# Training High-Dimensional Neural Networks with Cooperative Particle Swarm Optimiser

Anna S Rakitianskaia and Andries P Engelbrecht

***Abstract*— This paper analyses the behaviour of particle swarm optimisation applied to training high-dimensional neural networks. Despite being an established neural network training algorithm, particle swarm optimisation falls short at training high-dimensional neural networks. Reasons for poor performance of PSO are investigated in this paper, and hidden unit saturation is hypothesised to be a cause of the failure of PSO in training high-dimensional neural networks. An analysis of various activation functions and search space boundaries leads to the conclusion that hidden unit saturation can be slowed down by combining activation function choice with appropriate search space boundaries. Bounded search is shown to significantly outperform unbounded search in high-dimensional neural network error search spaces.**

## I. Introduction

NEURAL NETWORKS (NNs) are powerful mathematical models inspired by the mammalian brain, capable of finding patterns in data. A NN with enough units in the hidden layer can represent any non-linear function [1]. Among many applications of NNs are pattern recognition, classification, and function approximation [2], [3], [4]. The pattern recognition ability gained NNs a significant place in real-world applications such as speech recognition and computer vision. What makes the last two applications especially difficult is the high dimensionality of the data.

A few approaches to NN training exist, of which particle swarm optimisation (PSO) was applied with significant success to a wide range of problems. However, most of the existing research in PSO NN training considered only relatively low-dimensional problems. Thus, despite the established potential of PSO as a NN training algorithm, no relevant scalability studies exist.

In this paper, the applicability of PSO to high-dimensional NN training is investigated. It is discovered that PSO performs better as a NN training algorithm when the search space is bounded, and exhibits divergent behaviour in unbounded search spaces. The significance of choosing an appropriate activation function also becomes evident in the study.

The rest of this paper is organized as follows: Section II discusses NNs and high-dimensional NN training problems. Section III outlines the PSO algorithm and the variations of PSO developed for high-dimensional problems. The empirical study conducted is presented in Section IV. Section V summarises the paper and lists the conclusions arrived at.

Anna Rakitianskaia and Andries Engelbrecht are with the Department of Computer Science, University of Pretoria, Pretoria, South Africa (email: myearwen@gmail.com, engel@cs.up.ac.za).

## II. Neural Networks

A neural network is a mathematical model inspired by the learning mechanisms of the human brain. A NN is essentially a collection of interconnected neurons (also referred to as units further in this study) aligned in layers, where every neural connection is assigned a weight. Every neuron receives inputs from the previous layer, multiplied by the connection weights, and outputs a signal by passing the net input signal through the activation function. It was theoretically proved in [1] that a NN with enough hidden units can represent any non-linear mapping between the input space and the target space.

The NN itself is just a structure capable of representing a non-linear function, requiring to be trained on a problem in order to learn the mapping between inputs and target outputs. NN training is an optimisation problem, where the objective is to find the optimal set of weights and biases such as that the NN error is minimised. Training can be supervised, unsupervised or reinforced. This paper deals with supervised NNs only. Such NNs work on a set of data patterns, where each pattern is a vector of problem inputs and corresponding targets. Given a set of data patterns with known targets, a NN is trained to learn the mapping between the inputs and the targets. A trained NN is then capable of accurately approximating outputs for the data patterns it has never seen before.

Neural networks are extensively applied to various classification, pattern recognition, and forecasting real-world problems in engineering, medical, financial and other fields [5], [6], [7]. What often makes real-life applications challenging is the inherent high dimensionality of NNs. The total number of weights in a NN increases non-linearly with a linear increase in the number of neurons, and complex problems require large numbers of hidden neurons to allow for necessary precision. Thus, a real-life NN training problem will be high-dimensional most of the time.

Particle swarm optimisation (PSO) was sucessfully used for optimising NN weights on a wide range of problems [8], [9], [10]. Details of the algorithm and its applicability to NN training are discussed in the next section.

## III. Particle Swarm Optimisation

The PSO algorithm is described in this section. Section III-A outlines the basic algorithm, Section III-B describes how PSO can be applied to train a NN, and Section III-C provides details on the existing high-dimensional PSO approaches.

### A. Basic algorithm

Particle Swarm Optimisation (PSO), first introduced by Kennedy and Eberhart in [11], is a nature-inspired population based optimisation technique. PSO operates on a set of particles, referred to as a swarm, where every particle represents a candidate solution to the optimisation problem. For an $n$-dimensional optimisation problem, a particle is represented by an $n$-dimensional vector, $\vec{x}$, also referred to as the particle's position. Every particle has a fitness value, indicating the quality of the candidate solution, and a velocity vector, $\vec{v}$, which determines the step size and direction of the particle's movement. Social interaction is imitated by forming neighbourhoods within a swarm. Each particle remembers its own best position found so far, and can also query the neighbourhood for the best position as discovered by the neighbouring particles. PSO searches for an optimum by moving the particles through the search space. At each time step, $t$, the position $\vec{x}_i(t)$ of particle $i$ is modified by adding the particle velocity $\vec{v}_i(t)$ to the previous position vector:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \tag{1}$$

