Chapter 4

Global Optimization Algorithms

Gradient descent (GD) is the most popular local optimization algorithm to train multilayer NNs. While GD has shown to be successful in training SUNNs, GD fails to train PUNNs under general assumptions of weight initialization, as shown in the previous chapter. This chapter presents an overview of the following global optimization algorithms: Particle Swarm Optimization (PSO), Genetic Algorithms (GAs) and Leapfrog Optimization (LFOP). These algorithms are subsequently applied to approximate a set of functions, using PUNNs. The results are compared with that of SUNNs, using gradient descent optimization.

4.1 Particle Swarm Optimization

Particle swarm optimization (PSO) is a global optimization approach, modeled after the social behaviour of flocks of birds [Eberhart *et al* 1996, Heppner *et al* 1990, Reynolds 1987] and schools of fish [Wilson 1975] . Heppner was interested in discovering the underlying rules that enabled large numbers of birds to flock synchronously, often changing direction suddenly, scattering and regrouping [Kennedy *et al* 1995b].

These scientists had the insight that local processes, such as those modelled by cellular automata, might underlie the unpredictable group dynamics of bird social behaviour [Kennedy *et al* 1995b]. The models proposed by these scientists relied heavily on manipulation of inter-individual distances; that is, the synchrony of flocking behaviour was thought to be a function of birds' efforts to maintain an optimum distance between themselves and their neighbours.

Particle swarm optimization was originally developed by Eberhart and Kennedy [Eberhart *et al* 1995, Eberhart *et al* 1996, Kennedy 1995a, Kennedy *et al* 1995b]. PSO is a population based search procedure where the individuals, referred to as particles, are grouped into a swarm. Each particle in the swarm represents a possible solution to the optimization problem under consideration. In a PSO system, each particle is 'flown' through the multidimensional search space, adjusting its position in search space according to own experience and that of neighbouring particles. Each particle is treated as a point in a D-dimensional space. The p^{th} particle is represented as $\vec{x}_p = (x_{p,1}, x_{p,2}, ..., x_{p,D})$. The best previous position (i.e. the position that produces the best fitness value) of the p^{th} particle is recorded and represented as $\overrightarrow{BESTx_p} = (BESTx_{p,1}, BESTx_{p,2}, ..., BESTx_{p,D})$, and the index of the best particle among all the particles in the population is represented by, GBEST. Let the rate of change in position (i.e. velocity) for particle p be represented as $\vec{v}_p = (v_{p,1}, v_{p,2}, ..., v_{p,D})$. The p^{th} particle is adjusted according to the following equation,

$$\begin{aligned}\vec{v}_p(t) = w \times \vec{v}_p(t-1) \ + \ c_1 \times rand1() \times (\overrightarrow{BESTx_p} - \vec{x}_p(t)) \ + \\ c_2 \times rand2() \times (\overrightarrow{BESTx_{GBEST}} - \vec{x}_p(t)) \qquad (4.1)\\ \vec{x}_p(t+1) \ = \ \vec{x}_p(t) + \vec{v}_p(t) \qquad (4.2)\end{aligned}$$

where c_1 and c_2 are positive constants, referred to as the acceleration constants, and $c_1 + c_2 < 4$ to ensure convergence [Van den Bergh 2001a], $rand1()$ and $rand2()$ are two random functions with output in the range $[0, 1]$, w is the inertia weight and $\vec{v}_p$ the velocity of particle p before the adjustment [Shi *et al* 1998].

Equation (4.1) is used to calculate the particle's new velocity using its previous velocity and the distances of its current position from its own best experience (position) and the group's best experience, which is defined in terms of the type of social interaction that is being modeled [Shi *et al* 1998]. Two approaches of PSO have been developed by Eberhart and Kennedy, one globally oriented, referred to as GBEST, and one locally oriented referred to as LBEST [Eberhart *et al* 1995]. In both approaches, each particle of the swarm keeps track of its coordinates in search space which are associated with the best solution the particle has achieved so far. This position is referred to as $\overrightarrow{BESTx}$. In the local version of PSO, each particle keeps track of the best solution called 'LBEST', attained within a local topological neighbourhood of particles. In the GBEST model the group's best experience is indicated to by index 'GBEST'. The particle therefore makes use of the best position encountered by itself and the overall best position of either,

- all particles, as indicated to by 'GBEST' (GBEST model) or
- a neighbourhood of particles, as indicated to by 'LBEST' (LBEST model)

to position itself towards the global minimum. The effect is that particles 'fly' towards the global minimum, while still searching a wide area around the best solution.

The performance of each particle (i.e. the 'closeness' of a particle to the global minimum) is measured according to a predefined fitness function which is related to the problem being solved. Research has shown that the GBEST version of PSO performs best in terms of a median number of iterations to converge compared to the LBEST model [Eberhart *et al* 1996].

The PSO algorithm is summarized below to illustrate its simplicity:

4.1.1 PSO Algorithm

1. Initialize a swarm of S D-dimensional particles, with positions and velocities, where D is the number of weights and biases.

2. Evaluate the fitness f_p of each particle p as the MSE over a given data set.

3. If $f_p < BEST_p$ then $BEST_p = f_p$ and $\overrightarrow{BESTx_p} = \vec{x}_p$, where $BEST_p$ is the current best fitness achieved by particle p, $\vec{x}_p$ is the current position of particle p in D-dimensional weight space, and $\overrightarrow{BESTx_p}$ is the position corresponding to particle p's best fitness so far.

4. If $f_p < BEST_{GBEST}$ then $GBEST = p$, where $GBEST$ is the particle having the overall best fitness over all particles in the swarm.

5. Change the velocity $\vec{v}_p$ of each particle p using equation (4.1).

6. Fly each particle p to $\vec{x}_p + \vec{v}_p$

7. Loop to step 2 until convergence

In step 5, the coordinates $\overrightarrow{BESTx_p}$ and $\overrightarrow{BESTx}_{GBEST}$ are used to pull the particles towards the global minimum, and the acceleration constants, c_1 and c_2, control how far particles fly from one another.

Initially, all particles are assigned random positions, selected from a range that covers the entire search space, and random velocities that do not exceed a maximum velocity as specified for the problem.

The next section discusses the parameters; inertia weight, maximum velocity and acceleration constant of PSO.

4.1.2 Inertia Weight

The purpose of the inertia weight is to control the impact of the previous history of velocities on the current velocity. A larger inertia weight favours global exploration, while a smaller inertia weight tends to facilitate local exploration of the search area [Kennedy *et al* 1995b]. Suitable selection of the inertia weight w can provide a balance between local and global exploration abilities of PSO, and thereby reducing the number of iterations required in reaching an optimum. The value for the inertia weight is problem dependent, but usually values between 0 and 1.0 are used. PSO with decreasing inertia weight has also been implemented, where the PSO usually starts off with a large inertia weight, say 1.0, and gradually reduces it with time.

4.1.3 Maximum Velocity

The maximum velocity is used to prevent large velocity updates, thereby preventing particles from leaving the search space. Thus, in PSO, the value of the maximum velocity is *limited* to prevent particles from flying out of the search space. Shi and Eberhart pointed out that the maximum velocity acts as a constraint that controls the maximum global exploration ability of PSO [Shi *et al* 1998]. The maximum global exploration ability of PSO is limited if the maximum velocity is too small. If the maximum velocity is too small, then particles may not explore sufficiently beyond good regions. Further, they may become trapped in local minima, unable to jump far enough to reach a better position in the search space [Eberhart *et al* 1996]. A larger maximum velocity, increases PSO's maximum global exploration ability. A too high maximum velocity, however, will result in particles flying past good solutions. The maximum velocity determines also the fineness with which regions between the present position and the target position will be searched. The value of the maximum velocity should thus be selected carefully. The maximum velocity is limited by the maximum value of the parameters for the problem at hand, i.e. the maximum value of the inputs. Usually values in the range 0 to 5 are chosen for the maximum velocity.

4.1.4 Acceleration Constants

The acceleration constants, c_1 and c_2, represent the weighting of the stochastic acceleration terms that pull each particle toward positions $\overrightarrow{BESTx}_p$ and $\overrightarrow{BESTx}_{GBEST}$. Thus, adjustment of this factor changes the amount of 'tension' in the system. Low values allow particles from far target regions to explore the search space before being tugged back, while high values result in abrupt movement toward the target regions [Eberhart *et al* 1995]. The selection of values for the acceleration constant is problem dependent, however values normally range between 0 and 5. Also, if the constraint, $c_1 + c_2 < 4$, is not satisfied then PSO does not usually converge [Eberhart *et al* 2000, Van den Bergh 2001a].

4.1.5 Applications of PSO

Particle swarm has been used successfully to train SUNNs [Kennedy *et al* 1995b, Van den Bergh 1999, Van den Bergh *et al* 2000] and PUNNs [Engelbrecht *et al* 1999a, Ismail *et al* 1999, Ismail *et al* 2000, Van den Bergh *et al* 2001b], for function optimization [Eberhart *et al* 1995, Shi *et al* 1999, Van den Bergh *et al* 2001d] and for human tremor analysis [Eberhart *et al* 1999]. Van den Bergh found that training of multilayer feed-forward networks using various gradient descent based algorithms can be improved significantly by using particle swarm optimization in selecting initial weights. Van den Bergh showed that the initial weights produced by PSO increased the speed and accuracy with which gradient descent algorithms find the minimum [Van den Bergh *et al* 2000]. Another researcher, Salerno, also applied particle swarm optimization successfully to train a recurrent neural network in parsing natural language phrases [Salerno 1997].

PSO has also been demonstrated to perform well in optimizing genetic algorithm test functions, such as the extremely nonlinear Schaffer $f6$ function. The $f6$ function is very difficult to optimize, as the highly discontinuous data surface features many local minima. PSO found the global optimum each run and appears to approximate the results reported by Davis for basic genetic algorithms in terms of the number of evaluations to reach certain performance levels [Kennedy *et al* 1995b]. PSO can be used to solve many of the same kind of problems as solved by genetic algorithms [Kennedy *et al* 1995b]. Eberhart *et al* used PSO successfully to extract rules from fuzzy neural networks [Eberhart *et al* 1998]. BK Birge, a former student of Eberhart, one of the developers of PSO, and Y Shi is currently applying PSO to 'intelligent control' for NASA's next generation 'Robotic Mars Landers'. Current research in PSO use constriction coefficients which have lead to improved performance of PSO [Eberhart *et al* 2000].

4.1.6 Advantages of PSO

The PSO offers several advantages, which makes it an excellent choice to solve optimization problems with a continuous search space. These advantages include:

- PSO is conceptually simple and can be implemented in a few lines of code, requiring only basic mathematical operations.
- In PSO, neural network weights and structures are evolved in such a way as to make preprocessing of neural network data unnecessary.
- PSO is computationally inexpensive in terms of both memory requirements and speed [Kennedy *et al* 1995b].
- PSO is a stochastic global optimization algorithm.

Another advantage is that PSO does not suffer from some of the difficulties encountered with genetic algorithms (e.g. running the risk of finding suboptimal solutions); interaction in the group enhances rather than detracts from progress toward a solution [Eberhart *et al* 1996]. PSO also has memory, which a genetic algorithm generally does not have. Change in genetic populations results in destruction of previous knowledge of the problem except when elitism (i.e. when individuals with highest fitness of the current generation is copied into the next generation) is employed, in which case a small number of individuals retain their identities. This serves as limited memory.

PSO is a global optimization algorithm and training of a NN is an optimization problem. Hence, PSO can be used for training a NN, in which case each particle will represent a weight of the NN (including biases). The dimension of the search space is therefore the total number of weights and biases. The fitness function is the mean squared error (MSE) over the training set, or the test set (as measure of generalization). This thesis implements the GBEST version of PSO.

This concludes the presentation of PSO. The next section presents genetic algorithms (GAs) as an optimization algorithm.

4.2 Genetic Algorithms

Evolutionary computing has been used successfully to solve optimization problems. Of these, genetic algorithms (GAs) are the most popular. GAs are based on the principle of natural evolution where principles such as survival of the fittest, natural selection, reproduction and mutation are used to produce a 'best' individual. The idea of a genetic algorithm as a global optimization tool was first introduced by John Holland in the 1970's [Goldberg 1989]. A genetic algorithm is a *global search* technique compared to gradient descent that is a *local* search method. A GA represents an intelligent exploitation of a random search used to solve optimization problems.

Genetic algorithm paradigms work on populations of individuals, rather than on single data points or vectors. In a GA, a population of individuals compete to survive. Each individual represents a point in search space, which represents one possible solution to the optimization problem. In this thesis, an individual represents a weight vector (including biases and distortion units) of a NN. Each individual is represented as a character string that is analogous to the chromosome that occurs in DNA. The survival strength, or fitness, of an individual is measured using a fitness function, the MSE when a GA is used to train a NN. The fitness value represents the abilities of an individual to survive.

Most optimization paradigms move around in the search space using some heuristic. One of the drawbacks of this approach is the likelihood of getting stuck at a local minimum. GAs on the other hand start off with a diverse set of points called a population. From one population to the next the same number of individuals is maintained, thus allowing many maxima to be explored efficiently and thereby lowering the probability of getting stuck in a local minimum. GAs use 'selective breeding' of the solutions to produce 'offspring' that exhibit better fitness than the parents by combining 'genes' of the parents. GAs do not require any auxiliary information, such as derivatives in determining the maximum (or minimum). GAs are generally more robust than conventional artificial intelligence systems, in that they will still produce reasonable results in the presence of noise or if the inputs change. A GA may offer significant benefits over more typical search optimization techniques, such as linear programming, heuristics, depth-first and breadth-first [Mitchel 1996].

