Weight Regularisation in Particle Swarm Optimisation Neural Network Training

Anna Rakitianskaia
Department of Computer Science
University of Pretoria
Pretoria, South Africa
Email: myearwen@gmail.com

Andries Engelbrecht
Department of Computer Science
University of Pretoria
Pretoria, South Africa
Email: engel@cs.up.ac.za

***Abstract*—Applying weight regularisation to gradient-descent based neural network training methods such as backpropagation was shown to improve the generalisation performance of a neural network. However, the existing applications of weight regularisation to particle swarm optimisation are very limited, despite being promising. This paper proposes adding a regularisation penalty term to the objective function of the particle swarm. The impact of different penalty terms on the resulting neural network performance as trained by both backpropagation and particle swarm optimisation is analysed. Swarm behaviour under weight regularisation is studied, showing that weight regularisation results in smaller neural network architectures and more convergent swarms.**

I. Introduction

A neural network (NN) is a structure of interconnected neurons, where every neuron is a simple non-linear function, and every neuron-to-neuron connection bears a weight. A NN is capable of discovering patterns in data by means of NN training. The training is achieved by feeding the data through the NN, and adjusting the NN connection weights such that the error is minimised.

Two "schools" of NN training exist: gradient descent-based training and population-based training. Gradient descent-based approaches minimize the NN error by adjusting the NN weights such that the negative gradient of the error function is followed. One of the best-known gradient descent-based training methods is the backpropagation algorithm [1]. Population-based approaches work with a pool of candidate solutions, in other words, different NN weight vectors. The candidate solutions interact with each other and move towards the area of the weight space where a smaller NN error is achieved. A notable population-based approach to NN training is particle swarm optimisation (PSO) [2], where interaction between candidate solutions models the social behaviour of a bird flock.

One of the crucial characteristics of a trained NN is its ability to generalise, in other words, to correctly predict the output of an input pattern that was not used for training. A NN that cannot generalise has no practical use. It was shown both theoretically and empirically that the generalisation ability of a NN can be improved by applying weight regularisation. Weight regularisation in a neural network penalises network complexity by adding a penalty term to the objective function. Network complexity is penalised by decreasing the rate of weight growth, as well as by driving irrelevant weights to zero. Weight regularisation has been successfully applied to gradient descent-based methods before, but the applicability of the same technique to population-based approaches is less explored [3]. The aim of this paper is to investigate the impact of weight regularisation on the performance of PSO NN training.

The rest of this paper is organized as follows: Section II discusses NNs. Section III elaborates on the benefits of weight regularisation. Section IV outlines backpropagation. Section V describes the PSO algorithm, the existing approaches to incorporate weight regularisation in the PSO, and the suggested approach. The empirical study conducted is presented in Section VI. Section VII summarises the paper and lists the conclusions.

II. Neural Networks

A neural network is a mathematical model inspired by the learning mechanisms of the brain, capable of approximating any non-linear function [4]. A NN is a collection of interconnected neurons 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.

The NN has to be trained on a problem in order to learn the required function. NN training is an optimisation problem, where the objective is to find the optimal set of weights and biases such that the NN error is minimised. Supervised training presents a NN with a set of data patterns, where each pattern is a vector of inputs and 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 approximating outputs for the data patterns it has never seen before. This ability is also known as the generalisation ability of a NN.

Neural networks are applied to various classification, pattern recognition, and forecasting real-world problems in engineering, medical, financial and other fields [5], [6]. To be practical in a real-life scenario, a NN must be able to generalise well. The generalisation ability of a NN is sensitive to the architecture of a NN, in other words, to the total number of neurons and weights. A NN with too few connections is not capable of representing a complex input-target relationship. A NN with too many connections, on the other hand, would learn irrelevant information such as noise and the order of

input patterns, thus hindering generalisation. The phenomena of learning irrelevant information is known as overfitting. The next section discusses weight regularisation, a technique developed to penalise irrelevant NN weights in order to decrease overfitting.