Particle velocity determines the step size and direction of the particle. Velocity is updated using

$$\begin{aligned}\vec{v}_i(t) = \omega\vec{v}_i(t-1) &+ c_1\vec{r}_1(\vec{x}_{pbest,i}(t-1) - \vec{x}_i(t-1)) \\ &+ c_2\vec{r}_2(\vec{x}_{nbest,i}(t-1) - \vec{x}_i(t-1))\end{aligned} \tag{2}$$

where $\omega$ is the inertia weight [12], controlling the influence of previous velocity values on the new velocity; $c_1$ and $c_2$ are acceleration coefficients used to scale the influence of the *cognitive* (second term of Equation (2)) and *social* (third term of Equation (2)) components; $\vec{r}_1$ and $\vec{r}_2$ are vectors with each component sampled from a uniform distribution $U(0,1)$; $\vec{x}_{pbest,i}(t)$ is the personal best of particle $i$, or, in other words, the best position encountered by this particle so far; similarly, $\vec{x}_{nbest,i}(t)$ is the neighbourhood best of particle $i$, or the best position found by any of the particles in the neighbourhood of particle $i$. Thus, each particle is attracted to both the best position encountered by itself so far, as well as the overall best position found by the neighbourhood so far.

A particle's neighbourhood is determined topologically rather than spatially, meaning that the distance between particles is determined by particle's indices and not the actual position in the search space [11]. The GBest neighbourhood topology [11] was used in this study. In the GBest topology, the entire swarm constitutes the neighbourhood of a particle.

### B. PSO for NN training

PSO can easily be applied to NN training. Each particle is used to represent a candidate solution to the NN training problem, in other words, a vector of all the weights and biases of a NN. Fitness of a particle is calculated by substituting the particle position (i.e., NN weights) into the NN, and calculating the mean squared error (MSE) over the training set to obtain the training error ($E_T$), or over the test set to obtain the generalisation error ($E_G$). The PSO algorithm is then used to move particles through the weight space in order to minimise the MSE.

### C. PSO for high-dimensional problems

Many real-life optimisation problems deal with a large number of variables, and the applicability of PSO to large-scale optimisation has been investigated in the past. It was observed that the original PSO indeed suffers from the so-called "curse of dimensionality", the phenomenon of problem complexity increasing exponentially with a linear increase in the number of free parameters [13], [14]. Some of the notable PSO approaches to high-dimensional optimisation are discussed below.

*1) Cooperative PSO:* The cooperative PSO (CPSO) was first introduced by Van den Bergh and Engelbrecht in [13]. CPSO implements the *divide and conquer* principle by subdividing the search space dimension-wise into $K$ mutually-exclusive subsets, and assigning each subset to a separate subswarm. Each subswarm then optimises the problem over a limited set of dimensions. In order to get the complete solution vector, the best particles from all the subswarms, representing partial solutions, have to be combined. A *context vector* is used to evaluate the quality of partial solutions and to combine the partial solutions into a complete problem solution. Partial solution evaluation is accomplished by substituting values into the context vector only for the dimensions that a subswarm is responsible for, and keeping the rest of the context vector fixed at the best values as obtained from other subswarms. Each subswarm evaluates the context vector for each of its particles, and returns as the best solution the particle that, when substituted into the context vector, yielded the best fitness. Applying this procedure to every subswarm will result in the context vector containing the optimal solution found so far.

Pseudocode for CPSO is given below:

1) Initialise $K$ sub-swarms. Assuming problem dimension is $n$ and $n$ is divisible by $K$, each sub-swarm will be of dimension $\frac{n}{K}$. When $n$ is not divisible by $K$, the closest round-off approximation is obtained such that subswarms are not all of the same dimensionality.
2) For each sub-swarm:
    a) For each particle:
        i) Evaluate fitness using the context vector.
        ii) Adjust velocity and position.
        iii) Determine personal best.
    b) Determine global best.
3) Repeat step 2 until a stopping criterion is met.

*2) CPSO with random grouping:* Li and Yao [15] expanded on the original CPSO idea by introducing the concept of random grouping. Random grouping CPSO follows the logic of CPSO, the only difference being that at every iteration the problem is randomly subdivided into subproblems, instead of doing the subdivision once only in the beginning of the algorithm run. Li and Yao [15] argue that the advantage of this approach is the decoupling of correlated variables. When

no prior information about variable correlation is available, poor subdivision of dimensions might slow down algorithm convergence. However, when NN training is concerned, the variables being optimised are NN weights. Weights in the same layer connect to the same neurons and therefore exhibit a high degree of correlation. Thus, in the specific case of NN training, randomly grouping weights between different NN layers would not make the problem any easier, and might in fact slow the algorithm down.

*3) PSO with constrained boundaries:* Helwig and Wanka [16] suggested simplifying high-dimensional problems by introducing search space boundaries, and then controlling the behaviour of PSO by adjusting boundary constraint techniques. Indeed, if the area in which the optimum is expected to be located can be estimated, searching through the rest of the search space is nothing but a waste of computational power. NN weights are typically initialised in a small area around zero [2], thus if boundaries are applied, they should be symmetrical. The applicability of enforcing boundaries on PSO NN training is investigated in this paper. As suggested by Helwig and Wanka [16], a random boundary constraint technique is applied to keep particles within predefined bounds. If a particle exceeds the lower or upper limit of the $j$-th dimension, a random value sampled from a uniform distribution $U(lb, ub)$, where $lb$ is the lower bound value and $ub$ is the upper bound value, is assigned to the $j$-th component of the particle's position vector. Particle velocity is subsequently reset according to:

$$\vec{v}_{ij}(t) = \vec{x}_{ij}(t) - \vec{x}_{ij}(t-1) \tag{3}$$

## IV. Empirical Analysis

This section provides a description of the experimental procedure followed and experimental results obtained for this study. The goal of the experiments was to investigate the performance and behaviour of CPSO applied to high-dimensional NN training in bounded and unbounded search spaces. This study hypothesises that PSO performs poorly on large NNs due to hidden unit saturation. Saturation occurs when hidden units predominantly output values close to the asymptotic ends of the activation function in use. Saturation is undesirable, because if multiple training patterns cause hidden units to output the same values, differentiation between the patterns will become impossible. Firstly, it is shown in the experiments that hidden unit saturation is indeed present with CPSO NN training. Secondly, various activation functions and search space boundaries are tested as a means of controlling the saturation. The rest of this section presents the high-dimensional problem used to test the hypothesis, describes the experimental setup and the experimental results obtained, and discusses the conclusions arrived at.

### A. The MNIST Benchmark

The aim of this paper was to investigate the NN training performance of the PSO applied to large-scale NNs, therefore a benchmark out of the computer vision domain was selected for the experiments. Results on the MNIST (Modified National Institute of Standards and Technology) handwritten digit recognition problem were first published by LeCun in [17], and the original data set is available on-line at [18]. The MNIST problem has become a classic image recognition benchmark, still widely used today [19], [20]. The MNIST data set is a collection of grayscale images encoded in binary form, where every image is 28 by 28 pixels, and encodes one of the handwritten digits from 0 to 9. Every pixel encodes an integer value in $[0, 255]$. The data set features both inputs (images) and targets (corresponding labels from 0 to 9), thus supervised learning techniques can be easily applied. There are $70000$ images with corresponding labels in total, subdivided into a training set of $60000$ instances and a test set of $10000$ instances.

### B. Neural Network Setup

For all the experiments in this study, a three-layered NN was used, comprising of an input layer, a hidden layer, and an output layer. The dimensionality of a NN training problem is determined by the total number of weights and biases. For all the experiments conducted in this study, fully connected NNs were used. Thus, the total number of NN weights, $n_w$, taking bias units into account, was

$$n_w = (I+1)J + (J+1)K \tag{4}$$

where $I$ is the number of inputs, $J$ is the number of hidden units, and $K$ is the number of outputs.

For the unmodified MNIST data set, the total number of inputs would be $28 \times 28 = 784$. If $100$ units are used in the hidden layer, and $10$ output units, corresponding to the $10$ possible outcomes, are used in the output layer, the overall dimensionality of the NN, according to Equation (4), becomes $(784+1) \times 100 + (100+1) \times 10 = 79510$. It should be noted here that all results on large-scale PSO optimisation published to date report on problems of *up to* 2000 dimensions [15]. Clearly, the given NN training problem goes way beyond the current published studies. In order to reduce dimensionality and make the problem more approachable for the initial experiments, the input patterns were reduced from 28 by 28 pixels to 14 by 14 pixels by taking an average value of every 4 pixels in 2D space. The reduced number of inputs thus became $14 \times 14 = 196$, resulting in a NN of $(196+1)\times100+(100+1)\times10 = 20710$ dimensions. The resulting dimensionality is still roughly 10 times higher than the dimensionality of the large-scale PSO problems previously reported on. Thus, this paper tackled a problem of a higher dimensionality than PSO has ever been applied to in the past.

The training performance of the PSO was separately evaluated using three different activation functions in the hidden layer of the NN, namely, the sigmoid function, the hyperbolic tangent function, and the Elliott function, illustrated in Fig.1.

The sigmoid function is defined as

$$f_{NN}(net) = 1/(1+e^{-net}) \tag{5}$$

where $net$ is the sum of weighted incoming signals. The sigmoid function is the most commonly used activation

function. The output of the sigmoid function is in the range $(0, 1)$.

The hyperbolic tangent function, further referred to as TanH, is defined as

$$f_{NN}(net) = (e^{net} - e^{-net})/(e^{net} + e^{-net}) \quad (6)$$

The output of TanH is in the range $(-1, 1)$.

The third activation function used in this paper is the Elliott function [21], further referred to as Elliott and defined as

$$f_{NN}(net) = net/(1 + |net|) \quad (7)$$

The output range of Elliott is also $(-1, 1)$, but it has a shallower gradient than TanH and thus approaches the asymptotes slower.

In all the experiments, the input patterns were rescaled according to the output range of the activation function in use.

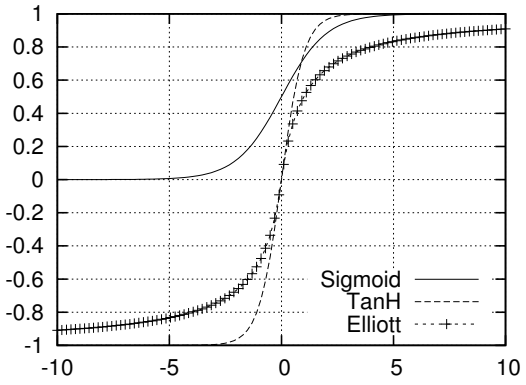



Fig. 1. Activation functions

### C. Particle Swarm Optimisation Setup

The experiments presented in this study were conducted with the cooperative PSO described in Section III-C.1, further referred to as CPSO for brevity. A CPSO consisting of 5 subswarms was used, where every subswarm consisted of 10 particles, giving a total number of particles equal to 50. Subdivision of weights into 5 subsets was done by applying the procedure described in Section III-C.1 to the weight vector of the NN. In all experiments, the inertia weight $\omega$ was set to 0.729844 while the values of the acceleration coefficients $c_1$ and $c_2$ were set to 1.496180. This choice is based on [22], where it was shown that such parameter settings give convergent behaviour. Velocity was not constrained, and the GBest neighbourhood topology was used for every subswarm.