Optimization in a GA proceeds through the generation of new individuals by probabilistically applying crossover and mutation operators. Parents are selected for reproduction based on their fitness. Individuals with high fitness are given more opportunities to reproduce, than individuals with low fitness. Thus the larger the fitness of an individual, the more likely it is that it will be used during crossover to exchange 'genetic material' with another individual to produce better individuals. Thus 'genes' from good individuals produce offspring that are often 'better' than the parents. Mutation occurs by randomly changing a 'gene' of an individual. New offspring replace other individuals with lower fitness. It is hoped that after successive generations better solutions will replace weaker ones.

4.2.1 Applications of GAs

GAs have been used successfully for many applications, which include the training of NNs. Schiffman *et al* used GAs to train SUNNs [Schiffman *et al* 1992], while Frenzel has applied GAs to train PUNNs [Janson *et al* 1993]. In a study conducted by Dagli and James to search for optimal parameters (such as the number of nodes in each layer and the number of layers) for a neural network, the parameters rather than the weights were encoded in the GA chromosome where the neural network's performance with these parameters was used as the fitness function [Dagli *et al* 1995]. Other applications of GAs include,

- pattern classification [Chang *et al* 1991],
- feature selection for neural networks [Guo *et al* 1992],
- the initialization of Radial Basis Networks [Billings *et al* 1995, Burdsall *et al* 1997],
- the training of cellular neural networks [Zamparelli 1997],
- to explain the behaviour of neural networks by defining a function linking the network inputs and outputs [Opitz *et al* 1994], and
- to configure radial basis function (RBF) neural networks [Billings *et al* 1995, Kuo *et al* 1994, Whitehead *et al* 1996]. Specifically, they have been applied to find the optimal (Gaussian) parameters used (centres, widths), as well as the structure (number of hidden nodes) of the RBF network.

A general genetic algorithm for training NNs is presented below.

4.2.2 Genetic Algorithm

1. Initialize a population, $G(t)$, of individuals (weight vectors).
2. Calculate the fitness of each individual of the population as the MSE over the training set.
3. Select parents for reproduction from the current population, $G(t)$. Two individuals are selected from the population using *ranking* as the selection operator (refer to section 4.2.8).
4. Perform crossover to produce new individuals for population, $G(t+1)$. A two-point crossover operator is used (refer to section 4.2.8).

5. Perform mutation of population, $G(t+1)$.

6. Loop to step 2 until best individual is acceptable.

4.2.3 Initialization and Size of Population

Initialization of the population is usually done stochastically. It is sometimes appropriate to start with one or more individuals that are selected heuristically, to aim the GA in a promising direction. Generally the population should represent a wide assortment of individuals. Researchers have shown that the urge to skew the population significantly should generally be avoided. Choosing the size of the population is more an art than science. Following De Jong's guidelines, a moderately sized population should be used initially [De Jong 1975].

4.2.4 Representation

The objective of applying GAs to neural network training is to find a suitable set of weights that results in the smallest MSE on the training set and that generalizes well on the test set. To achieve this objective the GA has to be populated with sets of weights where each set of weights is a possible solution in training the network. Thus, for NN training, each individual of the GA contains the same number of genes as the number of weights (including biases) that occur in the neural network. Each weight value has to be converted to a binary representation, since this thesis assumes that the GA paradigm uses a binary alphabet. The accuracy of the final weight values are determined by the number of bits used in the binary representation and the range of values that is mapped onto this binary representation, e.g. to map real numbers in the range [-3.0,3.0] onto a 30 bit binary representation, implies a mapping onto 2^{30},(i.e. 1073741824), distinct values. Thus -3.0 is mapped onto say 000000000000000000000000000000 and 3.0 mapped onto

111111111111111111111111111111 (30 ones); a finite number of real values between -3.0 and 3.0 are then mapped onto the remaining binary representations that exist between these two binary numbers. In this representation two successive real numbers will therefor differ by magnitude $(3.0-(-3.0))/(2^{30}-1) = 6/1073741823 = 5.5879E-09$. In this representation any two weights that differ by less than this magnitude are thus indistinguishable or regarded as representing the same number. For the implementation in this thesis, each weight is mapped onto a 30 bit binary number. Weight values are restricted to the range $[w_{min}, w_{max}]$, in other word the evolutionary process cannot evolve weights beyond these boundaries. The following mapping function is used to convert floating-point weight values to binary representation:

$$(2^{30}-1)\frac{w-w_{min}}{w_{max}-w_{min}} \tag{4.3}$$

4.2.5 Fitness

One method of fitness calculation is to 'equally space' the fitness values in some manner, say from 0 to 1. The most fit individual has a maximum fitness of 1. Another method of fitness calculation is 'scaling' that takes into account the recent history of the population. If the objective of a GA is to maximize some function, then scaling involves keeping a record of the minimum fitness value obtained in the last s generations, where s is the size of the scaling window. If, for example , $s = 10$, then the minimum fitness value in the last 10 generations is kept and used instead of 0 as the 'floor' of fitness values. Fitness values are then assigned a value based on their actual distance from the floor value. The fitness function used in this thesis is defined as $f(w) = \frac{1}{1+MSE(w)}$. Hence, the smaller the MSE, the larger the fitness value.

4.2.6 Crossover

Crossover is inspired by natural evolution processes. Crossover is a reproduction operator which forms a new individual (chromosome) by combining parts of each of two 'parent' chromosomes with an objective of increasing the fitness value of the new individual. The most 'basic' crossover type is *one-point crossover*, as describe by Holland [Holland 1992] and others, e.g. Goldberg [Goldberg 1989] and Davis [Davis 1991]. One-point crossover involves selecting a single crossover point at random and exchanging the portions of the individual strings of the parents to the right of the crossover point. In *two-point crossover*, on the other hand, two parents are randomly selected from the population and a stochastic decision is made whether or not to perform crossover. Subsequently, if crossover has to be performed, a two-point crossover site along the character string is randomly chosen. The corresponding values occurring between these two points in each parent are then exchanged. An alternative is *uniform crossover*, where two parents are chosen at random and a stochastic decision is made whether or not to perform crossover [Syswerda 1991]. If crossover has to be performed then a random decision is made at each bit position in the string as to whether or not to exchange corresponding bits between the two parent strings. De Jong suggested a high crossover rate of between 0.5 and 0.9 [De Jong 1975]. The values for crossover is, however, problem dependent. In this thesis two-point crossover is used with crossover rates varying between 0.5 and 0.9.

4.2.7 Mutation

Mutation is a way of varying the 'gene pool' that provides some protection against 'in breeding' in a population. Mutation is achieved by stochastically flipping the bits of the individuals during each generation at a certain probability. Mutation is usually performed with a low probability, but higher probabilities are not unusual. A good strategy is to start off with a fairly high probability for mutation that is decreased with time. This allows the GA, initially, greater exploration abilities. In this thesis fixed mutation rates between 0.001 and 0.5 are used.

4.2.8 Reproduction or Selection

Reproduction is a process in which individual strings are selected for mating according to their fitness values. Thus strings with high fitness values have a high probability of contributing to one or more offspring in the next generation [Goldberg 1989]. Operators for implementing reproduction are random selection, biased roulette wheel or tournament selection. In *random selection* individuals for the next generation are randomly selected from the current population. In the *biased roulette wheel* approach each individual is assigned a roulette wheel slot sized in proportion to its fitness. Individuals with high fitness will thus have a bigger size slot than individuals with low fitness values. The roulette wheel is then spun n times to generate a population of size n. The individual that corresponds to the slot that the dice ends up in after each spin is added to the next generation. Thus the bigger the slot size, the greater the probability that the dice will land in it and thus the greater the probability of that individual being added to the next generation. One variation on the roulette wheel was developed by Baker in which the portion of the roulette wheel is assigned, based on each unique string's relative fitness [Baker 1987]. One spin of the roulette wheel then determines the number of times each string will appear in the next generation.

In the most common variation of *tournament selection* two individuals are selected at random and the member with a higher fitness value is selected for the next generation. This process is repeated n times for a population of size n. Other variations include using more than two members selected at a time, and selecting the highest fitness valued member with a certain probability. In the reproduction operators discussed so far all individuals are replaced each generation.

Another approach for selection is referred to as ranking where the individuals are sorted or ranked in ascending fitness values. The pool of individuals to take part in the reproduction is constructed as follows: The top 20% individuals are placed in the mating pool and duplicated. The bottom 20% are culled and do not take part in reproduction. The remainder of the mating pool comprises all the individuals that appear between the top 20% and the bottom 20% on the sorted list of fitness values.

The fitness function defined in section 4.2.5 reveals that the higher the fitness, the lower the corresponding MSE for an individual. In this thesis the 'Simple Genetic Algorithm' (SGA) of Goldberg is implemented [Goldberg 1989]. In the SGA the bottom 20% of each population with respect to fitness is culled. A two-point crossover is used with ranking as the selection operator, where the top 20% individuals with respect to fitness were added twice to the mating pool. The other 60% comprise all individuals between the top 20% and the bottom 20% of the sorted list. For crossover two individuals were randomly selected from the mating pool. Random mutation is used on the offspring. In this thesis, De Jong's guidelines were followed by using a relatively high crossover rate, a relatively low mutation rate and a moderately sized population [De Jong 1975]. On subsequent simulations the crossover rate is decreased, while the mutation and size of population are increased in order to find optimal values for these parameters.

4.2.9 Advantages of GAs

Genetic algorithms have several advantages, for example,

- they require no knowledge or gradient information about the error surface,
- they are generally not trapped by local optima,
- discontinuities on the error surface have little effect on overall optimization performance,
- GAs perform very well for large-scale optimization problems, and
- GAs can be used in a wide variety of optimization problems,

4.2.10 Disadvantages of GAs

While GAs do offer several advantages as mentioned above, they do have drawbacks, such as:

- they generally require *more fitness evaluations* compared to hill-climbing techniques,
- have trouble finding the *exact minimum*. GAs are best at reaching the global region but sometimes have difficulty reaching the exact optimum location. This problem can be overcome by a hybrid approach that uses a genetic algorithm to find the general area of a minimum followed by using gradient descent to find the corresponding minimum.
- finding a *suitable configuration* for a GA, using the various parameters and operators for GAs, is not straightforward.

The next section presents an overview of the Leapfrog Optimization Algorithm (LFOP) developed by Snyman [Snyman 1982b].

4.3 Leapfrog

Leapfrog is a gradient-based optimization approach based on the physical problem of the motion of a particle of unit mass in an n-dimensional conservative force field [Snyman 1982a, Snyman 1982b]. The potential energy of the particle in the force field is represented by the function to be minimized - in the case of NNs, the potential energy is the MSE. The objective is to conserve the total energy of the particle within the force field, where the total energy consists of the particle's potential and kinetic energy. The optimization method simulates the motion of the particle, and by monitoring the kinetic energy, an interfering strategy is adopted to appropriately reduce the potential energy. This algorithm employs an improved time step selection routine in which the time step is automatically reduced or increased to ensure an efficient utilization of the basic dynamic algorithm developed by Snyman [Snyman 1982b]. Snyman recorded results for leapfrog optimization that performed well compared to the conjugate gradient algorithm on the three test functions; Rosenbrock, the homogeneous quadratic and Oren's extended functions [Snyman 1982b]. Other researchers, Holm and Botha have shown that the leapfrog optimization algorithm is a robust algorithm for training summation unit neural networks [Holm *et al* 1999].

The algorithm is summarized below:

4.3.1 Leapfrog Algorithm

1. Compute an initial weight vector $\vec{w}_0$, with random components. Let $\Delta t = 0.5$, $\delta = 1$, $m = 3$, $\delta_1 = 0.001$ and $\epsilon = 10^{-5}$. Initialize $i = 0$, $j = 2$, $s = 0$, $p = 1$ and $k = -1$, where δ denotes the maximum allowable stepsize.

2. Compute the initial acceleration $\vec{a}_0 = -\nabla E(\vec{w}_0)$ and velocity $\vec{v}_0 = \frac{1}{2}\vec{a}_0\,\Delta t$, where $E(\vec{w}_0)$ is the MSE for weight vector $\vec{w}_0$.

3. Set $k = k + 1$ and compute $\|\Delta\vec{w}_k\| = \|\vec{v}_k\|\Delta t$.

4. If $\|\Delta\vec{w}_k\| < \delta$ goto step 5, otherwise set $\vec{v}_k = \frac{\delta\vec{v}_k}{\Delta t\|\vec{v}_k\|}$ and goto step 6.

5. Set $p = p + \delta_1$ and $\Delta t = p\Delta t$.

6. If $s < m$, goto step 7, otherwise set $\Delta t = \frac{\Delta t}{2}$ and $\vec{w}_k = \frac{\vec{w}_k + \vec{w}_{k-1}}{2}$, $\vec{v}_k = \frac{\vec{v}_k + \vec{v}_{k-1}}{4}$, $s = 0$ and goto 7.

7. Set $\vec{w}_{k+1} = \vec{w}_k + \vec{v}_k\Delta t$.

8. Compute $\vec{a}_{k+1} = -\nabla F(\vec{w}_{k+1})$ and $\vec{v}_{k+1} = \vec{v}_k + \vec{a}_{k+1}\Delta t$.

9. If $\vec{a}_{k+1}^T \cdot \vec{a}_k > 0$, then $s = 0$ and goto 10, otherwise $s = s + 1$, $p = 1$ and goto 10.

10. If $\|\vec{a}_{k+1}\| \leq \epsilon$ then stop, otherwise goto 11.

11. If $\|\vec{v}_{k+1}\| > \|\vec{v}_k\|$ then $i = 0$ and goto 3, otherwise $\vec{w}_{k+2} = \frac{\vec{w}_{k+1} + \vec{w}_k}{2}$, $i = i + 1$ and goto 12.