III. Weight Regularisation

The most common function used in NN training is the sum squared error (SSE):

$$E_{sse} = \sum_{p=1}^{P} \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2 \tag{1}$$

where P is the total number of training patterns, K is the total number of output units, $t_{k,p}$ is the k'th target of pattern p, and $o_{k,p}$ is the k'th output obtained for pattern p. Minimising the SSE minimises the error produced by the NN.

The goal of weight regularisation is to minimise both the NN error and its complexity. In order to do this, a penalty term is added to the objective function:

$$E_{nn} = E_{sse} + \lambda E_p \tag{2}$$

where λ is a hyperparameter controlling the strength of regularisation, and E_p is a penalty function that penalises NNs of high complexity. The complexity of a NN is determined by the overall number of neurons and the corresponding number of weights. Too few weights result in a poor approximation. Too many weights, however, promote overfitting by learning irrelevant information and noise.

A few penalty functions that optimize the overall number of NN weights have been developed by NN researchers. One of the most popular penalty functions suggested in the literature [7] is the weight decay, given by

$$E_p = \frac{1}{2} \sum_{l=1}^{W} w_l^2 \tag{3}$$

where W is the total number of weights in the NN, and w_l is the l'th weight. The weight decay penalty function simply calculates the overall magnitude of the weight vector. Minimising this function promotes smaller NN weights. It was shown in [8] that limiting the weight growth improves NN generalisation, because the relevant weights are reinforced by the training algorithm at every iteration, and the irrelevant ones decrease towards zero over time.

Another popular penalty function is the weight elimination function [9], given by

$$E_p = \sum_{l=1}^{W} \frac{w_l^2/w_0^2}{1 + w_l^2/w_0^2} \tag{4}$$

where w_0 is a user-defined constant. Weight elimination effectively counts the number of weights, and minimizing this function reduces the number of weights in a NN by driving irrelevant weights towards zero. Optimizing the value of w_0 is very important to the success of this penalty function, since larger w_0 promotes smaller weights, and smaller w_0 promotes larger weights. The preference of one over the other is problem-dependent.

IV. Backpropagation

Backpropagation is perhaps the most well-known NN training algorithm, invented in 1974 by Werbos [1]. Backpropagation uses gradient descent to adjust weight values. The gradient of the objective function is calculated, and NN weights are iteratively adjusted to follow the negative gradient. To apply weight regularisation to backpropagation, the gradient calculations are modified by adding the desired penalty term to the objective function. Like any gradient descent method, backpropagation uses the hill-climbing approach to function minimisation, which makes backpropagation prone to premature convergence to a local minimum. As a solution to the problem of premature convergence, alternative training algorithms such as PSO were suggested [10].

V. Particle Swarm Optimisation

The PSO algorithm is described in this section. Section V-A outlines the basic algorithm, Section V-B discusses PSO variations used in this study, Section V-C describes how PSO is applied to NN training, and Section V-D discusses weight regularisation in the context of PSO.

A. Basic algorithm

Particle swarm optimisation (PSO), first introduced by Kennedy and Eberhart in [2], 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:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \tag{5}$$

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

$$\begin{aligned} \vec{v}_i(t) = \omega \vec{v}_i(t-1) &+ c_1 \vec{r}_1(t)(\vec{x}_{pbest,i}(t-1) - \vec{x}_i(t-1)) \\ &+ c_2 \vec{r}_2(t)(\vec{x}_{nbest,i}(t-1) - \vec{x}_i(t-1)) \end{aligned} \tag{6}$$

where ω is the inertia weight [11], 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 (6)) and *social* (third term of Equation (6)) components; $\vec{r}_1(t)$ and $\vec{r}_2(t)$ 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, 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.

The global best (GBest) neighbourhood topology [2] was used in this study. In the GBest topology, the entire swarm constitutes the neighbourhood of a particle. The choice of topology is based on [12], where it was shown that the performance of GBest is not inferior to other PSO topologies.