CPSO with no boundary constraints as well as CPSO with the random boundary constraint as described in Section III-C.3 was used. Different initialisation ranges for particle positions (in other words, NN weights) and particle velocities were chosen according to the size of the interval to which the particles were confined by the imposed boundaries. A summary of the initialisation ranges is given in Table I. Unconstrained CPSO is further referred to as simply CPSO, while CPSO constrained to intervals $[-2, 2]$, $[-1, 1]$, and $[-0.5, 0.5]$ is referred to as $\text{CPSO}_2$, $\text{CPSO}_1$, and $\text{CPSO}_{0.5}$, respectively.

TABLE I
PARTICLE INITIALISATION INTERVALS

| Boundary Interval | Position | Velocity |
|---|---|---|
| No boundary | $[-1, 1]$ | $[-0.5, 0.5]$ |
| $[-2, 2]$ | $[-1, 1]$ | $[-0.5, 0.5]$ |
| $[-1, 1]$ | $[-0.5, 0.5]$ | $[-0.25, 0.25]$ |
| $[-0.5, 0.5]$ | $[-0.25, 0.25]$ | $[-0.125, 0.125]$ |

Swarm diversity $D_S$ was also measured in every simulation. The diversity measurement used is the average distance around the swarm center [23], given by

$$D_S = \frac{1}{S_P} \sum_{y=1}^{S_P} \sqrt{\sum_{l=1}^{n} (x_{yl} - \bar{x}_l)^2} \quad (8)$$

where $S_P$ is the swarm size, $n$ is the dimensionality of the problem space, $\vec{x}_y$ and $\vec{\bar{x}}$ are particle position $y$ and the swarm center, respectively. The choice of diversity measure is based on [24], where this measure was shown to be a valid diversity measure. Applied to cooperative PSO, the diversity is calculated by substituting every particle of each subswarm into the context vector, thus generating $S_P$ particles of dimension $n$, and calculating the diversity of the resulting collection of particles according to Equation (8).

All reported results are averages over 30 simulations. Each algorithm ran for 500 iterations.

### D. Empirical Results

Table II summarises the average $E_T$, $E_G$, and $D_S$ obtained in the experiments, minimum error values for every algorithm are displayed in bold, and Table III summarises the respective overall algorithm ranks. Algorithms were ranked based on their mean $E_T$ and $E_G$ values. Algorithm ranking involved the two-tailed non-parametric Mann-Whitney $U$ test [25], used to determine whether the difference in performance between any two algorithms was statistically significant. The choice of the significance test is based on [26], where the authors showed that the Mann-Whitney $U$ test is safer than the parametric tests such as the $t$-test, since the Mann-Whitney $U$ test assumes neither normal distributions of data, nor homogeneity of variance. The null hypothesis $H_0 : \mu_1 = \mu_2$, where $\mu_1$ and $\mu_2$ are the means of the two samples being compared, was evaluated at a significance level of 95%. The alternative hypothesis was defined as $H_1 : \mu_1 \neq \mu_2$.

It is evident from Tables II and III that smaller search space boundaries yielded significantly smaller errors. Table III also shows that the Elliott activation function performed significantly worse than other activation functions in bounded spaces, and that the Sigmoid significantly outperformed the other activation functions in the unbounded space. Next, the behaviour of PSO is analysed to explain the observed phenomena.

TABLE II
AVERAGE MSE

| Activation | | CPSO | $CPSO_2$ | $CPSO_1$ | $CPSO_{0.5}$ |
|---|---|---|---|---|---|
| Sigmoid | $E_T$ | $\mathbf{0.157141 \pm 0.202442}$ | $\mathbf{0.181511 \pm 0.116224}$ | $0.117036 \pm 0.034167$ | $\mathbf{0.055927 \pm 0.001802}$ |
| | $E_G$ | $\mathbf{0.157744 \pm 0.202563}$ | $\mathbf{0.18225 \pm 0.115888}$ | $0.117101 \pm 0.034353$ | $\mathbf{0.055874 \pm 0.001981}$ |
| | $D_S$ | $1249.55 \pm 1655.11$ | $7.41944 \pm 4.5701$ | $5.67141 \pm 2.33124$ | $2.99356 \pm 0.847664$ |
| TanH | $E_T$ | $0.842769 \pm 1.15126$ | $0.212523 \pm 0.074053$ | $\mathbf{0.104675 \pm 0.024872}$ | $0.085831 \pm 0.028571$ |
| | $E_G$ | $0.543946 \pm 0.667151$ | $0.213253 \pm 0.072649$ | $\mathbf{0.104977 \pm 0.025771}$ | $0.08595 \pm 0.028338$ |
| | $D_S$ | $139.408 \pm 42.9574$ | $5.87239 \pm 1.14979$ | $3.11595 \pm 0.44452$ | $1.9375 \pm 1.50791$ |
| Elliott | $E_T$ | $0.893402 \pm 0.716636$ | $0.827706 \pm 0.585609$ | $0.39684 \pm 0.105096$ | $0.254966 \pm 0.013528$ |
| | $E_G$ | $0.896516 \pm 0.719555$ | $0.830121 \pm 0.585241$ | $0.397196 \pm 0.105313$ | $0.255185 \pm 0.013892$ |
| | $D_S$ | $644.303 \pm 615.979$ | $8.66452 \pm 5.02397$ | $4.11974 \pm 2.50938$ | $2.83175 \pm 1.25224$ |