12. Perform a restart: If $i \leq j$, then $\vec{v}_{k+1} = \frac{\vec{v}_{k+1} + \vec{v}_k}{4}$ and $k = k + 1$, goto 8, otherwise $\vec{v}_{k+1} = 0$, $j = 1$, $k = k + 1$ and goto 8.

Whereas PSO and GA perform stochastic parallel searches, leapfrog uses gradient information to guide the search from one search point to the next.

4.4 Experimental Results

This section applies the optimization algorithms discussed in the previous section to the training of product unit neural networks. Results are also compared to that obtained from applying these optimization algorithms to SUNNs. The comparison also includes training PUNNs and SUNNs using gradient descent. The objective of these experiments is to determine if any gain in performance with respect to generalization can be achieved by using PUs. First, the functions used in the experiments are summarized, followed by a description of the experimental procedure.

4.4.1 Test Functions

Eight functions, varying in complexity, were used. These functions are summarized below.

1. The quadratic function $f(z) = z^2$, with $z \sim U(-1, 1)$. The training, test and validation sets consisted of 50 distinct randomly selected patterns.

2. The cubic function $f(z) = z^3 - 0.04z$, with $z \sim U(-1, 1)$. The training, test and validation sets consisted of 50 distinct randomly selected patterns.

3. The henon time series $z_t = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2$, with $z_1, z_2 \sim U(-1, 1)$. The training, test and validation sets consisted of 200 distinct randomly selected patterns.

4. The surface $f(x, y) = y^7x^3 - 0.5x^6$, with $x, y \sim U(-1, 1)$. The training, test and validation sets consisted of 300 distinct randomly selected patterns.

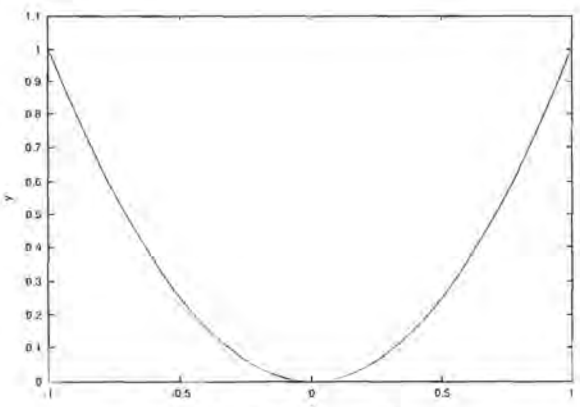
5. The paraboloid $f(x, y) = x^2 + y^2$, with $x, y \sim U(-2, 2)$. The training, test and validation sets consisted of 300 distinct randomly selected patterns.

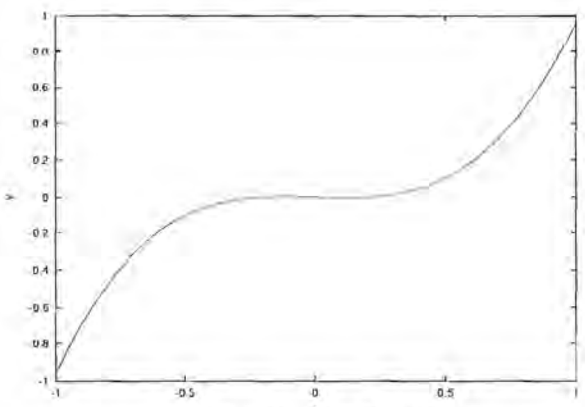
6. The function $f(x, y) = sin(x^2) + sin(y^2)$, with $x, y \sim U(-2, 2)$. The training, test and validation sets consisted of 300 distinct randomly selected patterns.

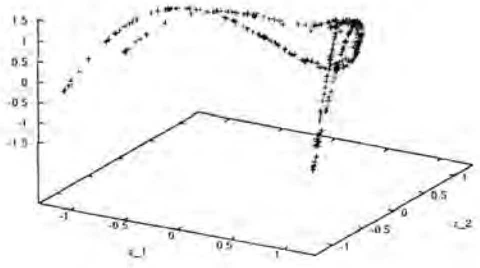
7. The camel function $f(x, y) = 4 - 2.1x^2 + \frac{x^3}{3}x^2 + x \cdot y + (4y^2 - 4)y^2$, with $x \sim U(0, 10)$ and $y \sim U(0, 10)$. The training, test and validation sets consisted of 500 distinct randomly selected patterns.

8. The function $f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$, with $x, y \sim U(0,10)$. This function is also referred to as the 'graph' in this thesis. The training, test and validation sets consisted of 500 distinct randomly selected patterns.

The graphs for the 8 test functions above are displayed in figures 4.1 and 4.2 on pages 100 and 101, respectively.


(a) F1: $f(x) = x^2$


(b) F2: $f(x) = x^3 - 0.04x$


(c) F3: $z_t = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2$

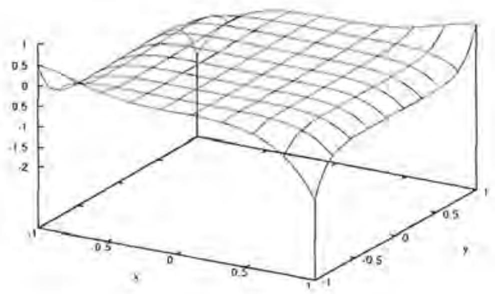
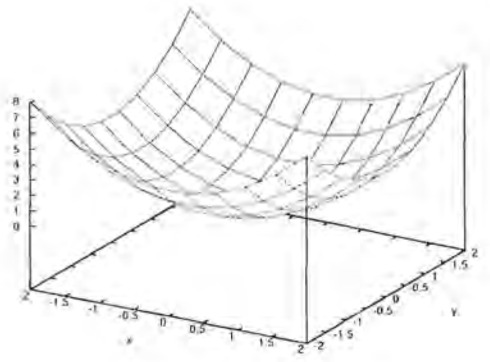
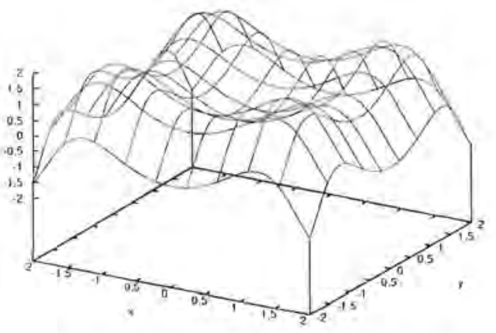
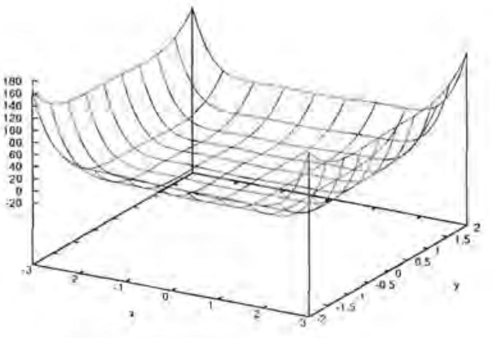

(d) F4: $f(x,y) = y^7x^3 - 0.5x^6$

Figure 4.1: Functions to be approximated

The next section discusses the performance criteria used for optimization of the algorithms.


(e) F5: $f(x,y) = x^2 + y^2$


(f) F6: $f(x,y) = sin(x^2) + sin(y^2)$


(g) F7: $f(x,y) = (4 - 2.1x^2 + (\frac{x^3}{3}))x^2 + xy + (4y^2 - 4)y^2$

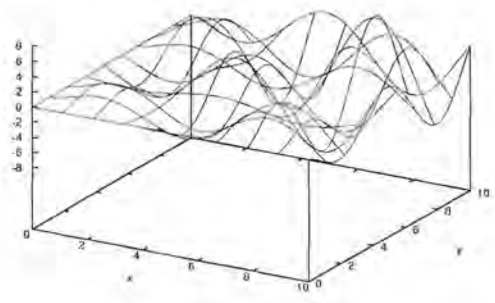

(h) F8: $f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$

Figure 4.2: Functions to be approximated

4.4.2 Performance Criteria

Each optimization algorithm contains a set of parameters that must be fine-tuned to improve convergence. Thus, the optimal parameters for each global optimization algorithm have to be determined before comparing the performance of the various optimization algorithms.

Three performance criteria were considered to determine the optimal PSO, GA, LFOP and BP for each function, namely,

1. the average of 30 simulations of the mean squared error (MSE) on the training and test sets after 500 training epochs for each of the eight functions;
2. the number of epochs required within a maximum of 1000 epochs to reach various generalization levels, i.e. 0.5, 0.1, 0.05, 0.01, 0.001, 0.0001 and 0.00001;
3. the number of simulations that converged for each of the generalization levels mentioned above.

The next section describes the experimental procedure applied to determine the optimal parameters for each optimization algorithm.

4.5 Experimental procedure

In determining the optimal parameters for each of the optimization algorithms, the following procedure was adhered to. All but one parameter were fixed, while the parameter that was not fixed (i.e. the one for which the optimal value has to be determined) assumed values from a range of possible values for that parameter. For each training session one value was selected from the range of values for the parameter under consideration. Training proceeded until all the values from the range were exhausted. A training session consisted of 30 simulations, where each simulation was trained for 500 epochs (for GD and LFOP), iterations (for PSO) or generations (for GA). In each simulation a different training set was used. The average MSE on the test set over the 30 simulations and the number of simulations that converged to a predefined generalization level were recorded. The parameter value that resulted in the lowest average MSE on the training and test sets and that had a high number of simulations that converged to a predefined generalization level was then selected as the best value for the parameter under consideration. This optimal value is then subsequently used in training to determine the optimal parameter values for the remaining parameters. The other remaining parameters are optimized in a similar fashion.

The next sections determines the optimal values of parameters for each of PSO, GA, LFOP and BP for each function.

4.5.1 Parameters for the Optimization Methods

Performance of each of the optimization algorithms used in this thesis is influenced by a number of parameters which should be optimized for each new problem. This section lists for each algorithm the set of parameters for which optimal values were found.

1. **PSO**

   Parameters that influence the performance of PSO include,

   (a) the inertia weight, which controls the balance between the global and local exploration abilities.

   (b) the maximum velocity, which limits the maximum jump that a particle can make in one step.

   (c) the acceleration constants, which control how far particles fly from one another.

   (d) the size of the population (i.e. number of particles in the swarm) affects the run-time of PSO; the larger the swarm, the longer the PSO will take to find a solution.

2. **Back-propagation (BP)**

   A number of factors influence the performance of back-propagation by gradient descent. These include,

   (a) the interval for initial weights, which influences the speed and accuracy with which BP will find the minimum.

(b) the learning rate, which controls the step sizes in the direction of the negative gradient of the error surface.

(c) the momentum, which smoothes out the oscillatory behaviour caused by the stochastic selection of training patterns for on-line learning.

3. **GA**

Convergence of genetic algorithms is influenced by the following three factors,

(a) probability for crossover, which determines how much genetic material will be exchanged between individuals. The higher the crossover rate, the greater the chance of convergence.

(b) probability for mutation, which determines the rate at which bits are mutated or flipped in a bit string. Convergence of a GA, generally requires a small rate for mutation.

(c) the size of the population; the larger the population, the greater the chance of convergence, but the longer it takes to find the solution.

4. **LFOP**

Parameters that influence the convergence of LFOP are,

(a) δ (the maximum allowable stepsize)

(b) δ_1

(c) Δt (the time step) (refer to section 4.3.1 on page 97).

(d) m specifies the number of steps before re-start. A value of 3 for m, worked well in practice.

4.6 Optimizing the Parameters

This section determines the optimal values for parameters for BP and each of the global optimization algorithms. Experimental results are presented to support decisions on which values to use.

4.6.1 Optimal Parameters for PSO

A range of values for the inertia weight, maximum velocity, acceleration constant and number of particles have been tested to find the best combination of parameter values for each experiment (these values are listed in tables 4.1 to 4.3). The acceleration constant in this thesis was implemented using one value to represent both acceleration constants c_1 and c_2 as suggested by Eberhart [Eberhart *et al* 1996]. The PUNNs were trained for each of the functions for a fixed number of epochs (500), where an epoch is one training pass through the training set. Training started with a swarm of 50 particles and parameters maximum velocity and acceleration constant, both initialized to 1.0. In all 8 functions, the weights (or particles) were initialized to random values in the range [-1, 1], also ensuring that approximately 50% of the particles had negative values.

The values for the parameters inertia weight, maximum velocity and acceleration constant appear below in tables 4.1, 4.2 and 4.3, respectively.

Parameter	**Values**								
Inertia weight	0.01	0.25	0.5	0.75	0.875	0.9	0.925	0.95	1.0

Table 4.1: Range of values for inertia weight for PSO

Parameter	**Values**							
Maximum velocity	0.0	0.5	1.0	1.5	2.0	2.5	5.0	10.0

Table 4.2: Range of values for maximum velocity for PSO

Parameter	**Values**							
Acceleration constant	0.0	0.25	0.5	0.75	1.0	1.5	1.75	2.0

Table 4.3: Range of values for acceleration constant for PSO

The optimal values for each of the parameters for the eight functions using PUs and SUs appear in tables 4.4 and 4.5, respectively.

Function	**PSO Parameters (PU)**			
	Inertia weight	Maximum velocity	Acceleration Constant	No of particles
$f(x) = x^2$	1.0	1.0	2.0	30
$f(x) = x^3 - 0.04x$	0.95	1.5	0.75	30
Henon	0.8	5.0	0.75	50
$f(x,y) = y^7x^3 - 0.5x^6$	0.75	10.0	1.5	50
$f(x,y) = x^2 + y^2$	0.9	2.0	1.0	50
$f(x,y) = sin(x^2) + sin(y^2)$	0.75	2.0	1.5	50
Camel	0.75	10.0	1.0	100
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	0.5	1.0	1.5	100

Table 4.4: Best parameters for PSO using PUs

The following section determines the optimal parameters for BP for PUNNs and SUNNs.