B. PSO variations

The basic PSO outlined in the previous section has significantly developed over the past decade, numerous changes and improvements to the original algorithm were suggested. In this study, the cooperative PSO, described below in Section V-B1, was used alongside the basic PSO due to its known superior performance on NN training [10]. This study also introduces normalised velocity clamping, described in Section V-B2.

1) Cooperative PSO: The cooperative PSO (CPSO) was first introduced by Van den Bergh and Engelbrecht in [10], where it was shown that CPSO performed better than the original PSO on NN training. CPSO implements the *divide and conquer* principle by subdividing the search space dimension-wise into K mutually-exclusive subsets, where the optimal value of K is problem-dependent, 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. Application of this procedure to every subswarm results 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 the context vector fitness.
        ii) Adjust velocity and position.
        iii) Determine personal best.
    b) Determine global best.
3) Repeat step 2 until a stopping criterion is met.

2) Normalised velocity clamping: A maximum velocity $\vec{V}_{max}$ [11] is sometimes used to limit (or clamp) particle velocity in every dimension. Velocity clamping is done to prevent particles from traversing the search space too fast, since unreasonably large steps prevent particles from exploiting good regions. It was also observed in [13] that PSO tends to diverge on NN training problems unless swarm expansion is restricted. With velocity clamping, $\vec{V}_{max}$ is enforced by restricting $\vec{v}_i(t)$ per dimension j as follows:

$$v_{ij}(t) = \begin{cases} V_{max,j} & \text{if } v_{ij}(t) > V_{max,j} \\ -V_{max,j} & \text{if } v_{ij}(t) < -V_{max,j} \\ v_{ij}(t) & \text{otherwise} \end{cases} \quad (7)$$

There is, however, a complication with this widespread intuitive approach: When a particle's velocity $\vec{v}_i(t)$ is clamped according to Equation (7), $v_{ij}(t)$ may exceed $V_{max,j}$ for a dimension j, and not exceed in others. Thus, only a selection of dimensions of $\vec{v}_i(t)$ will be clamped, causing $\vec{v}_i(t)$ to change direction. The direction of $\vec{v}_i(t)$, however, is rather important to the algorithm's success, since particle velocities are the "memory" of the swarm, driving particles towards fruitful regions of the search space.

An alternative method of velocity clamping is suggested and used in this study, called normalised velocity clamping. The goal of normalised velocity clamping is to prevent velocity explosion, as well as preserve the direction of $\vec{v}_i(t)$ at all times. Normalised velocity clamping enforces $\vec{V}_{max}$ by normalising (i.e., dividing by vector length) $\vec{v}_i(t)$ whenever $V_{max,j}$ is exceeded by any $v_{ij}(t)$:

$$\vec{v}_i(t) = \begin{cases} \vec{v}_i(t)/\|\vec{v}_i(t)\| & \text{if } |v_{ij}(t)| > V_{max,j} \text{ for any } j \\ \vec{v}_i(t) & \text{otherwise} \end{cases} \quad (8)$$

Thus, a threshold $\vec{V}_{max}$ is set, up to which the particle velocity is allowed to grow. When the threshold is exceeded in any of the dimensions, $\vec{v}_i(t)$ is scaled down to a unit vector with the magnitude of 1. Velocity direction is thus preserved.

C. PSO for NN training

PSO can easily be applied to NN training [10]. 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 into the NN, and calculating the SSE 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 SSE.

D. Weight regularisation and PSO

Carvalho and Ludermir [3] have implemented weight decay by adding a penalty term to the position update Equation (5) of the standard PSO:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) - \lambda_i(t-1)\vec{x}_i(t-1) \quad (9)$$

where λ_i is a penalty coefficient. Every particle i maintains its own λ_i. At the beginning of the algorithm run, λ_i is initialised as $\lambda_i(0) = 5 \times 10^{-5}$. At every iteration the value of λ_i changes as follows:

$$\lambda_i(t+1) = \begin{cases} \lambda_i(t) - 1 \times 10^3 & \text{when } E_{sse} \text{ increases} \\ \lambda_i(t) + 1 \times 10^3 & \text{when } E_{sse} \text{ decreases} \end{cases} \quad (10)$$

The proposed weight decay method, further referred to as PSO-WD, was shown to improve PSO performance [3]. Although weight decay is a legitimate weight regularisaton technique, it was also observed that weight decay penalises large weights at the same rate as the small weights, undesirably interfering with the NN learning process. Thus, other regularisation methods than weight decay, such as weight elimination, might improve the performance of PSO NN training even further.

