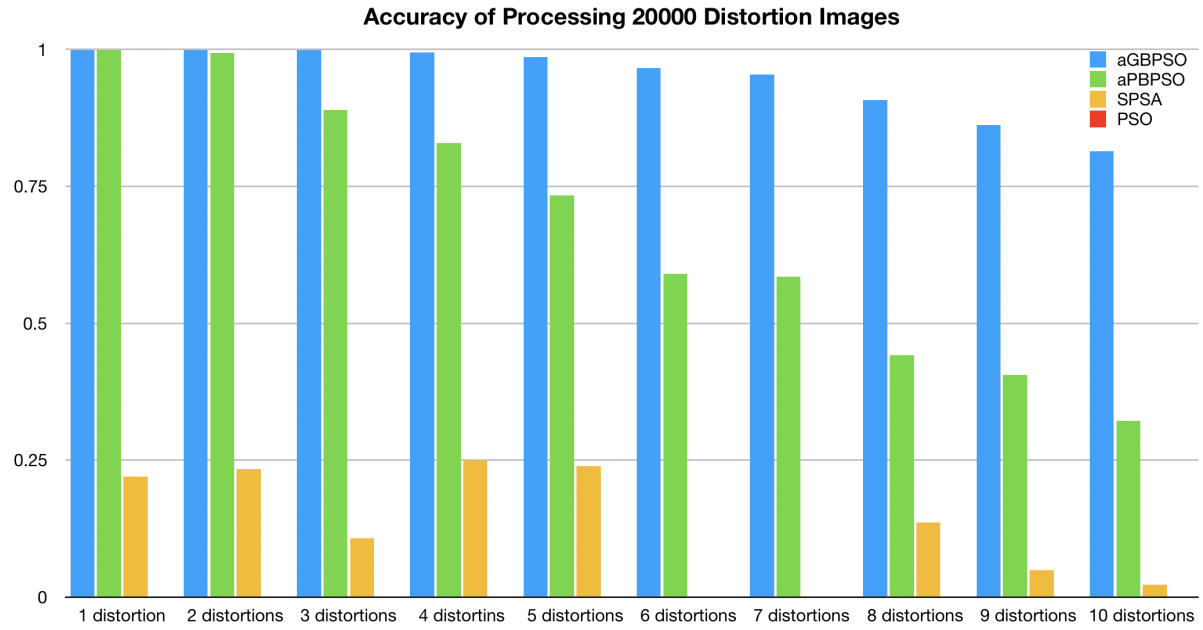
To solid the performance of improvement to the basic algorithms, a statistical test is implemented.

In the experiment, the patterns in the memory are still the "AUTS" binary images. However, another setting is different from the previous 4-patterns test. The number of particles and the number of movements in the proposed algorithms is shrunk to 10 and 10, respectively. Due to the reduction of the total optimization iteration, the standard SPSA is also decreased to 15000 (i.e., the total number of optimization of the proposed algorithms and the standard SPSA remains the same.) Since the PSO performs worse, the parameter is not reduced.

| | aGBPSO | aPBPSO | SPSA | PSO |
|---|---|---|---|---|
| Number of particles | 10 | 10 | | 100 |
| Number of motions | 10 | 10 | | 10000 |
| Local optimized iteration | 1000 | 1000 | 15000 | |

Figure 5.8: Parameters using in statistical analysis

From the Fig.5.9, the aGBPSO is proved that it is the most optimal algorithm. The standard SPSA performs unstably with 15000 iterations. Therefore, with the same amount of optimization iteration, the standard SPSA is weaker, proving that the aGBPSO improves the standard algorithms.




| | 1 distortion | 2 distortions | 3 distortions | 4 distortins | 5 distortions | 6 distortions | 7 distortions | 8 distortions | 9 distortions | 10 distortions |
|---|---|---|---|---|---|---|---|---|---|---|
| **aGBPSO** | 1 | 1 | 1 | 0.99425 | 0.9863 | 0.9657 | 0.9544 | 0.90735 | 0.8616 | 0.8138 |
| **aPBPSO** | 1 | 0.99365 | 0.88975 | 0.82955 | 0.7333 | 0.59 | 0.5846 | 0.44155 | 0.4058 | 0.32135 |
| **SPSA (15000)** | 0.2198 | 0.23375 | 0.10735 | 0.25 | 0.23935 | 0 | 0 | 0.13595 | 0.04925 | 0.0231 |
| **PSO** | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 5.9: Accuracy of Processing 20000 Distorted Images