Function	**PSO Parameters (SU)**			
	Inertia weight	Maximum velocity	Acceleration Constant	No of particles
$f(x) = x^2$	0.875	5.0	1.0	30
$f(x) = x^3 - 0.04x$	0.875	1.75	0.75	30
$f(x,y) = y^7x^3 - 0.5x^6$	0.75	10.0	1.0	50
Henon	1.0	1.5	0.75	50
$f(x,y) = x^2 + y^2$	0.875	1.5	1.0	50
$f(x,y) = sin(x^2) + sin(y^2)$	0.75	2.5	1.0	50
Camel	0.75	1.0	1.0	100
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	0.75	1.5	1.75	100

Table 4.5: Best parameters for PSO using SUs

4.6.2 Optimal Parameters for BP

The same procedure for determining the optimal parameters in PSO, was also applied to BP, using various initial weight initialization, learning rate and momentum values. The range of values tested for weight initialization appear in table 4.6 on page 108. Various intervals were considered ranging from values close to zero to intervals that contain larger initial values such as [2.0, 4.0], since research by Leerink *et al* suggested larger values for weight initialization [Leerink *et al* 1995]. During weight selection for SUNN and PUNN, it was ensured that approximately 50% of the weights were negative. Tables 4.7 and 4.8 contain the range of values tested for the learning rate and momentum, respectively. The number of simulations of BP applied to PUNNs were increased to 70 to compensate for the high number of simulations that resulted in overflows, due to exponentiation in the learning rule of gradient descent, while the number of simulations for SUNNs remained at 30. In the case of PUNN using gradient descent, optimal parameter values for functions F3, F4, F5, F6, F7 and F8 could not be established, since none of the simulations returned a result, other than overflows.

Parameter	**Intervals**					
Weight interval	$[-1.0, 1.0]$	$[-1.5, 1.5]$	$[-2.0, 2.0]$	$[-3.0, 3.0]$	$[-4.0, 4.0]$	$[-1.0, -0.5]$ & $[0.5, 1.0]$
	$[-1.5, -0.5]$ & $[0.5, 1, 5]$	$[-2.0, -0.5]$ & $[0.5, 2.0]$	$[-2.0, -1.0]$ & $[1.0, 2.0]$	$[-2.5, -1.0]$ & $[1.0, 2.5]$	$[-3.0, -1.0]$ & $[1.0, 3.0]$	$[-2.0, -1.5]$ & $[1.5, 2.0]$
	$[-3.0, -1.5]$ & $[1.5, 3.0]$	$[-3.5, -1.5]$ & $[1.5, 3.5]$	$[-3.0, -2.0]$ & $[2.0, 3.0]$	$[-4.0, -2.0]$ & $[2.0, 4.0]$	$[-4.0, -3.0]$ & $[3.0, 4.0]$	$[-5.0, -3.0]$ & $[3.0, 5.0]$

Table 4.6: Intervals for initial weights for BP

Parameter	**Values**											
Learning rate	0.001	0.01	0.025	0.0275	0.05	0.075	0.1	0.15	0.2	0.25	0.5	0.75

Table 4.7: Range of values for learning rate for BP

A possible explanation for this behaviour is that the weights selected from the initial interval are too far from the optimal weights, causing too large jumps in weight space. This already illustrates the failure of GD to train PUs. Tables 4.9 and 4.10 on pages 109 and 109 contain the optimal values for the parameters for BP applied to product and summation unit networks, respectively. In tables 4.9 and 4.10, the notation $\pm[1.5, 3.5]$ is an abbreviation for the intervals $[-3.5, -1.5]$ and $[1.5, 3.5]$.

The next section determines the optimal parameters for GAs applied to SUNNs and PUNNs.

Parameter	**Values**											
Momentum	0.0	0.25	0.5	0.6	0.7	0.8	0.9	1.0	1.25	1.5	2.0	5.0

Table 4.8: Range of values for momentum for BP

Backpropagation (SU)			
Function	**weight interval**	**learning rate**	**momentum**
$f(x) = x^2$	$\pm[1.5, 3.5]$	0.01	0.7
$f(x) = x^3 - 0.04x$	$\pm[1.5, 3.0]$	0.0275	0.5

Table 4.9: Best parameters for BP using PUs

Backpropagation (SU)			
Function	**weight interval**	**learning rate**	**momentum**
$f(x) = x^2$	$\pm[0.5, 1.5]$	0.25	2.0
$f(x) = x^3 - 0.04x$	$\pm[0, 3.0]$	0.5	2.0
Henon	$\pm[0, 1.5]$	0.5	1.25
$f(x,y) = y^7x^3 - 0.5x^6$	$\pm[1.5, 3.5]$	0.5	2.0
$f(x,y) = x^2 + y^2$	$\pm[0.5, 1.0]$	0.25	0.7
$f(x,y) = sin(x^2) + sin(y^2)$	$\pm[0.5, 2.0]$	0.15	1.0
Camel	$\pm[0.5, 2.0]$	0.25	1.5
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{xy}$	$\pm[0.5, 1.0]$	0.5	0.9

Table 4.10: Best parameters for BP using SUs

Probability of crossover	0.5	0.6	0.7	0.8	0.9

Table 4.11: Range of values for crossover

Probability mutation	0.001	0.005	0.01	0.05	0.1	0.25	0.5

Table 4.12: Range of values for mutation

4.6.3 Optimal parameters for GA

In determining the optimal parameter values for the GA, 30 simulations were used. Various initial values were used for the probability of crossover and mutation in the GA algorithm. The number of individuals was also increased to determine the optimal size of the population. The range of values for mutation and crossover appear in tables 4.11 and 4.12 follow the guidelines suggested by De Jong [De Jong 1975]. In determining the optimal population size, the number of individuals were gradually increased from 50 to 200 during training.

The optimal values for GAs using SUNNs and PUNNs appear in tables 4.13 and 4.14 on page 111.

The optimal parameters for leapfrog optimization are determined in the next section.

4.6.4 Optimal parameters for LFOP

In determining the best parameters, the average MSE of 30 runs was used during training. During each training session the parameters were fixed to values that appear

Genetic Algorithm - PU			
Function	Probability of crossover	Probability of mutation	Size of population
$f(x) = x^2$	0.6	0.01	50
$f(x) = x^3 - 0.04x$	0.6	0.01	50
Henon	0.7	0.005	100
$f(x,y) = y^7x^3 - 0.5x^6$	0.6	0.01	100
$f(x,y) = x^2 + y^2$	0.7	0.005	120
$f(x,y) = sin(x^2) + sin(y^2)$	0.6	0.005	120
camel	0.8	0.005	200
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	0.7	0.005	200

Table 4.13: Best parameters for GA using PUs

Genetic Algorithm - SU			
Function	Probability of crossover	Probability of mutation	Size of population
$f(x) = x^2$	0.5	0.005	50
$f)(x) = x^3 - 0.04x$	0.8	0.005	50
Henon	0.7	0.005	100
$f(x,y) = y^7x^3 - 0.5x^6$	0.8	0.005	100
$f(x,y) = x^2 + y^2$	0.5	0.005	120
$f(x,y) = sin(x^2) + sin(y^2)$	0.8	0.001	120
camel	0.8	0.001	200
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	0.5	0.001	200

Table 4.14: Best parameters for GA using SUs

Parameter	Values		
δ	10	100	1000

Table 4.15: Range of values for δ

Parameter	Values		
δ_1	0.001	0.01	0.1

Table 4.16: Range of values for δ_1

in tables 4.15 to 4.17.

The optimal values for LFOP applied to PUNNs and SUNNs appear in tables 4.18 and 4.19 respectively. An entry of '$-$' in table 4.18 indicates that all simulations produced overflows and optimal values could not be determined in these cases. The overflows can again be ascribed to the fact that gradient methods suffer from an explosion in the growth of weight values due to large derivatives.

The next section summarizes the initial NN architectures used in training each function using SUNNs and PUNNs.

4.7 Initial Neural Network Architectures

Tables 4.20 and 4.21 contain the initial neural network architectures that were used in optimizing the parameters for each of PSO, GA, LFOP and BP using PUNNs and SUNNs. The oversized networks of tables 4.20 and 4.21 are used to determine the

Parameter	Values			
Δt	0.01	0.05	0.075	0.1

Table 4.17: Range of values for Δt

Leapfrog Algorithm - PU			
Function	δ	δ_1	Δt
$f(x) = x^2$	10.0	0.01	0.05
$f(x) = x^3 - 0.04x$	100.0	0.01	0.01
Henon	10.0	0.01	0.05
$f(x,y) = y^7x^3 - 0.5x^6$	10.0	0.001	0.01
$f(x,y) = x^2 + y^2$	10.0	0.01	0.01
$f(x,y) = sin(x^2) + sin(y^2)$	–	–	–
camel	–	–	–
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	–	–	–

Table 4.18: Best parameters for LFOP using PUs

Leapfrog Algorithm - SU			
Function	δ	δ_1	Δt
$f(x) = x^2$	100.0	0.001	0.075
$f(x) = x^3 - 0.04x$	10.0	0.001	0.01
Henon	100.0	0.001	0.05
$f(x,y) = y^7x^3 - 0.5x^6$	10.0	0.001	0.01
$f(x,y) = x^2 + y^2$	10.0	0.001	0.075
$f(x,y) = sin(x^2) + sin(y^2)$	100.0	0.001	0.01
camel	100.0	0.001	0.1
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	100.0	0.001	0.075

Table 4.19: Best parameters for LFOP using SUs

Configuration for PUNNs	
Function	**Initial Configuration**
$f(x) = x^2$	$1:4:1$
$f(x) = x^3 - 0.04x$	$1:5:1$
Henon	$2:6:1$
$f(x,y) = y^7x^3 - 0.5x^6$	$2:5:1$
$f(x,y) = x^2 + y^2$	$2:4:1$
$f(x,y) = sin(x^2) + sin(y^2)$	$2:10:1$
Camel	$2:10:1$
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	$2:12:1$

Table 4.20: Initial configuration for PUNNs

optimal parameters for the different global optimization algorithms.

The next section discusses the procedure to determine the optimal number of hidden units for SUNNs and PUNNs.

4.8 Best Configuration for SUNNs and PUNNs

Once the optimal parameters for each of the global optimization algorithms and BP have been determined, pruning by 'brute force' is then applied to the optimization algorithms using the optimal parameters of section 4.6 to find near optimal configurations for PUNNs and SUNNs. The optimal configurations for PUNNs and SUNNs were determined by comparing results of experiments where the number of hidden units varied for each training session. The architecture that produced the smallest average MSE on the test set over 30 simulations and the highest number of simulations that converged to a specified generalization level of 0.001 was accepted as the best configuration. Table 4.22 contains the results for PUNNs, and table 4.23 for SUNNs, where training started with the initial configuration tabulated. After each training

Configuration for SUNNs	
Function	**Initial Configuration**
$f(x) = x^2$	1 : 4 : 1
$f(x) = x^3 - 0.04x$	1 : 5 : 1
Henon	2 : 8 : 1
$f(x, y) = y^7x^3 - 0.5x^6$	2 : 10 : 1
$f(x, y) = x^2 + y^2$	2 : 6 : 1
$f(x, y) = sin(x^2) + sin(y^2)$	2 : 10 : 1
Camel	2 : 12 : 1
$f(x, y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	2 : 15 : 1

Table 4.21: Initial configuration for SUNNs

session, consisting of 30 simulations, the number of hidden units was decreased by 1 and training was re-started on this smaller network until the performance on the test set deteriorated by 20% or more compared to the initial oversized network. The number of hidden units that occurred in optimal PUNNs expressed as a percentage of the number of hidden units that occurred in the equivalent optimal SUNNs, are for functions F1 50%, F2 33.3%, F3 80%, F4 33.3%, F5 50%, F6 66.7%, F7 75% and F8 77.8%.

The optimal configurations or architectures for each function obtained in this section is subsequently used in the remainder of this thesis in experiments that compare the various global optimization algorithms and BP. In chapter 5 the variance nullity pruning algorithm is applied to the oversized PUNNs to determine optimal architectures. The results obtained in chapter 5 will then be compared to the results obtained by brute force pruning.