This paper proposes a generic way of applying a penalty function of choice to PSO NN training. Instead of applying a regularisation term to every particle position update, a regularisation term is added to the objective function of the algorithm, as given in Equation (2). Thus, the value of λ is shared by the swarm, and any penalty function can be used for E_p. This paper analyses the performance and behaviour of PSO with the added penalty term. Weight decay and weight elimination are considered. Because the proposed method puts no restrictions on the algorithm itself, cooperative versions of PSO are also considered under both weight decay and weight elimination.

VI. Empirical Study

This section describes the experimental procedure followed and discusses the experimental results obtained for this study. The goal of the experiments was to investigate the performance and behaviour of PSO with weight regularisation, and to show that the addition of a regularisation term improves generalisation performance and reduces NN architectures by driving irrelevant weights to zero. The penalty functions considered in this work were weight decay and weight elimination. The rest of this section lists the benchmark problems used to test the hypothesis, describes the experimental setup and the experimental results obtained, and discusses the conclusions arrived at.

A. Benchmark problems

For the purpose of investigating the effect of weight regularisation on PSO performance, seven popular benchmark problems were chosen. Problems with known optimal number of hidden units were chosen to reduce the initial number of irrelevant weights. Out of the seven problems selected, the first five were classification problems and the last two were time-series prediction problems. The benchmark problems along with the corresponding architectures are summarised in Table I. The specified sources point to papers from which the NN architectures were adopted.

B. Neural network setup

Feed-forward neural networks with a single hidden layer were used in the experiments. The identity (linear) activation function was used in the input layer, while the hidden layer and the output layer both used the sigmoid activation function, defined by $f_{NN}(net) = 1/(1+e^{-net})$ where net is the sum of weighted incoming signals. For all the experiments, the inputs were scaled to $[-1, 1]$, and the outputs were scaled to $[0.1, 0.9]$ to correspond with the active domain and range of the sigmoid function.

TABLE I. Benchmark Problems

Problem	# Input	# Hidden	Source
Iris	4	4	Gupta and Lam [14]
Glass Identification	9	9	Gupta and Lam [14]
Cancer	9	6	Carvalho and Ludermir [3]
Heart	35	6	Carvalho and Ludermir [3]
Diabetes	8	6	Carvalho and Ludermir [3]
Mackey-Glass	4	25	Larsen *et al* [15]
Sunspots	4	3	Weigend *et al* [9]

C. Training algorithms setup

For the purpose of investigating the effect of weight regularisation on PSO NN training, seven PSO setups were investigated: PSO with no weight regularisation, PSO-WD as described in [3], PSO with weight decay as defined by Equation (3), further referred to as PSOD, PSO with weight elimination as defined by Equation (4), further referred to as PSOE, and the cooperative versions of PSO, PSOD, and PSOE, referred to as CPSO, CPSOD, and CPSOE, respectively. Performance of the different PSOs was compared to backpropagation with no regularisation (BP), backpropagation with weight decay (BPD), and backpropagation with weight elimination (BPE). For the sake of a fair comparison, algorithm parameters were iteratively optimised on the training data set for each problem. Parameters used in the experiments are listed in Table II. The total number of particles in a PSO was set to 30 for the purpose of this study. Even though swarm size is also an optimizable parameter, fixing the number of particles ensures that no PSO variant has a computational advantage. For CPSO, 30 particles were divided into K equally-sized subswarms (either 6 subswarms of 5 particles each, or 2 subswarms of 15 particles each).