TABLE III
OVERALL RANKS

| Activation | | CPSO | $CPSO_2$ | $CPSO_1$ | $CPSO_{0.5}$ |
|---|---|---|---|---|---|
| Sigmoid | $E_T$ | 5 | 6.5 | 3.5 | 1 |
| | $E_G$ | 5 | 6.5 | 3.5 | 1 |
| TanH | $E_T$ | 11 | 6.5 | 3.5 | 2 |
| | $E_G$ | 10.5 | 6.5 | 3.5 | 2 |
| Elliott | $E_T$ | 11 | 11 | 9 | 8 |
| | $E_G$ | 10.5 | 10.5 | 10.5 | 8 |

First of all, the reader might have noticed that classification error is not reported in this study. The reason for omitting this performance measure is simple: CPSO struggled at training the 20710-dimensional NN to such an extent that the classification error remained fixed at 100%, decreasing down to 95% in some cases. The CPSO clearly struggled to tackle the given optimisation problem, and the main goal of this study was to clarify the reasons for such poor performance of the PSO.

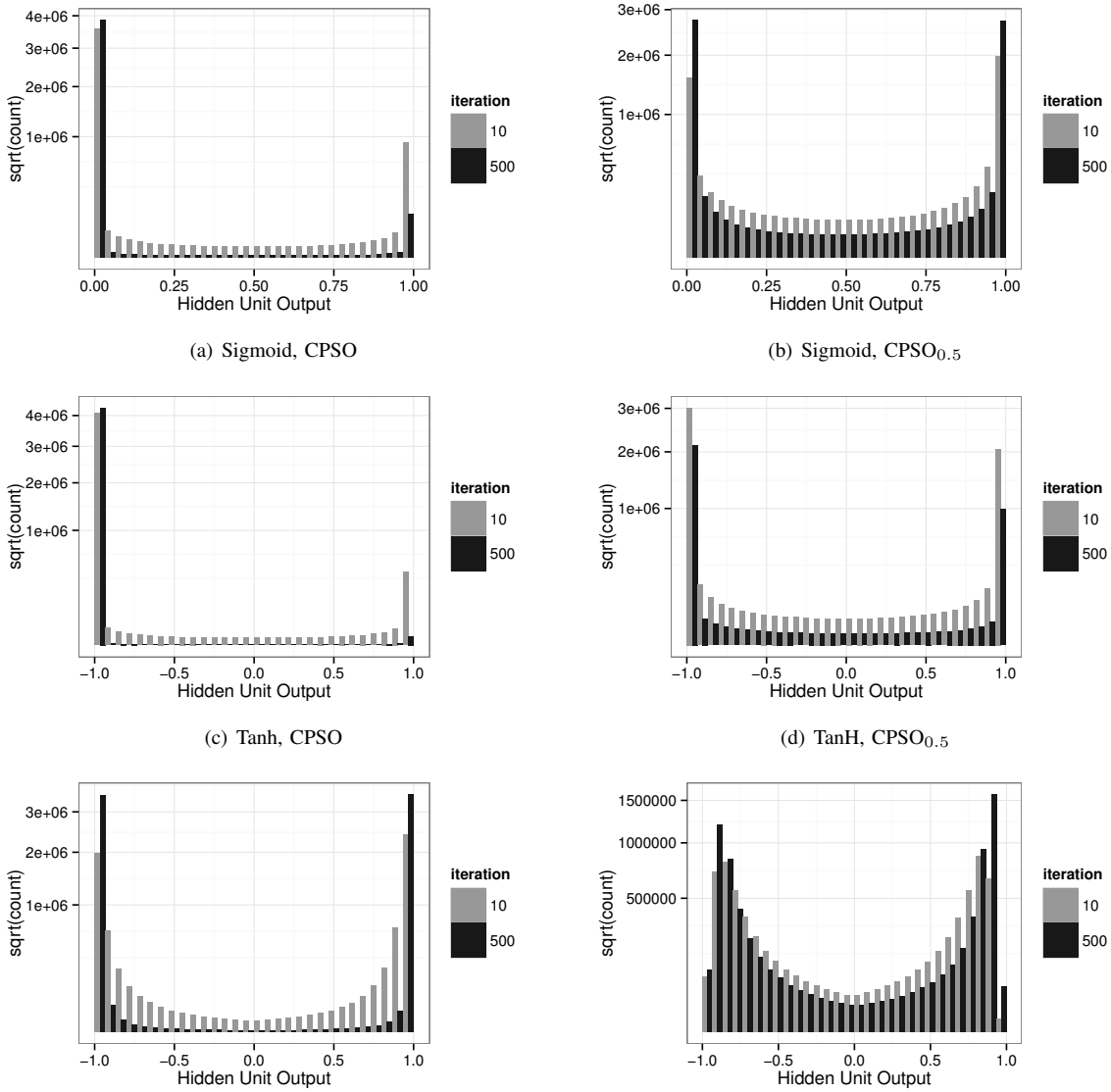
Initial experiments were conducted in an unbounded search space, since no rule exists specifying the possible limits for NN weights. Theoretically, NN weights can take on any floating-point values. Practically, NN weights are usually initialised in a small interval around zero to alleviate activation function saturation [2]. Saturation occurs when the hidden units of a NN predominantly output values close to the asymptotic ends of the activation function range. Derivatives of the sigmoid-like activation functions are close to zero near the asymptotic ends, which hinders traditional gradient descent NN training methods, making training very slow. Although PSO makes no use of activation function derivatives, saturation remains an issue, especially taking the NN size into account: a large fully connected NN architecture implies many incoming connections to every hidden unit, therefore a sum over a large number of weighted inputs. If the weights cause the resulting net input signal to always be a large positive or negative number, the hidden unit will always output a value close to either end of the activation function range. Reducing hidden units to this binary output state damages the overall information capacity of the NN, causing learning to be slow and inefficient.