Configuration for PUNNs		
Function	**Initial Configuration**	**Best Configuration**
$f(x) = x^2$	1 : 4 : 1	1 : 1 : 1
$f(x) = x^3 - 0.04x$	1 : 5 : 1	1 : 1 : 1
Henon	2 : 6 : 1	1 : 4 : 1
$f(x, y) = y^7x^3 - 0.5x^6$	2 : 5 : 1	1 : 2 : 1
$f(x, y) = x^2 + y^2$	2 : 4 : 1	2 : 2 : 1
$f(x, y) = sin(x^2) + sin(y^2)$	2 : 10 : 1	2 : 4 : 1
Camel	2 : 10 : 1	2 : 6 : 1
$f(x, y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	2 : 12 : 1	2 : 7 : 1

Table 4.22: Configuration for PUNNs

Configuration for SUNNs		
Function	**Initial Configuration**	**Best Configuration**
$f(x) = x^2$	1 : 4 : 1	1 : 2 : 1
$f(x) = x^3 - 0.04x$	1 : 5 : 1	1 : 3 : 1
Henon	2 : 8 : 1	1 : 5 : 1
$f(x, y) = y^7x^3 - 0.5x^6$	2 : 10 : 1	1 : 6 : 1
$f(x, y) = x^2 + y^2$	2 : 6 : 1	2 : 4 : 1
$f(x, y) = sin(x^2) + sin(y^2)$	2 : 10 : 1	2 : 6 : 1
Camel	2 : 12 : 1	2 : 8 : 1
$f(x, y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	2 : 15 : 1	2 : 9 : 1

Table 4.23: Configuration for SUNNs

4.9 Comparison Between PUNNs Containing Bias and Distortion Units

The MSEs of two PUNNs, one containing a bias and the other a distortion unit in the hidden layer, are compared in this section. Both architectures contained a bias in the output layer. The objective is to determine whether there is any gain in performance when using a distortion unit compared to a bias unit in the hidden layer of a PUNN. PSO was used to compare the resulting MSEs of these two architectures of PUNNs. First, optimal parameters were determined for both type of architectures (similarly to section 4.6.1 on page 105) for the following three functions: $f(x) = x^3 - 0.04x$, $f(x,y) = y^7x^3 - 0.5x^6$ and $f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$. Subsequently, PSO with optimal parameter values for each type of PUNN architecture (i.e. bias or distortion unit PUNN) was trained to approximate each of the three functions for a maximum of 500 epochs. The average MSEs on the training and test sets over 30 simulations together with a 95% confidence interval for the three functions are displayed in Table 4.24.

The results of the tests show that PUNNs with a distortion unit produced smaller MSEs on the training set and generalized much better than PUNNs that contain a bias in the hidden layer. The PUNN with distortion unit is therefor chosen as the PUNN architecture for implementation in the remainder of this thesis.

Function	PUNN using a bias unit		PUNN using a distortion unit	
	MSE on Training set	MSE on Test set	MSE on Training set	MSE on Test set
$f(x) = x^3 - 0.04x$	0.002970 $\pm$0.001128	0.002785 $\pm$0.000967	0.000018 $\pm$0.0000043	0.000016 $\pm$0.0000042
$f(x, y) = y^7x^3 - 0.5x^6$	0.002574 $\pm$0.000777	0.002840 $\pm$0.00079	0.000919 $\pm$0.000476	0.001213 $\pm$0.000625
$f(x, y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	0.002383 $\pm$0.001665	0.002614 $\pm$0.001841	0.0005684 $\pm$0.0004004	0.0007953 $\pm$0.0005696

Table 4.24: Comparison of MSEs on PUNN containing a bias and a PUNN containing a distortion unit

4.10 Comparison of Global Optimization Algorithms

This section uses the optimized parameters from section 4.6.1 to compare the performance of PSO, GA, LFOP and BP on both PUNNs and SUNNs.

Tables 4.25 and 4.26 summarize the average mean squared error (MSE) for the training and test sets for PUNNs and SUNNs, respectively, after 500 training passes for each of the algorithms, PSO, LFOP, GA and BP, together with 95% confidence intervals. A '$\star$' in tables 4.25 and 4.26 indicates the algorithm that performed the best in each case of PUNNs and SUNNs, respectively. A '†' in tables 4.25 and 4.26 indicates the algorithm that performed the best in both, PUNNs and SUNNs, cases.

Figures 4.3, 4.4 and 4.5 illustrate the learning profiles for each optimization method for the training and test sets (as a measure of generalization). Tables 4.27 to 4.29 list the average number of epochs to reach specified MSE levels on the training set. The entries in the first column for the tables 4.27 to 4.29 refer to (a) the type of algorithm and (b) the type of network used, i.e. BP:SU refers to back-propagation by gradient descent applied to a summation unit neural network. Similarly, LFOP:PU refers to leapfrog optimization applied to a product unit neural network. Tables 4.30 to 4.33 summarize, for different generalization levels (i.e. the MSE on the test set), the percentage of simulations that did converge to these generalization levels. A '$-$' entry in table 4.25 implies that not a single simulation out of the 30 simulations produced any result other than overflows. In tables 4.27 to 4.29, a '$-$' entry means that not a single simulation reached convergence within the maximum of 1000 epochs allowed. In tables 4.32 and 4.33, a '$-$' indicates that all simulations produced overflows.

The fact that only the quadratic and cubic functions produced results for BP using product units, is an indication of the difficulty that is associated with backpropagation by gradient descent when applied to PUs. LFOP for training PUs, a derivative based algorithm, also produced overflows for the functions $f(x,y) = sin(x^2) + sin(y^2)$, $f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$ and the camel function as reflected in table 4.32.

4.11 Analysis of Results

The results in this section were produced by experiments that used optimal architectures as determined in section 4.8 for both PUNNs and SUNNs. Table 4.25 indicates that PSO produced better MSEs on both the training and test sets in training PUNNs compared to the other global optimization algorithms. GAs also performed fairly well, but not as good as PSO. In table 4.26 LFOP using SUs produced the smallest MSEs on both the training and test sets, except for $f(x) = x^2$ and $f(x,y) = x^2 + y^2$, where BP using SUs produced much better results. Thus, in training SUNNs, LFOP would be the recommended optimization algorithm.

Next, the results for each function are discussed separately.

$\underline{f(x) = x^2}$

The graphs in figure 4.3 show that PSO using PUs and GA using PUs started off with fairly low MSEs on both training and test sets. All the global optimization algorithms, (i.e. PSO, LFOP and GA) produced substantially better training errors and generalization than the equivalent SUs for the quadratic function as shown in tables 4.25 and 4.26. BP:SUs (read as back-propagation using summation units) performed slightly better than BP:PUs (read as back-propagation using PUs). However, the

Function	Algorithm	Average Mean Squared Error		
		Training set	Test set	
$f(x) = x^2$	BP	0.006823 ± 0.004255	0.005949 ± 0.003679	
	PSO	0.000344 ± 0.000112	0.000334 ± 0.000112	$\star$ †
	GA	0.000518 ± 0.000518	0.001340 ± 0.002131	
	LFOP	0.001583 ± 0.000725	0.001930 ± 0.000893	
$f(x) = x^3 - 0.04x$	BP	0.001210 ± 0.000696	0.000977 ± 0.000510	
	PSO	0.000018 ± 0.0000043	0.000016 ± 0.0000042	$\star$ †
	GA	0.000072 ± 0.0000528	0.000082 ± 0.0000585	
	LFOP	0.000122 ± 0.000141	0.000156 ± 0.0001863	
Henon	BP	–	–	
	PSO	0.003173 ± 0.001754	0.007881 ± 0.004455	
	GA	0.004365 ± 0.002613	0.006101 ± 0.004645	
	LFOP	0.000651 ± 0.000698	0.000608 ± 0.000651	$\star$
$f(x,y) = y^7x^3 - 0.5x^6$	BP	–	–	
	PSO	0.000919 ± 0.000476	0.001213 ± 0.000625	$\star$
	GA	0.0021079 ± 0.0007105	0.0022844 ± 0.0007620	
	LFOP	0.0043720 ± 0.0005151	0.0049928 ± 0.0004130	
$f(x,y) = x^2 + y^2$	BP	–	–	
	PSO	0.0088368 ± 0.0028955	0.0083125 ± 0.0027094	$\star$
	GA	0.0094581 ± 0.0025222	0.0089599 ± 0.0023852	
	LFOP	0.0200444 ± 0.0054125	0.0195222 ± 0.0050495	
$f(x,y) = sin(x^2) + sin(y^2)$	BP	–	–	
	PSO	0.0021190 ± 0.0009562	0.0041067 ± 0.0031883	$\star$ †
	GA	0.005998 ± 0.002610	0.005807 ± 0.002668	
	LFOP	–	–	
camel	BP	–	–	
	PSO	0.0316965 ± 0.0026735	0.0398755 ± 0.0032916	$\star$
	GA	0.0509293 ± 0.0037854	0.0632008 ± 0.0059496	
	LFOP	–	–	
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	BP	–	–	
	PSO	0.0005684 ± 0.0004004	0.0007953 ± 0.0005696	$\star$ †
	GA	0.0007359 ± 0.0002295	0.0009474 ± 0.0002870	
	LFOP	–	–	

Table 4.25: Mean squared error results for PUs

Function	Algorithm	Average Mean Squared Error		
		Training set	Test set	
$f(x) = x^2$	BP	0.001477 ± 0.001323	0.001434 ± 0.001023	$\star$
	PSO	0.001945 ± 0.00172	0.001873 ± 0.001355	
	GA	0.005696 ± 0.002512	0.006965 ± 0.003288	
	LFOP	0.006117 ± 0.003559	0.009518 ± 0.005372	
$f(x) = x^3 - 0.04x$	BP	0.001695 ± 0.000498	0.001957 ± 0.000501	
	PSO	0.000095 ± 0.0000254	0.000165 ± 0.0000496	
	GA	0.000410 ± 0.0002135	0.000525 ± 0.0002425	
	LFOP	0.000053 ± 0.0000177	0.000065 ± 0.0000197	$\star$
Henon	BP	0.000800 ± 0.000773	0.000926 ± 0.000810	
	PSO	0.0004200 ± 0.0000490	0.0004050 ± 0.0000450	
	GA	0.0016431 ± 0.0003998	0.0017022 ± 0.0003907	
	LFOP	0.0002004 ± 0.0000232	0.0001943 ± 0.0000231	$\star$ †
$f(x, y) = y^7x^3 - 0.5x^6$	BP	0.000493 ± 0.000061	0.002172 ± 0.000637	
	PSO	0.001086 ± 0.000089	0.003604 ± 0.000953	
	GA	0.0014528 ± 0.0001069	0.0030766 ± 0.0005732	
	LFOP	0.0003959 ± 0.0000482	0.0007645 ± 0.0000765	$\star$ †
$f(x, y) = x^2 + y^2$	BP	0.000413 ± 0.000013	0.000529 ± 0.000019	$\star$ †
	PSO	0.001860 ± 0.000776	0.002164 ± 0.000753	
	GA	0.0096575 ± 0.0020188	0.0143728 ± 0.0029755	
	LFOP	0.0010507 ± 0.0010435	0.0012780 ± 0.0012015	
$f(x, y) = sin(x^2) + sin(y^2)$	BP	0.005595 ± 0.000883	0.006722 ± 0.001068	
	PSO	0.008542 ± 0.000572	0.011924 ± 0.000804	
	GA	0.012365 ± 0.001019	0.014279 ± 0.001087	
	LFOP	0.004477 ± 0.001194	0.005483 ± 0.001184	$\star$
camel	BP	0.000560 ± 0.000090	0.002256 ± 0.000702	
	PSO	0.001228 ± 0.000217	0.002044 ± 0.000344	
	GA	0.005017 ± 0.000917	0.006842 ± 0.000809	
	LFOP	0.0000535 ± 0.0000022	0.0000963 ± 0.0000058	$\star$ †
$f(x, y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	BP	0.008764 ± 0.000144942	0.010274 ± 0.000189	
	PSO	0.010991 ± 0.000368	0.012457 ± 0.000309	
	GA	0.0111821 ± 0.0003309	0.0125407 ± 0.0002641	
	LFOP	0.004335 ± 0.000975	0.005601 ± 0.001115	$\star$

Table 4.26: Mean squared error results for SUs

PUNN used a smaller architecture; the SUNN consisted of 2 hidden units, whereas the PUNN contained only 1 hidden unit. Table 4.27 shows that PSO:PUs, GA:PUs and LFOP:PUs required substantially fewer epochs than its SU equivalents to reach each of the generalization levels, 0.01 and 0.001, with LFOP:PUs having the least number of epochs for each of the generalization levels. Not one of the global optimization algorithms using SUs could generalize up to a level of 0.0001; these algorithms could only manage to generalize up to a level of 0.001, with convergence observed in 46.7% of the simulations of PSO:SUs. All algorithms using PUs were able to reach an MSE of 0.0001 on the training sets as shown in tables 4.30 to 4.33. For the quadratic function, GA:PUs and LFOP:PUs managed to generalize up to a low level of 0.00001 with LFOP:PUs having the highest percentage of simulations (47.1%) that converged to this low level followed by GA where 40% of the simulations converged. The average number of simulations required by LFOP:PUs and GA:PUs to reach this level of generalization are 90.1 and 301.5, respectively. Tables 4.25 and 4.26 show that GA using PUs and LFOP using SUs, overfitted the data, as indicated by the MSE on the test set compared to the MSE on the training set. In both cases the MSE on the test set exceeded the MSE on the training set, indicating overfitting of the data. Thus, PSO:PUs is the recommended algorithm for training the quadratic function.

$f(x) = x^3 - 0.04x$

The graphs in figure 4.3 show that GA:PUs started off with very small MSEs on both training and test sets. Once, again all the global optimization algorithms produced a smaller MSE on the training set than its SU equivalent, except for LFOP. The optimal architecture of PUNNs for this function contained 1 hidden unit compared to the 3 hidden units in the case of SUNNs. LFOP using SUs with the larger network than its PUs equivalent had a much lower MSE on the training and test sets than LFOP:PUs as shown in tables 4.25 and 4.26. Table 4.26 indicates that all the algorithms that used

SUs overfitted the data. Tables 4.25 and 4.26, further show that BP:PUs produced results similar to BP:SUs, with BP:PUs being slightly better than BP:SUs. BP:SUs exhibits a generalization that is much lower than the MSE on the training set. It should however also be borne in mind that the PUNN contained 1 hidden unit and is much smaller than the SUNN with 3 hidden units. Table 4.27 shows that PSO:PUs, GA:PUs and LFOP:PUs required fewer epochs than its SU equivalents to reach the generalization level of 0.001, 0.0001 and 0.00001, with LFOP:PUs having the least number of epochs (314.7 ± 64.03) to reach 0.00001. Not one of the global optimization algorithms using SUs could generalize up to a level of 0.00001. Global optimization algorithms using SUNNs could only manage to generalize up to 0.0001, with 33.3% of the simulations of PSO converging to this low generalization level. The results of the cubic function using GA with PUs are significantly better than GA using SUs as reflected in Tables 4.25 and 4.26. PSO:PUs produced the best training and test error compared to all the other algorithms including SUNNs. It is interesting to note that although LFOP:PUs has a larger number of simulations that converged to a level of 0.00001 as indicated in table 4.32, it has an average MSE much greater than PSO:PUs. PSO:PUs is thus recommended for training the function $f(x) = x^3 - 0.04x$.