All reported results are averages over 30 simulations. Every simulation ran for 1000 iterations. Every data set was divided into the training set and the test set. 80% of data patterns constituted the training set, and the remaining 20% were used for testing. Test data used to calculate the final generalisation error values was not used for parameter optimisation.

D. Empirical results

Table III summarises the average E_T and E_G obtained for the problems considered. For the classification problems, classification error (E_C) is also reported in Table III. Minimum error values are shown in bold. In addition to E_T and E_G, the generalization factor, ρ_F, suggested in [16] and defined as $\rho_F = E_G/E_T$, is reported in Table III. The generalisation factor is an indication of overfitting: $\rho_F < 1$ indicates that $E_G < E_T$ (no overfitting), and $\rho_F > 1$ indicates that $E_G > E_T$ (overfitting is present).

Table III indicates that the addition of a penalty term to the objective function has a visible effect on the PSO performance. For most problems considered, PSO with regularisation produced a higher E_T and a lower E_G than PSO with no regularisation. In other words, the first glance at the data in Table III confirms the hypothesis that regularisation reduces overfitting of the PSO. Further evidence is provided by the values of ρ_F, that are often lower for at least one of the regularised PSOs as compared to no regularisation.

To further investigate the effect of regularisation on the behaviour of the PSO, a frequency distribution of NN weights for the Iris data set as obtained at the last algorithm iteration

TABLE II. OPTIMISED PARAMETER VALUES

Algorithm		Iris	Glass	Cancer	Heart	Diabetes	Mackey-Glass	Sunspots
PSO, PSOD, PSOE	w_{int}	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.4, 0.4]$	$[-0.3, 0.3]$	$[-0.8, 0.8]$	$[-0.9, 0.9]$
	V_{max}	1	2	1	5	1	5	no V_{max}
	c_1	1.3	1.5	1.5	1.7	1.5	1.7	1.5
	c_2	0.9	1.5	0.5	1.5	1.5	1.5	0.7
	ω	0.8	0.7	0.8	0.8	0.8	0.7	0.7
CPSO, CPSOD, CPSOE	w_{int}	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.4, 0.4]$	$[-0.3, 0.3]$	$[-0.8, 0.8]$	$[-0.9, 0.9]$
	V_{max}	1	2	1	5	1	5	1
	c_1	1.3	1.5	1.5	1.7	1.5	1.5	1.5
	c_2	0.9	1.5	1.5	1.5	1.5	1.5	1.5
	ω	0.8	0.7	0.7	0.8	0.7	0.7	0.7
	K	2	6	6	2	2	6	2
PSO-WD	w_{int}	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.4, 0.4]$	$[-0.3, 0.3]$	$[-0.8, 0.8]$	$[-0.9, 0.9]$
	V_{max}	1	10	1	10	2	1	5
	c_1	1.5	2.3	1.5	1.5	2.1	1.5	1.5
	c_2	1.5	1.5	1.5	1.7	1.5	1.1	0.7
	ω	0.7	0.7	0.9	0.7	0.8	0.7	0.8
PSOD, CPSOD	λ	0.00005	0.00001	0.00005	0.000005	0.000001	0.00005	0.000005
PSOE, CPSOE	λ	0.00005	0.00001	0.00005	0.000001	0.00005	0.00001	0.00001
	w_0	2	10	0.0001	0.001	0.0001	1	0.0001
BP, BPD, BPE	w_{int}	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.2, 0.2]$	$[-0.4, 0.4]$	$[-0.3, 0.3]$	$[-0.8, 0.8]$	$[-0.9, 0.9]$
	η	0.1	0.4	0.1	0.3	0.3	0.5	0.4
	α	0.9	0.9	0.7	0.9	0.8	0.5	0.5
BPD	λ	0.00001	0.000005	0.000005	0.000001	0.000001	0.000005	0.000005
BPE	λ	0.00005	0.00001	0.00005	0.00005	0.00001	0.00001	0.000001
	w_0	2	10	1	0.1	0.0001	0.0001	0.0001