Constraining the CPSO to a small interval around zero is hypothesised to decrease the hidden unit saturation by producing a smaller net input signal. Every particle represents a vector of NN weights and biases. Thus, limiting the particle's search space effectively limits the weight values to the respective interval. Initialising NN weights to a small interval was shown to be good practice [2], and decreasing weights over training epochs tends to improve the performance of gradient descent approaches to NN training [2]. Thus, limiting weights to a predefined small interval is hypothesised to improve the NN training performance of the PSO.

This study hypothesises that the hidden unit saturation is one of the reasons behind poor performance of the PSO on large-scale NN training problems. Figure 2 shows frequency distributions obtained for the hidden unit outputs for the three activations functions considered, in unbounded (CPSO) and bounded ($CPSO_{0.5}$) search spaces. Light grey corresponds to frequencies after 10 iterations of the algorithm, and dark grey corresponds to frequencies after 500 iterations. For display purposes, frequencies were square root scaled. It is evident from Figure 2 that hidden unit saturation was indeed present in both the unbounded and bounded spaces, and progressed with algorithm iterations. Comparing the frequency distributions obtained for the CPSO and the $CPSO_{0.5}$, it can, however, be concluded that the PSO exhibited stronger saturation in the unbounded search space than in the bounded search space. Tables II and III show that the overall MSE was also significantly lower in the bounded search spaces, thus confirming the hypothesis that limiting the search space improves the NN training performance of the PSO, and that hidden unit saturation indeed hinders the PSO NN training performance.

However, decreased saturation is not the only reason why bounding the search space improves the training performance of the PSO. Table II shows that the diversity values of the PSO in the unbounded search space are much higher than the respective values in the bounded search spaces. Indeed, investigating the diversity profile of the CPSO under different search space boundaries as shown in Figures 3 and 4 leads to the conclusion that the CPSO exhibited divergent behaviour in the unbounded search space, regardless of the activation function, and converged in the bounded search spaces. Enforcing boundaries on the search space significantly simplifies the optimisation problem, making it more feasible for the population-based approach.

Figure 2 shows that the Elliott activation function saturated




(a) Sigmoid, CPSO

(b) Sigmoid, $\text{CPSO}_{0.5}$

(c) Tanh, CPSO

(d) TanH, $\text{CPSO}_{0.5}$

(e) Elliott, CPSO

(f) Elliott, $\text{CPSO}_{0.5}$

Fig. 2. Hidden unit output values frequency distributions

less than the other two activation functions in the bounded search space. However, despite exhibiting less saturation, Elliott performed significantly worse than both Sigmoid and TanH in terms of the MSE, as seen in Table III. Such behaviour is explained by the fact that Elliott approaches the asymptotes much slower than Sigmoid and TanH, as shown in Figure 1. Therefore, output signals produced by Elliott units did not reach the actual target values (scaled to $[-0.9, 0.9]$), resulting in higher MSE. Actual error values, however, are relative measures of algorithm success, and the dynamics of an algorithm are to be examined to gain insight into the algorithm behaviour. Figure 5 shows the error profiles obtained by the CPSO under various boundary constraints with the three activation functions considered. It can be concluded from Figure 5 that Elliott indeed closely resembled the behaviour of Sigmoid, producing error curves of very similar shape. This indicates the potential competitiveness of Elliott. Scaling input pattern targets to a narrower range might improve the MSE performance of Elliott, and allow this activation function to make better use of its relative saturation resilience.

Although Table III shows that no statistically significant difference was observed between the performance of TanH and Sigmoid in search space bounded to $[-2, 2]$ and $[-1, 1]$, Figure 5 illustrates that these two activation functions yielded different error profiles. Indeed, Figures 5(b), 5(c), and 5(d) show that TanH consistently produced a steeper decreasing error profile than Sigmoid. It is especially evident in Figure 5(c) that, where Sigmoid was stagnating, TanH was not. Therefore, if the experiments were run for a larger number of iterations, TanH would have had a good chance to significantly outperform Sigmoid on the given NN training