<u>$f(z) = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2$ (henon)</u>

The graphs in 4.3 reflect fairly low MSEs for LFOP using PUs and BP using SUs on both training and test sets early in training. The global optimization algorithms applied to the SUs outperformed the PUs as indicated in tables 4.25 and 4.26, with the results of the optimization algorithms of SUs having a much smaller variance than its PU equivalents. LFOP:SUs had the lowest training error and generalization than the optimization algorithms applied to PUs. Note that LFOP:SUs had a larger network (2:5:1) with a training error of 0.0002004 ± 0.0000232, compared to LFOP:PUs, 2:4:1 network that produced a training error of 0.000651 ± 0.000698. Interestingly, In the case of the SUs the global optimization algorithms PSO:SUs and LFOP:SUs gave better training and generalization than BP:SUs (which is a local optimizer). BP:PUs did not produce a single result, except for overflows (as indicated to by '-' in table 4.31), in all the training sessions. LFOP using PUs required the least number of epochs of all the optimization algorithms to reach the various generalization levels for the henon time series. The LFOP:PUs as shown in table 4.27 used much less epochs than LFOP:SUs to reach generalization levels 0.01, 0.001, and 0.0001. Only LFOP:PUs and PSO:PUs managed to reach low generalization levels of 0.00001, with convergence reached in only 1.4% of the simulations in LFOP:PUs. In PSO:PUs slightly more simulations (3.3%) converged than LFOP:PUs at this low level of generalization. However, LFOP:PUs only needed 63.3 epochs compared to 465.3 epochs required by PSO:PUs to reach this low level. For a generalization of 0.0001, 35.7% of simulations for LFOP:PUs converged compared to only 10% convergence in the case of PSO:PUs. In the case of SUNNs only the LFOP managed to reach a level of 0.0001, with convergence in 13.3% of the simulations. PSO and GA using PUs overfitted the data as reflected in table 4.25. LFOP:SUs, with a low training and test error (see table 4.25) is recommended as training algorithm for the henon time series.

$\underline{f(x,y) = y^7x^3 - 0.5x^6}$

PSO using PUs produced small MSEs on the training set but overfitted the data as shown by MSEs on the test set in table 4.25. Similarly, BP:SUs produced low training errors but did not generalize equally well. GAs:SUs performance was similar to GA:PUs. LFOP:SUs produced the best MSEs on the training and test sets of all the algorithms. LFOP:SUs, however, could not reach a generalization level of 0.0001 (all 30 simulations ended in MSEs on the test sets between 0.0002 and 0.0007), whereas 13.3% simulations of PSO:PUs achieved 0.00001 (refer to table 4.30). PSO using PUs is the only algorithm that managed to reach a generalization level of 0.00001 as shown in table 4.30. The lowest generalization level achieved by SUNNs is 0.001; with convergence of 63.3% and 6.7% of simulations for LFOP and BP, respectively. Also, did PSO:PUs and GA:PUs require fewer epochs than its SU equivalent to reach a generalization level of 0.001. BP:PUs did not produce any results, other than overflows in all training sessions. LFOP:SUs with its much lower MSEs on training and test sets, as tabulated in 4.26 is recommended as the optimization algorithm for the function $f(x,y) = y^7x^3 - 0.5x^6$.

$\underline{f(x,y) = x^2 + y^2}$

PSO:PUs reached lower generalization levels than PSO:SUs as indicated in table 4.28. Table 4.26 shows that BP:SUs produced the smallest MSEs on the training and test sets, but could only generalize up to a level of 0.001, despite the fact that the generalization in table 4.26 is 0.000529 ± 0.000019; the average of all simulations is 0.000529 with not a single simulation reaching an MSE on the test set lower than 0.0001 (all simulations ended with values greater than 0.0001 but smaller than 0.0009). However, both PSO:PUs and LFOP:PUs managed to reach lower generalization levels than BP:SUs as shown in table 4.28. The lowest generalization level achieved by SUNNs is 0.001, with 16.7% of the simulations of BP:SUs and 13.3% of the simulations of PSO:SUs converging at this low level. However, all the global optimization algorithms using PUs managed to reach generalization levels of 0.001 and 0.0001, with 33.3% of PSO:PUs and 15.7% of LFOP:PUs generalizing up to a level of 0.00001. PSO:PUs had twice as many simulations than LFOP:PUs that converged to this low level of generalization as shown in tables 4.30 and 4.32. Of all the global optimization algorithms, LFOP:SUs, not forgetting the bigger network (i.e. 2:4:1) than the equivalent PUNN (2:2:1), produced the lowest MSE on the training and test sets within the allowed 500 epochs as reflected in table 4.25. LFOP:SUs, however, could not manage to reach a generalization level lower than 0.001 within the 1000 epochs allowed. Neither

BP:PUs nor LFOP:PUs produced any results. BP:PUs, with the smallest MSE average as reflected in table 4.26, is recommended for training the function $f(x,y) = x^2 + y^2$.

$f(x,y) = sin(x^2) + sin(y^2)$

The graphs in figure 4.4 show that PSO using PUs produced large MSEs early in training, but eventually had the lowest MSEs of all the algorithms when training terminated. In this case PSO:PUs produced the lowest training error as indicated in table 4.25. GA:PUs and LFOP:SUs produced similar training errors. Table 4.28 shows that only PSO:PUs and LFOP:SUs managed to reach generalization levels of 0.0001, with PSO:PUs taking fewer epochs to achieve this generalization level. In both cases only 3.3% simulations converged. BP:PUs and LFOP:PUs did not produce any results. PSO:PUs with its good generalization ability is recommended as global optimization algorithm for the function $f(x,y) = sin(x^2) + sin(y^2)$.

$f(x,y) = 4 - 2.1x^2 + (\frac{x^3}{3})x^2 + xy + (4y^2 - 4)y^2$(camel)

The graphs in figure 4.5 show that all algorithms using SUs had small MSEs early in training and that not one of the algorithms using PUs managed to reach MSE levels lower than 0.03. The SUs performed much better than the PUs in this case. The LFOP:SUs produced the smallest MSEs on the training and test sets. LFOP:SUs was able to achieve a generalization level of 0.0001 (refer to table 4.29). BP:SUs also produced very good MSEs but indicated overfitting of the data in table 4.25. BP:PUs and LFOP:PUs did not produce any results, other than overflows in all the training sessions. LFOP:SUs is recommended for training of the camel function.

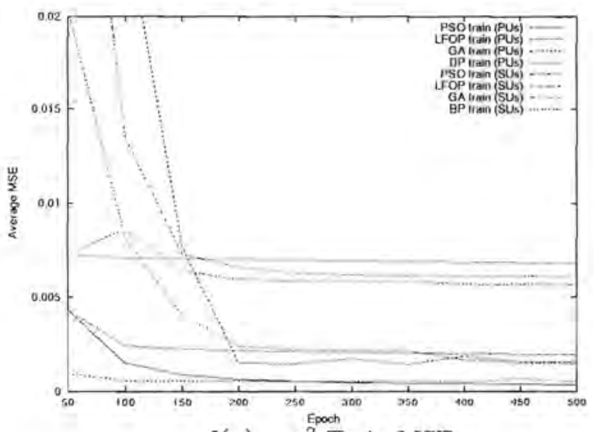
$f(x,y) = sin(x)sin(y)\sqrt{x \cdot y}$

The graphs in figure 4.5 clearly show that PSO using PUs had the lowest MSEs early in training and ended the training session with the lowest MSEs on both the training and test sets. PSO:PUs and GA:PUs outperformed all the other algorithms. PSO:PUs is the only algorithm that managed to generalize up to a level of 0.00001, GA:PUs could only generalize up to 0.001 (with MSEs on the test set ranging between 0.0002 and 0.0006) simulations. The lowest generalization level achieved by SUNNs is 0.01, with 15.7% of simulations converging at this level. In the case of SUNNs, LFOP:SUs BP:PUs and LFOP:PUs did not produce any results, other than overflows. Only PSO:PUs managed to generalize up to level 0.00001 (refer to table 4.29), with only 3.3% simulations converging as reflected in table 4.30.

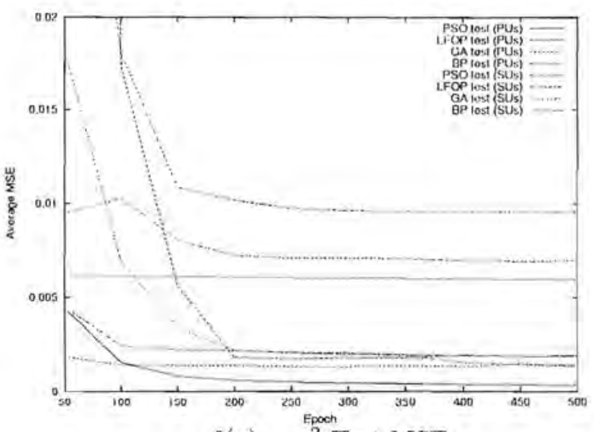
Table 4.27 shows that LFOP:PUs had the most simulations that converged to the different generalization levels, especially for the low generalization level of 0.00001. BP:SUs did manage to have simulations that converged to a generalization level up to 0.001 but not a single simulation converged to a MSE on the test set lower than 0.0001. Figures 4.3 to 4.5 illustrate that PSO:PUs and GA:PUs have larger reductions in error early in training reaching low errors using substantially less training epochs. LFOP:PUs has shown to use much less epochs than do PSO and GA for low generalization levels of 0.001, 0.0001 and 0.00001.

4.12 Conclusion

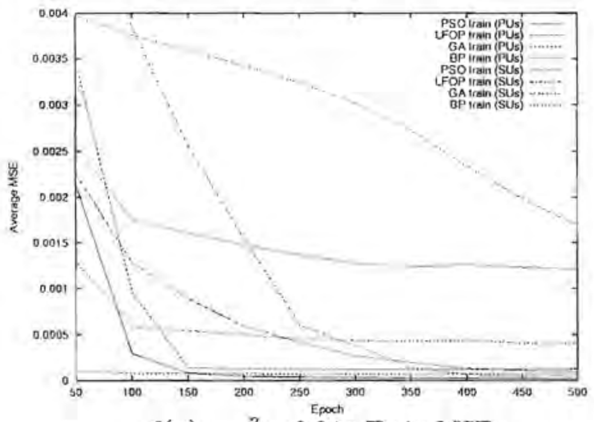
PUNNs compared favourably with SUNNS with respect to functions F1, F2, F6 and F8. However, SUNNs performed much better on functions F3, F4, F5 and F7. Generally, LFOP using SUs, produced a much smaller training error than BP:SUs. LFOP:SUs also generalized far better than BP:SUs. LFOP:SUs produced much smaller MSEs than BP:SUs in training the test functions, except for functions $f(x) = x^2$ and $f(x,y) = x^2 + y^2$, where BP:SUs outperformed PSO:SUs and LFOP:SUs. Although the global optimization algorithms did not perform better than BP:SUs in all cases, it did however manage to achieve lower generalization levels using much fewer epochs than BP:SUs with a corresponding higher convergence than BP:SUs. Thus, global optimization algorithms tend to find the best minimum on the error surface faster than BP:SUs. PSO:PUs is the only algorithm that managed to reach a low generalization level of 0.0001 in all functions except for the camel function. Although, BP:SUs applied to function F5 had a smaller training error than PSO:PUs, it did not manage to reach the low generalization level of 0.0001, that was achieved by PSO:PUs. The results also show that global optimization algorithms can reach lower generalization levels than BP when applied to SUNNs. The tests have also indicated that PUNNs are not always an improvement over SUNNs, even though PUs may produce smaller networks. These smaller networks do not always produce good training errors and generalization compared to the slightly bigger SUNNs. However, global optimization using SUs showed an improvement in performance compared to back-propagation. In certain cases the PUNNs ($f(x,y) = x^2$, $f(x,y) = x^3 - 0.04x$, $f(x,y) = sin(x^2) + sin(y^2)$ and $f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{(x \cdot y)}$) outperformed the SUNNs. PSO appears to be more robust with respect to functions F1, F2 and F5 since they have a larger percentage of simulations that converged to a generalization level of 0.00001 than LFOP:PUs. In general, PUNNs did not show a remarkable gain in performance, other than reaching lower generalization levels faster than back-propagation.