for 30 algorithm runs was constructed, shown in Figures 1 and 2. As can be seen in Figure 1, PSOD significantly decreased the weight range of the NN: weights produced by PSOD did not exceed the $[-10, 10]$ interval. Although PSOE also produced smaller weights than PSO, Figure 2 shows that both PSO and PSOE produced a few very large weights. Weight decay penalises large weights at the same rate as the small weights, while weight elimination concentrates on driving irrelevant weights to zero, focusing less on penalising large weights. Large weights, however, may cause hidden unit *saturation* [13]. Saturation occurs when the hidden units of a NN predominantly output values close to the asymptotic ends of the activation function range. If large weights dominate the input signals and cause the net input signal to always be a large 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.

Indeed, Figure 3(a) shows that adding a weight decay term to CPSO significantly reduced saturation compared to that of CPSO with no regularisation. Similar frequency diagram can be obtained for PSOD and PSO. Reduced saturation is hypothesised to be one of the reasons for the superior performance of PSOD and CPSOD on the Iris problem compared to PSO and CPSO. The hypothesis is confirmed by similar observations on the Cancer data set: Figure 3(b) shows that both PSOD and PSOE saturated less than the PSO.

Even though weight decay performed better than weight elimination on the Iris data set, the choice of the ultimate regularisation function remains problem-dependent. Table III shows that on the Glass data set, CPSOs performed better than the other algorithms in terms of E_C, and that CPSOE achieved smaller E_G than both CPSO and CPSOD. The weight frequency distribution of CPSOD and CPSOE (Figure 4) shows that neither of the algorithms produced very large weights, and that CPSOE drove more weights to zero than CPSOD. Driving irrelevant weights to zero reduces overall model complexity, which is why CPSOE produced smaller E_G.

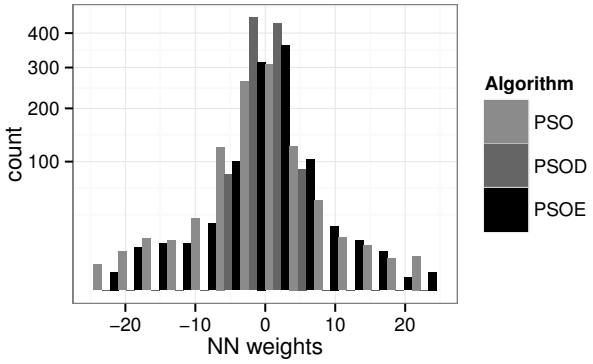


Fig. 1. Frequency distribution of NN weights in (-25, 25), Iris

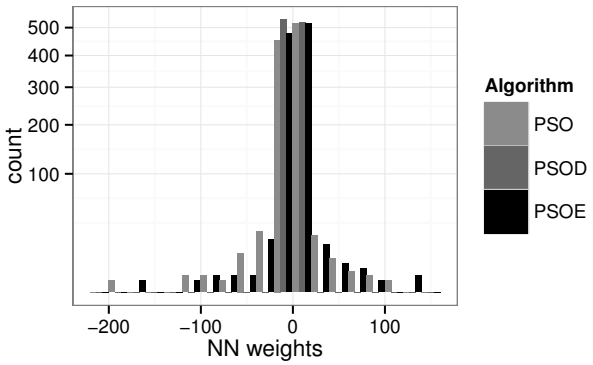


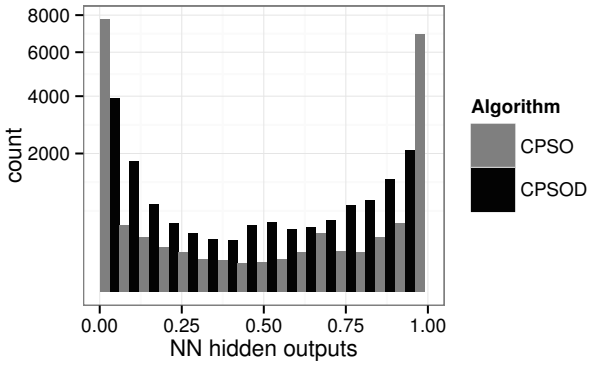
Fig. 2. Frequency distribution of all NN weights obtained, Iris

Note that Glass used a larger NN architecture than Iris in terms of the total number of weights (see Table I), thus there were more irrelevant weights that weight elimination could drive to zero.