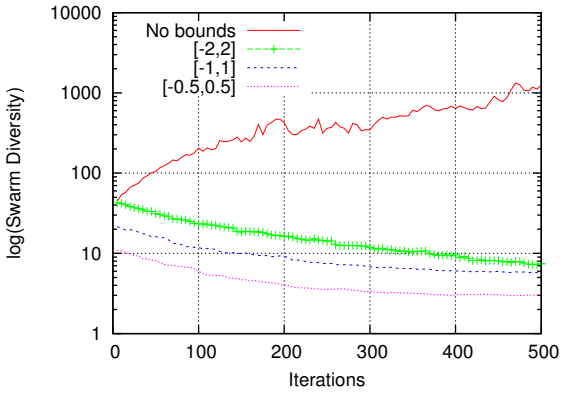



Fig. 3. Swarm diversity profile for Sigmoid

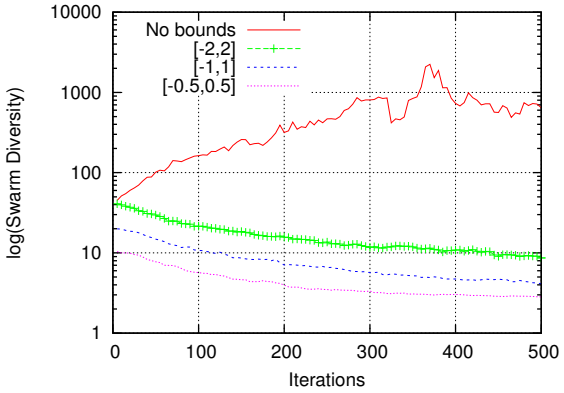



Fig. 4. Swarm diversity profile for Elliott

problem. Table II also shows that the use of TanH yielded smaller swarm diversity than the other two activation functions, thus TanH promoted swarm convergence better than either Sigmoid or Elliott. As described in Section IV-B, TanH has a wider output range than Sigmoid. This property of TanH increases the overall information capacity of the NN, making TanH a potentially better choice than Sigmoid.

## V. Conclusions

This study investigated the performance of the CPSO for training large-scale NNs. The MNIST data set was used as a benchmark, and it was discovered that the CPSO indeed struggles to solve the NN training problem of the given dimensionality. This study hypothesised that activation function saturation is one reason behind the poor performance of the PSO. Three different activation functions were considered in the experiments. It was also hypothesised that limiting the search space to a small interval around zero alleviates saturation, thus improving the performance of the PSO. Results obtained in the unbounded search space were compared to the results obtained in the search space limited to $[-2, 2]$, $[-1, 1]$, and $[-0.5, 0.5]$. Experimental results showed that saturation is indeed present, and that limiting the search space not only alleviates the observed saturation, but also yields convergent swarm behaviour. Out of the three activation functions considered, the Elliott function exhibited the least saturation, and the TanH function exhibited the least signs of stagnation. TanH also promoted convergent behaviour.

Bounding the search space slows down saturation, but does not stop it. Future work will include further investigation of the activation function saturation in the context of PSO NN training. Regularisation of NN weights applied to PSO training will be considered. Tuning additional PSO parameters such as maximum velocity, neighbourhood topology, or constriction factor in order to decrease the resulting saturation is also an interesting topic. The effect of constraining the search space for simpler PSO NN training problems will also be studied in future, and the effect of dynamically adjusting boundary size will be looked into. Another interesting thing to investigate is the effect of scaling input patterns to different ranges on the training performance of the PSO.

## VI. Acknowledgments

The authors would like to thank Aleksey Bondyakov (Joint Institute for Nuclear Research, Dubna, Russia) for technical support of the experiments conducted for this study. All reported experiments were run on the Data center (AZGRID) of the Institute of Physics of ANAS (Azerbaijan National Academy of Sciences). The authors are very thankful to ANAS for the provided resources, without which this study would not have been possible.

## References

[1] S. Lawrence, A. C. Tsoi, and A. D. Back, "Function approximation with neural networks and local methods: bias, variance and smoothness," in *Proceedings of the Australian Conference on Neural Networks*. Canberra, Australia: Australian National University, 1996, pp. 16–21.

[2] G. Dreyfus, *Neural networks: methodology and applications*. Berlin, Germany: Springer, 2005.

[3] D. W. Patterson, *Artificial Neural Networks: Theory and Applications*, 1st ed. Upper Saddle River, New Jersey, USA: Prentice Hall, 1998.

[4] R. Rojas, *Neural networks: a systematic introduction*. Berlin, Germany: Springer-Verlag, 1996.

[5] G. Falavigna, "Financial ratings with scarce information: A neural network approach," *Expert Systems with Applications*, vol. 39, no. 2, pp. 1784–1792, 2012.

[6] K. Palanikumar, B. Latha, V. Senthilkumar, and J. P. Davim, "Application of artificial neural network for the prediction of surface roughness in drilling GFRP composites," in *Materials Science Forum*, vol. 766. Trans Tech Publ., 2013, pp. 21–36.

[7] A. Szarek, M. Korytkowski, L. Rutkowski, R. Scherer, and J. Szyprowski, "Application of neural networks in assessing changes around implant after total hip arthroplasty," in *Artificial Intelligence and Soft Computing*. Springer, 2012, pp. 335–340.