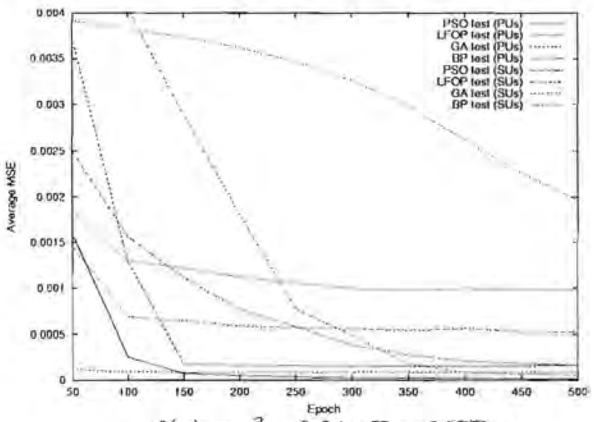
$f(x) = x^2$ Train MSE



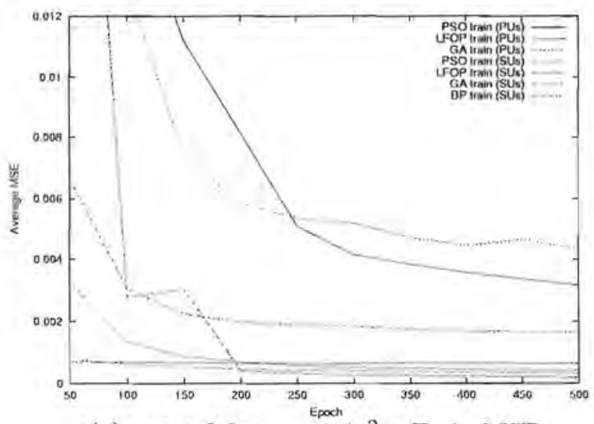
$f(x) = x^2$ Test MSE



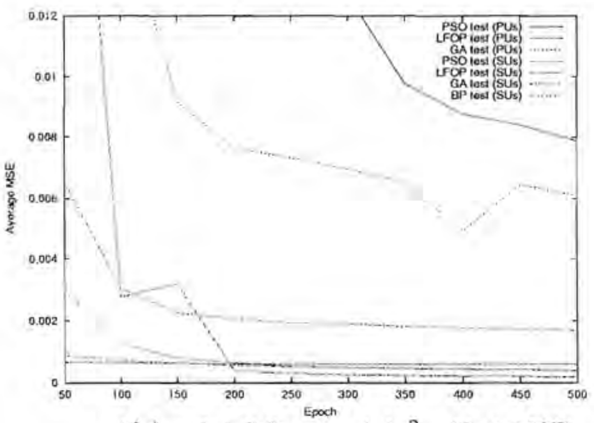
$f(x) = x^3 - 0.04x$ Train MSE



$f(x) = x^3 - 0.04x$ Test MSE

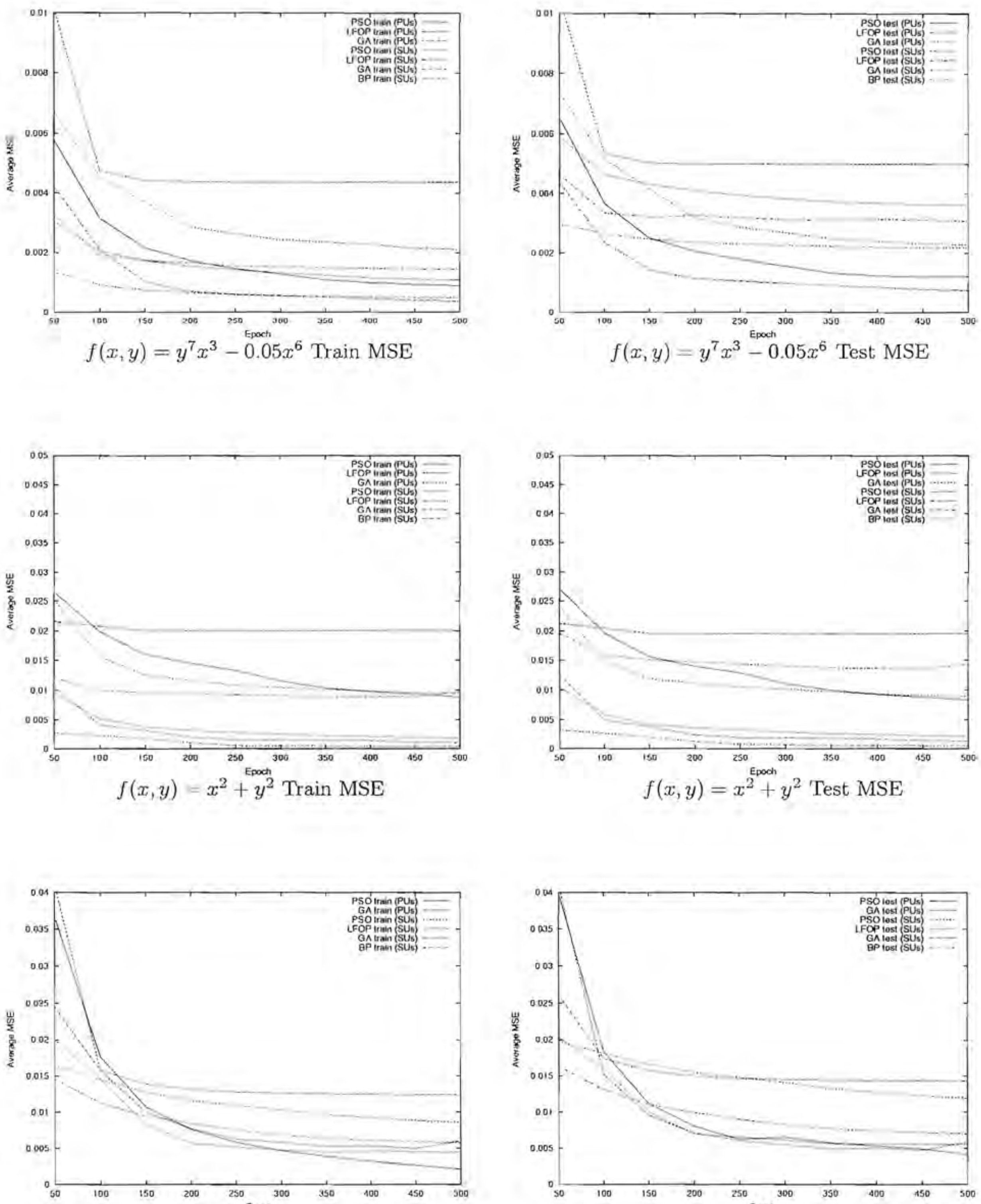


$f(z) = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2$ Train MSE



$f(z) = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2$ Test MSE

Figure 4.3: Learning profiles for functions F1, F2 and F3



$f(x, y) = y^7x^3 - 0.05x^6$ Train MSE

$f(x, y) = y^7x^3 - 0.05x^6$ Test MSE

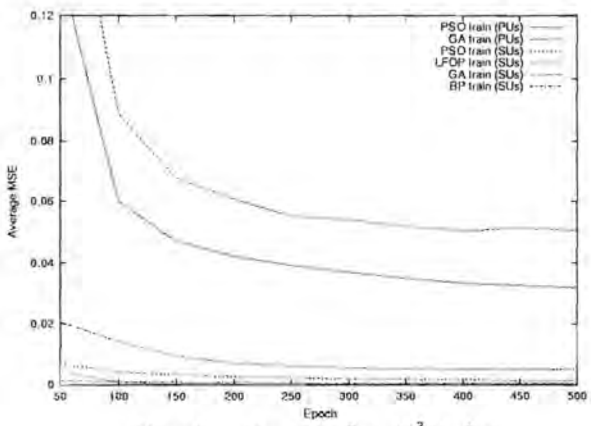
$f(x, y) = x^2 + y^2$ Train MSE

$f(x, y) = x^2 + y^2$ Test MSE

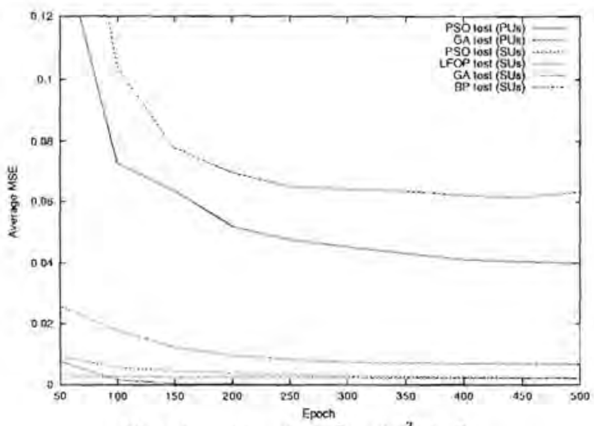
$f(x, y) = sin(x^2) + sin(y^2)$ Train MSE

$f(x, y) = sin(x^2) + sin(y^2)$ Test MSE

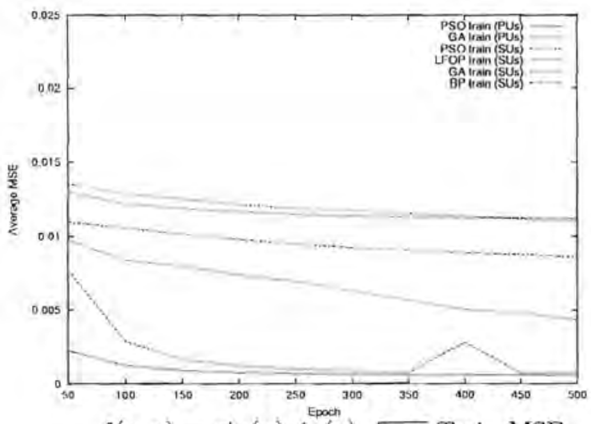
Figure 4.4: Learning profiles for functions F4, F5 and F6



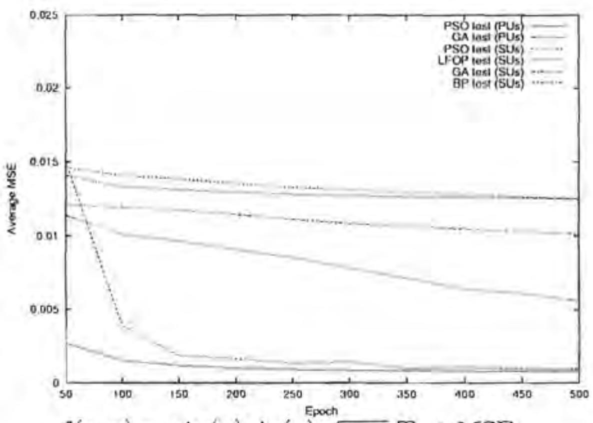
$f(x,y) = (4 - 2.1x^2 + (\frac{x^3}{3}))x^2 +$
$xy + (4y^2 - 4)y^2$ Train MSE



$f(x,y) = (4 - 2.1x^2 + (\frac{x^3}{3}))x^2 +$
$xy + (4y^2 - 4)y^2$ Test MSE



$f(x,y) = sin(x)sin(y)\sqrt{x \cdot y}$ Train MSE



$f(x,y) = sin(x)sin(y)\sqrt{x \cdot y}$ Test MSE

Figure 4.5: Learning profiles for functions F7 and F8

$f(x) = x^2$							
MSE	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
BP:SU	1.0 ± 0	1.0 ± 0.07	6.8 ± 4.00	108.5 ± 32.61	227.6 ± 53.72	–	–
BP:PU	2.2 ± 0.10	3.9 ± 0.48	17.8 ± 23.06	101.7 ± 52.95	216.2 ± 76.33	216.2 ± 76.33	395.4 ± 58.01
PSO:SU	1.0 ± 0	1.0 ± 0	1.4 ± 0.36	101.7 ± 58.73	228.5 ± 62.69	–	–
PSO:PU	1.0 ± 0	1.0 ± 0	1.3 ± 0.25	38 ± 19.84	154.5 ± 44.09	485.6 ± 28.16	–
GA:SU	1.0 ± 0	1.0 ± 0	1.2 ± 0.27	40.0 ± 31.85	494.9 ± 9.93	–	–
GA:PU	1.1 ± 0.09	1.2 ± 0.17	2 ± 0.57	8.8 ± 2.15	95.4 ± 58.81	211.3 ± 77.02	301.5 ± 74.58
LFOP:SU	2.4 ± 0.41	7.4 ± 1.64	9.6 ± 5.60	192.6 ± 99.35	273 ± 53.80	–	–
LFOP:PU	5.0 ± 0.69	10.3 ± 1.45	13.7 ± 1.69	78.4 ± 46.30	143.3 ± 62.80	173.9 ± 59.92	90.1 ± 40.11

$f(x) = x^3 - 0.04x$							
MSE	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
BP:SU	1.0 ± 0	1.0 ± 0.00	1.0 ± 0.00	6.3 ± 0.52	448.6 ± 22.51	–	–
BP:PU	1.9 ± 0.09	2.8 ± 0.31	48.8 ± 49.45	36.8 ± 29.00	224.5 ± 56.78	377.5 ± 50.54	408.5 ± 50.46
PSO:SU	1.0 ± 0	1.0 ± 0	1.1 ± 0.13	7.0 ± 0.55	144.7 ± 38.43	330.4 ± 50.85	–
PSO:PU	1.0 ± 0	1.0 ± 0	1.7 ± 0.43	16 ± 3.32	63.1 ± 12.01	138.4 ± 30.03	422.5 ± 40.42
GA:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	5.1 ± 1.50	91.1 ± 49.06	397.0 ± 64.84	–
GA:PU	1 ± 0	1.4 ± 0.26	2.8 ± 0.48	11.1 ± 2.81	105.8 ± 92.27	91.3 ± 50.08	485.5 ± 28.48
LFOP:SU	5.4 ± 1.12	16.7 ± 7.31	18.0 ± 3.94	43.3 ± 7.94	185 ± 11.17	310.7 ± 25.15	–
LFOP:PU	15.3 ± 2.19	22.4 ± 1.81	24.1 ± 2.15	30.8 ± 2.36	166.6 ± 68.53	159.6 ± 58.87	314.7 ± 64.03