# Chapter 6

# Conclusions and Future Works

## 6.1 Conclusions

In this work, we focus on solving the main drawback of the standard SPSA and the basic PSO: the weakness of exploring the global best optima. It can be a severe problem, resulting from the premature convergence to multiple local minima (or maxima) resulting from the nonlinear optimization problem's nature. Therefore, we basically seek methods to skip over those local optimum positions in the searching space.

The basic PSO also suffers from the poor equation of updating the global best particle mentioned in the previous research. Adding stochastic approximation (SA) of the gradient estimate about each particle's fitness, the position update procedure can purposefully guide the best particle towards a more optimal direction while avoiding local traps taking advantage of SA's stochasticity. The proposed algorithms are not two variants of the basic PSO or the standard SPSA, instead a "directional" algorithm, which combines the merit of the computational efficiency of the SPSA with the PSO.

We have proposed two approaches where the standard SPSA is applied to the variant of PSO. This variant is with linear reduced inertia weight and velocity clamping. The first method provides an artificial global best particle with the standard SPSA. In contrast, the second method optimizes each particles' position in the swarm (i.e., generates potential artificial personal best position candidates). In both proposed algorithms, only when the result calculated after using the standard SPSA for optimizing proves its superiority, the current position of the particle(s) will be replaced. Both the aGBPSO and the aPBPSO are tested over five situations. The experimental results and the statistical analysis demonstrated that they achieved a relatively optimal performance over all the optimization problems regardless of the problem's dimension without any exception. The experiments have been done by the proposed two algorithms and the other five algorithms, including the existing three algorithms developed by other researchers, the standard SPSA and the PSO. The standard SPSA also shows the ability to complete the optimization goal in all the cases. The constant parameters used in the standard SPSA are shared with the same part in the designed two algorithms. Hence, the excellent performance of the standard SPSA verifies those parameters, such as $A, a, c, \gamma, \alpha$, are appropriately set. Additionally, the aGBPSO and aPBPSO outperform the other algorithms, especially in complex testing cases.

## 6.2 Future Works

Encouraged by the outcomes, the current plan for future works includes the application of the aGBPSO and the aPBPSO to other problems in other domains. For instance, with the computationally-friendly property, they can be applied to some hardware owned a limited calculation power. The parameters tuning of the proposed algorithms is definitely an

important part of the applications. Depending on the situation, different sets of parameters could yield different results.

Besides, a better structure of the proposed algorithms can be designed further, e.g., shrinking the searching range at each optimization iteration. Theoretically, after some epochs, the global best particle could have a relatively clear direction to the best position in the searching space, so it is no need to searching in the same manner as the beginning of the algorithm and the searching range could be reduced.

# References


[1] Genetic algorithms: A powerful tool for large-scale nonlinear optimization problems. *Computers and Geosciences*, 20(7):1229–1236, 1994.

[2] Atyabi A Ab Wahab MN, Nefti-Meziani S. A comprehensive review of swarm optimization algorithms. *PLoS ONE*, 10(5), 05 2015.

[3] W.F. Abd-El-Wahed, Abd allah Mousa, and M. El-Shorbagy. Integrating particle swarm optimization with genetic algorithms for solving nonlinear optimization problems. *J. Computational Applied Mathematics*, 235:1446–1453, 01 2011.

[4] Ahmad Abdulsadda and Kamran Iqbal. An improved spsa algorithm for system identification using fuzzy rules for training neural networks. *International Journal of Automation and Computing*, 8:333–339, 08 2011.

[5] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[6] Wikipedia contributors. Multilayer perceptron.

[7] Andries P. Engelbrecht. *Computational Intelligence: An Introduction, Second Edition*. 2007 John Wiley and Sons, Ltd, 2007.

[8] S.B. Gelfand and S.K. Mitter. Recursive stochastic algorithms for global optimization in ir/sup d/. pages 220–221 vol.1, 1990.