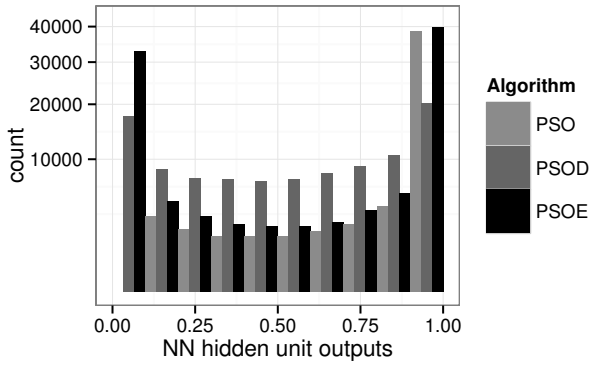
Carvalho and Ludermir [3] suggested PSO-WD as an implementation of weight decay applied to PSO. Table III

TABLE III. AVERAGE E_T, E_G, ρ_F, AND E_C VALUES, WITH CORRESPONDING STANDARD DEVIATION IN PARENTHESIS

Problem		PSO	PSOD	PSOE	CPSO	CPSOD	CPSOE	PSO-WD	BP	BPD	BPE
Iris	E_T	0.003429 (0.002622)	0.006637 (0.001182)	0.003764 (0.002141)	**0.002921** (0.002171)	0.006594 (0.001242)	0.003609 (0.001817)	0.006736 (0.00177)	0.006042 (0.001439)	0.006495 (0.001224)	0.005918 (0.001456)
	E_G	0.016464 (0.012975)	0.01047 (0.005916)	0.013717 (0.013307)	0.017478 (0.011497)	0.01087 (0.005914)	0.016051 (0.013369)	0.011505 (0.006651)	0.010891 (0.004844)	**0.009936** (0.006166)	0.010419 (0.007663)
	ρ_F	309.698 (1041.84)	1.79072 (1.31764)	66.5465 (196.822)	6182.68 (32138.1)	1.89556 (1.50347)	14.0172 (34.6317)	1.94868 (1.45472)	2.08136 (1.35759)	1.76679 (1.40932)	2.81229 (5.94817)
	E_C	6.33333 (4.66092)	8.22222 (3.88862)	5.22222 (3.5755)	5.11111 (3.4722)	9.55556 (4.60994)	**4.88889** (3.7888)	8.66667 (4.34173)	7.55556 (4.00511)	7.55556 (4.62654)	7.11111 (3.98593)
Glass	E_T	**0.00823** (0.001298)	0.010291 (0.001229)	**0.00823** (0.001478)	0.013268 (0.001504)	0.015613 (0.001727)	0.014113 (0.001666)	0.014088 (0.00123)	0.008801 (0.001495)	0.010632 (0.001693)	0.009261 (0.001489)
	E_G	0.024012 (0.008193)	0.024349 (0.009241)	0.026548 (0.010369)	0.022198 (0.008764)	0.024632 (0.007507)	0.021559 (0.00758)	**0.019805** (0.006141)	0.027805 (0.010056)	0.025266 (0.009472)	0.025866 (0.011428)
	ρ_F	3.02987 (1.19103)	2.50126 (1.45283)	3.50781 (1.97716)	1.74331 (0.846926)	1.63973 (0.663929)	1.60671 (0.779507)	1.43269 (0.523836)	3.37867 (1.78511)	2.47466 (1.1738)	3.04206 (1.88762)
	E_C	34.3411 (7.93202)	35.1163 (6.75538)	35.6589 (6.17812)	**15.6589** (5.21568)	15.8915 (5.1491)	16.2016 (5.16357)	36.2015 (7.24384)	36.5891 (7.60186)	37.7519 (8.71616)	34.2636 (8.21499)
Cancer	E_T	0.005241 (0.001035)	0.009592 (0.001404)	0.008861 (0.001565)	0.010753 (0.001589)	0.012635 (0.001548)	0.010736 (0.001556)	0.009318 (0.001275)	**0.005098** (0.001331)	0.008942 (0.001509)	0.007567 (0.001312)