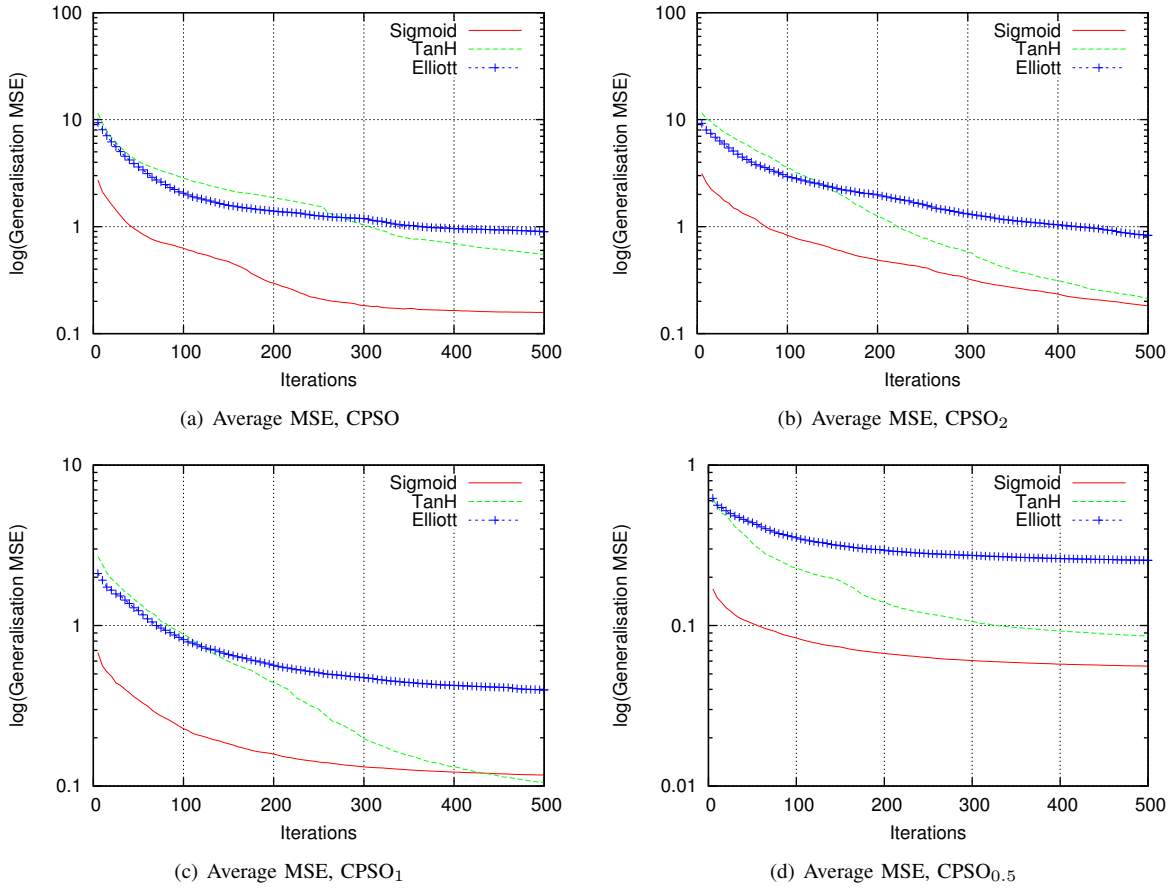
[8] M. T. Das and L. C. Dulger, "Signature verification (SV) toolbox: Application of PSO-NN," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 4, pp. 688–694, 2009.

[9] C. Jesuthanam, S. Kumanan, and P. Asokan, "Surface roughness prediction using hybrid neural networks," *Machining science and technology*, vol. 11, no. 2, pp. 271–286, 2007.

[10] R. Mendes, P. Cortez, M. Rocha, and J. Neves, "Particle swarms for feedforward neural network training," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2. Piscataway, USA: IEEE, 2002, pp. 1895–1899.

[11] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. IV. Piscataway, USA: IEEE, 1995, pp. 1942–1948.

[12] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimisation," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 3. Piscataway, USA: IEEE, 1999, pp. 1945–1950.




(a) Average MSE, CPSO

(b) Average MSE, $\text{CPSO}_2$

(c) Average MSE, $\text{CPSO}_1$

(d) Average MSE, $\text{CPSO}_{0.5}$

Fig. 5. Average $E_G$ under different boundary constraints


[13] F. Van Den Bergh and A. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[14] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Congress on Evolutionary Computation*, vol. 2. IEEE, 2004, pp. 1980–1987.

[15] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, 2012.

[16] S. Helwig and R. Wanka, "Particle swarm optimization in high-dimensional bounded search spaces," in *Proceedings of the Swarm Intelligence Symposium*. IEEE, 2007, pp. 198–205.

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

[18] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist/*, 1998.

[19] A. F. Krause, K. Essig, M. Piefke, and T. Schack, "No-Prop-fast-A high-speed multilayer neural network learning algorithm: MNIST benchmark and eye-tracking data classification," in *Engineering Applications of Neural Networks*. Springer, 2013, pp. 446–455.

[20] A. C. Pocock, "Feature selection via joint likelihood," Ph.D. dissertation, the University of Manchester, 2012.

[21] D. L. Elliott, "A better activation function for artificial neural networks," Institute for Systems Research, University of Maryland, Tech. Rep. TR 93-8, 1993.

[22] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1. Piscataway, USA: IEEE, 2000, pp. 84–88.

[23] T. Krink, J. S. Vesterstrom, and J. Riget, "Particle swarm optimisation with spatial particle extension," in *Proceedings of the Congress on Evolutionary Computation*, vol. 2, 2002, pp. 1474–1479.

[24] O. Olorunda and A. P. Engelbrecht, "Measuring exploration/exploitation in particle swarm using swarm diversity," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Hong Kong, 2008, pp. 1128–1134.

[25] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[26] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.