[9] D. Hutchison. On an efficient distribution of perturbations for simulation optimization using simultaneous perturbation stochastic approximation. 2002.

[10] Voratas Kachitvichyanukul. Comparison of three evolutionary algorithms: Ga, pso, and de. *Industrial Engineering and Management Systems*, 12:215–223, 09 2012.

[11] Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Stochastic approximation driven particle swarm optimization with simultaneous perturbation - who will guide the guide? *Appl. Soft Comput.*, 11:2334–2347, 03 2011.

[12] Serkan Kiranyaz, Turker Ince, Alper Yildirim, and Moncef Gabbouj. Fractional particle swarm optimization in multidimensional search space. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 40:298–319, 09 2009.

[13] Donald Knuth. Iris flower data set.

[14] H. J. Kushner. Asymptotic global behavior for stochastic approximation and diffusions with slowly decreasing noise effects: Global minimization via monte carlo. *SIAM Journal on Applied Mathematics*, 47(1):169–185, 1987.

[15] Y. Maeda and T. Kuratani. Simultaneous perturbation particle swarm optimization. pages 672–676, 2006.

[16] Yutaka Maeda and Masatoshi Wakamura. Bidirectional associative memory with learning capability using simultaneous perturbation. *Neurocomputing*, 69:182–197, 12 2005.

[17] J. Maryak and Daniel Chin. Global random optimization by simultaneous perturbation stochastic approximation. 2:910–916 vol. 2, 01 2001.

[18] J.L. Maryak and D.C. Chin. Global random optimization by simultaneous perturbation stochastic approximation. In *Proceeding of the 2001 Winter Simulation Conference (Cat. No.01CH37304)*, volume 1, pages 307–312 vol.1, 2001.

[19] Samira Moumen, Ellaia Rachid, and Rajae Aboulaich. A new hybrid method for solving global optimization problem. *Applied Mathematics and Computation*, 218:3265–3276, 12 2011.

[20] Dobrivoje Popovic, Andrew Teel, and Mrdjan Jankovic. An improved spsa algorithm for stochastic optimization with bound constraints. 16, 01 2005.

[21] P. Sadegh and J.C. Spall. Optimal random perturbations for stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 43(10):1480–1484, 1998.

[22] Amaresh Sahu, Sushanta Panigrahi, and Sabyasachi Pattnaik. Fast convergence particle swarm optimization for functions optimization. *Procedia Technology*, 4:319–324, 12 2012.

[23] S. Seyedpoor, Javad Salajegheh, Eysa Salajegheh, and S. Gholizadeh. Optimal design of arch dams subjected to earthquake loading by a combination of simultaneous perturbation stochastic approximation and particle swarm algorithms. *Applied Soft Computing*, 11:39–48, 01 2011.

[24] James Spall. An overview of the simultaneous perturbation method for efficient optimization. 02 2001.

[25] M.A. Styblinski and T.-S. Tang. Experiments in nonconvex optimization: Stochastic approximation with function smoothing and simulated annealing. *Neural Networks*, 3(4):467–483, 1990.

[26] Lijun Sun, Xiaodong Song, and Tianfei Chen. An improved convergence particle swarm optimization algorithm with random sampling of control parameters. *Journal of Control Science and Engineering*, 2019:1–11, 06 2019.

[27] Bala Deshpande Vijay Kotu. *Data Science (Second Edition)*. 2019.

[28] I.-J Wang and J.C. Spall. Stochastic optimization with inequality constraints using simultaneous perturbations and penalty functions. volume 4, pages 3808 – 3813 vol.4, 01 2004.

[29] Peter Wittek. *The Principles of Quantum Mechanics*. Academic Press, 2014.

[30] Hongtao Ye, Wenguang Luo, and Zhenqiang Li. Convergence analysis of particle swarm optimizer and its improved algorithm based on velocity differential evolution. *Computational intelligence and neuroscience*, 2013:384125, 08 2013.

[31] T. Yeap. Implementation of an associative memory using a restricted hopfield network. *Proc. Intl. Research Conference*, pages 112–116, 2020.

[32] Xiumei Yue. Improved simultaneous perturbation stochastic approximation and its application in reinforcement learning. pages 329 – 332, 01 2009.