$z_t = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2$ (henon)							
MSE	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
BP:SU	30.5 ± 3.26	53.2 ± 10.46	63.0 ± 6.68	85.0 ± 12.56	123.2 ± 58.01	152.5 ± 68.35	–
BP:PU	–	–	–	–	–	–	–
PSO:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	26.6 ± 4.84	147.9 ± 37.05	–	–
PSO:PU	2.4 ± 1.26	31.9 ± 4.27	83.9 ± 64.31	116 ± 49.06	233.6 ± 58.26	371.3 ± 53.32	465.3 ± 31.72
GA:SU	1.0 ± 0	1.0 ± 0	1.2 ± 0.18	31.8 ± 3.76	436.2 ± 47.46	–	–
GA:PU	5.1 ± 0.77	21.9 ± 3.66	45.4 ± 13.22	109.2 ± 32.05	357.1 ± 64.38	–	–
LFOP:SU	2.9 ± 0.51	15.3 ± 3.74	21.6 ± 5.28	73.0 ± 3.64	112 ± 8.70	480.4 ± 23.54	–
LFOP:PU	3.8 ± 0.53	15.2 ± 4.43	15.3 ± 2.35	38.4 ± 29.29	32.2 ± 2.38	69.8 ± 41.89	63.3 ± 28.65

Table 4.27: Epochs needed to reach MSE levels

$f(x,y) = y^7x^3 - 0.5x^6$							
MSE	**0.5**	0.1	0.05	0.01	0.001	0.0001	0.00001
BP:SU	1.0 ± 0	1.1 ± 0.13	1.1 ± 0.11	7.6 ± 10.92	–	–	–
BP:PU	1.0 ± 0	1.0 ± 0	1.0 ± 0	1.2 ± 0.33	–	–	–
PSO:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	1.2 ± 0.33	580.1 ± 102.99	–	–
PSO:PU	1.0 ± 0	1.5 ± 0.58	3.7 ± 1.59	25.2 ± 3.64	394 ± 121.32	783.3 ± 90.39	904.5 ± 81.34
GA:SU	1.1 ± 0.2	2.2 ± 0.6	$63. \pm 1.8$	54.8 ± 13.9	924.5 ± 78.60	–	–
GA:PU	3.5 ± 0.86	6.8 ± 1.01	7.8 ± 1.27	20.3 ± 2.11	279.1 ± 100.29	994.8 ± 8.88	–
LFOP:SU	1.8 ± 0.33	4.0 ± 0.43	4.9 ± 0.33	10.1 ± 1.27	473.2 ± 32.71	–	–
LFOP:PU	9.2 ± 1.84	18.7 ± 1.62	21.8 ± 0.95	37.5 ± 4.02	–	–	–

$f(x,y) = x^2 + y^2$							
MSE	**0.5**	**0.1**	**0.05**	0.01	**0.001**	**0.0001**	**0.00001**
BP:SU	1.0 ± 0	1.0 ± 0.00	1.0 ± 0.00	$16.23 \pm 2,68$	98.9 ± 40.68	–	–
BP:PU	–	–	–	–	–	–	–
PSO:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	56.1 ± 20.72	766.0 ± 154.22	–	–
PSO:PU	1.2 ± 0.24	3.0 ± 0.51	14.3 ± 2.28	543 ± 153.78	786.2 ± 110.16	821.0 ± 95.12	864.4 ± 86.07
GA:SU	1.0 ± 0	1.0 ± 0	1.1 ± 0.20	182.3 ± 67.47	–	–	–
GA:PU	4.1 ± 0.75	12.4 ± 1.19	19.2 ± 2.02	277.6 ± 77.15	480.8 ± 26.14	896.3 ± 59.61	–
LFOP:SU	30.9 ± 7.23	0.5 ± 0.35	1.6 ± 0.53	153.7 ± 55.90	–	–	–
LFOP:PU	1.7 ± 0.14	2.7 ± 0.53	95.3 ± 98.01	511.9 ± 124.63	850.6 ± 90.63	822.6 ± 95.57	818.1 ± 97.62

$f(x,y) = sin(x^2) + sin(y^2)$							
MSE	**0.5**	0.1	0.05	**0.01**	0.001	0.0001	0.00001
BP:SU	1.0 ± 0	1.0 ± 0.00	1.0 ± 0.00	195.8 ± 49.32	–		–
BP:PU	–	–	–	–	–	–	–
PSO:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	638.3 ± 177.60	–	–	–
PSO:PU	2.7 ± 1.18	17.4 ± 2.38	33.9 ± 6.61	171 ± 28.58	720.5 ± 118.47	927.2 ± 56.35	–
GA:SU	1.0 ± 0	1.0 ± 0	12.6 ± 3.52	870.6 ± 107.36	–	–	–
GA:PU	14.1 ± 1.03	28.2 ± 1.98	39 ± 5.01	180 ± 48.28	452 ± 33.52	–	–
LFOP:SU	1.0 ± 0	11.9 ± 6.09	24.8 ± 6.72	121.5 ± 4.90	981 ± 19.97	970.8 ± 55.30	–
LFOP:PU	–	–	–	–	–	–	–

Table 4.28: Epochs needed to reach MSE levels

$f(x,y) = (4 - 2.1x^2 + (\frac{x^3}{3}))x^2 + xy + (4y^2 - 4)y^2$							
MSE	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
BP:SU	1.0 ± 0	1.0 ± 0.07	1.1 ± 0.09	6.5 ± 1.52	75.5 ± 7.27	–	–
BP:PU	–	–	–	–	–	–	–
PSO:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	31.5 ± 5.37	–	–	–
PSO:PU	26.2 ± 1.68	66.9 ± 10.15	154.6 ± 61.23	–	–	–	–
GA:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0.07	214.1 ± 97.56	–	–	–
GA:PU	19.6 ± 1.92	77.7 ± 12.82	302.6 ± 96.03	–	–	–	–
LFOP:SU	1.0 ± 0	0.7 ± 0.47	0.7 ± 0.31	30.7 ± 3.18	89 ± 5.86	247.8 ± 28.7	–
LFOP:PU	–	–	–	–	–	–	–

$f(x,y) = sin(x) \cdot sin(y)\sqrt{x \cdot y}$							
MSE	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
BP:SU	1.0 ± 0	1.0 ± 0.00	1.0 ± 0.00	161.4 ± 27.37	–	–	–
BP:PU	–	–	–	–	–	–	–
PSO:SU	1.0 ± 0	1.0 ± 0	1.0 ± 0	–	–	–	–
PSO:PU	1.4 ± 0.26	4.0 ± 0.82	5.9 ± 0.79	16 ± 1.63	199.8 ± 88.82	761.3 ± 123.76	991.2 ± 17.18
GA:SU	1.0 ± 0	1.0 ± 0.07	1.0 ± 0	–	–	–	–
GA:PU	10 ± 1.15	16.7 ± 1.64	19 ± 1.56	35.1 ± 2.07	277.3 ± 103.85	–	–
LFOP:SU	1.0 ± 0	0.6 ± 0.38	3.2 ± 1.68	–	–	–	–
LFOP:PU	–	–	–	–	–	–	–

Table 4.29: Epochs needed to reach MSE levels

PSO								
		Generalization levels (MSE)						
Function	**Unit Type**	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
x^2	SU	100.0%	100.0%	100.0%	63.3%	46.7%	0.0%	0.0%
	PU	100.0%	100.0%	100.0%	96.7%	86.7%	13.3%	0.0%
$x^3 - 0.04x$	SU	100.0%	100.0%	100.0%	76.7%	53.3%	33.3%	0.0%
	PU	100.0%	96.7%	93.3%	90.0%	90.0%	90.0%	53.3%
Henon	SU	100.0%	100.0%	100.0%	93.3%	86.7%	0.0 %	0.0%
	PU	83.3%	80.0%	60.0%	53.3%	30.0 %	10.0%	3.3%
$y^7x^3 - 0.5x^6$	SU	100.0%	100.0%	100.0%	66.7 %	0.0%	0.0%	0.0%
	PU	100.0%	93.3%	93.3%	56.7%	20.0%	20.0 %	13.3%
$x^2 + y^2$	SU	100.0%	100.0%	100.0 %	33.3%	13.3%	0.0%	0.0%
	PU	100.0%	76.7%	70.0%	40.0%	30.0%	20.0%	33.3%
$sin(x^2) + sin(y^2)$	SU	100.0%	100.0%	100.0%	16.7%	0.0%	0.0%	0.0 %
	PU	96.7%	83.3%	66.7%	53.3%	16.7%	3.3%	0.0%
camel	SU	100.0%	100.0%	100.0%	16.7%	0.0%	0.0%	0.0%
	PU	70.0%	46.7%	10.0%	0.0%	0.0 %	0.0%	0.0%
graph	SU	100.0%	100.0 %	100.0%	0.0%	0.0%	0.0%	0.0%
	PU	96.7%	96.7%	90.0%	50.0%	6.7%	3.3%	3.3 %

Table 4.30: Percentage simulations that converged to MSE levels

GA								
Function	Unit	Generalization levels (MSE)						
	Type	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
x^2	SU	100.0%	100.0%	86.7%	63.3%	0.0%	0.0%	0.0%
	PU	100.00%	96.7%	93.3%	90.0%	66.7%	50.0%	40.0%
$x^3 - 0.04x$	SU	100.00%	100.0%	100.0%	80.0%	66.7%	13.3%	0.0
	PU	100.0%	96.7%	93.3%	83.3%	76.7%	70.0%	6.7%
Henon	SU	100.0%	100.0%	100.0%	93.3%	13.3%	0.0%	0.0%
	PU	93.3%	90.0%	83.3%	70.0 %	30.0%	0.0%	0.0%
$y^7x^3 - 0.5x^6$	SU	100.0%	100.0%	100.0%	100.0%	0.0 %	0.0%	0.0%
	PU	96.7%	80.0%	76.7%	50.0 %	43.3%	3.3%	0.0%
$x^2 + y^2$	SU	100.0%	100.0%	86.7%	20.0%	0.0%	0.0 %	0.0%
	PU	90.0%	83.3%	80.0%	60.0%	13.3%	6.7%	0.0%
$sin(x^2) + sin(y^2)$	SU	100.0%	96.7%	60.0%	3.3%	0.0%	0.0%	0.0%
	PU	90.0%	83.3%	80.0%	76.7%	23.3%	0.0 %	0.0%
camel	SU	100.0%	100.0%	100.0%	3.3%	0.0%	0.0%	0.0%
	PU	70.0%	36.7%	10.0%	3.3%	0.0%	0.0%	0.0%
graph	SU	100.0%	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%
	PU	63.3%	60.0%	56.7%	50.0%	50.0%	0.0%	0.0%

Table 4.31: Percentage simulations that converged to MSE levels

LFOP								
Function	Unit	Generalization levels (MSE)						
	Type	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
x^2	SU	100.0%	100.0%	90.0%	56.7%	33.3 %	0.0%	0.0%
	PU	62.9%	58.6%	48.6%	42.9%	41.4%	42.9%	47.1%
$x^3 - 0.04x$	SU	100.0%	100.0%	96.7%	93.3%	46.7%	26.7%	0.0%
	PU	94.3%	94.3%	93.3%	90.0%	84.3%	72.9 %	72.9%
Henon	SU	100.0%	100.0%	100.0%	76.7%	53.3%	13.3%	0.0%
	PU	75.7%	48.6%	44.3%	44.3%	37.1%	35.7%	1.4%
$y^7x^3 - 0.5x^6$	SU	100.0%	100.0%	100.0%	93.3%	63.3%	0.0 %	0.0%
	PU	91.4%	90.0%	78.6%	57.1%	0.0%	0.0%	0.0%
$x^2 + y^2$	SU	100.0%	90.0%	86.7%	6.7%	0.0%	0.0%	0.0%
	PU	98.6%	98.6%	87.1%	35.7%	12.9%	15.7 %	15.7%
$sin(x^2) + sin(y^2)$	SU	100.0%	90.0%	80.0%	13.3%	3.3%%	3.3%	0.0%
	PU	–	–	–	–	–	–	–
camel	SU	100.0%	96.7%	86.7%	16.7 %	10.0%	6.7%	0.0%
	PU	–	–	–	–	–	–	–
graph	SU	100.0%	90.0%	86.7%	15.7%	0.0%	0.0%	0.0%
	PU	–	–	–	–	–	–	–

Table 4.32: Percentage simulations that converged to MSE levels

BP								
Function	Unit	Generalization levels (MSE)						
	Type	0.5	0.1	0.05	0.01	0.001	0.0001	0.00001
x^2	SU	93.3%	86.7%	33.3%	10.0%	6.7%	0.0%	0.0%
	PU	100.0%	97.6%	97.6%	78.7 %	27.8%	13.8%	0.0%
$x^3 - 0.04x$	SU	90.0%	93.3%	83.3%	13.3%	10.0%	0.0%	0.0%
	PU	100.0%	100.0%	87.9%	67.2%	41.3%	17.0%	11.4%
Henon	SU	96.7%	96.7 %	93.3%	93.3%	63.3%	0.0%	0.0%
	PU	–	–	–	–	–	–	–
$y^7x^3 - 0.5x^6$	SU	100.0%	100.0%	100.0%	96.7%	6.7%	0.0%	0.0%
	PU	–	–	–	–	–	–	–
$x^2 + y^2$	SU	100.0%	100.0%	100.0%	83.3%	16.7%	0.0%	0.0%
	PU	–	–	–	–	–	–	–
$sin(x^2) + sin(y^2)$	SU	100.0%	100.0%	100.0%	0.0%	0.0%	0.0%	0.0%
	PU	–	–	–	–	–	–	–
camel	SU	100.0%	100.0%	100.0%	33.3%	10.0%	0.0%	0.0%
	PU	–	–	–	–	–	–	–
graph	SU	100.0%	100.0%	100.0 %	0.0%	0.0%	0.0%	0.0%
	PU	–	–	–	–	–	–	–

Table 4.33: Percentage simulations that converged to MSE levels