	E_G	0.018862 (0.007308)	0.014231 (0.005356)	0.015443 (0.006605)	0.017814 (0.00552)	0.018451 (0.005844)	0.018136 (0.006478)	0.01406 (0.005069)	0.01698 (0.006029)	**0.01195** (0.005261)	0.013726 (0.006617)
	ρ_F	3.85056 (1.76724)	1.57595 (0.817149)	1.87356 (1.05871)	1.72877 (0.66673)	1.52832 (0.663702)	1.77597 (0.826791)	1.56129 (0.648854)	3.77766 (2.36233)	1.41514 (0.754995)	1.96325 (1.2562)
	E_C	5.78947 (2.34472)	6.43275 (1.862)	5.1462 (1.73512)	7.48538 (2.14538)	8.62573 (2.58666)	7.2807 (2.97827)	5.76023 (2.1571)	5.64328 (1.76469)	5.38012 (2.4241)	**5** (2.13815)
Heart	E_T	0.060978 (0.003108)	0.061885 (0.00289)	0.061565 (0.002696)	0.061494 (0.002521)	0.06134 (0.003343)	0.061634 (0.002536)	0.06256 (0.002677)	**0.030671** (0.003903)	0.036188 (0.003559)	0.050527 (0.004824)
	E_G	0.086955 (0.011132)	0.084988 (0.009492)	0.086433 (0.010838)	0.090294 (0.011232)	0.089394 (0.013879)	0.091529 (0.008879)	**0.084426** (0.009426)	0.123929 (0.015527)	0.12182 (0.018673)	0.101682 (0.015768)
	ρ_F	1.4348 (0.233662)	1.38115 (0.2053)	1.4116 (0.22823)	1.47538 (0.229192)	1.46954 (0.288667)	1.49043 (0.181052)	1.35628 (0.196132)	4.1469 (0.955529)	3.43077 (0.819248)	2.05182 (0.491522)
	E_C	35.4891 (3.55997)	34.4928 (3.42459)	34.6196 (3.07818)	33.7681 (2.89957)	33.8406 (4.27681)	34.0761 (2.82273)	35.3442 (3.29345)	**27.5906** (3.77528)	27.9348 (3.64045)	29.6739 (4.64589)
Diabetes	E_T	0.082813 (0.002139)	0.082871 (0.003182)	0.083397 (0.002813)	0.081936 (0.003457)	0.08221 (0.003645)	0.083216 (0.003848)	0.088018 (0.002198)	0.077494 (0.003861)	0.079775 (0.002674)	**0.077109** (0.003393)
	E_G	0.108005 (0.009498)	0.11083 (0.014276)	0.109111 (0.011091)	0.108167 (0.012814)	0.112442 (0.011838)	0.111179 (0.013766)	**0.103589** (0.00967)	0.120319 (0.01634)	0.119278 (0.012384)	0.124941 (0.015897)
	ρ_F	1.30716 (0.142632)	1.34486 (0.219301)	1.31288 (0.170295)	1.32815 (0.210588)	1.37503 (0.19715)	1.34553 (0.231442)	1.17987 (0.135852)	1.56297 (0.265996)	1.4996 (0.187269)	1.6293 (0.259323)
	E_C	45.7792 (3.85766)	46.3203 (3.63482)	46.71 (4.32729)	43.8961 (4.05914)	45.8009 (4.20735)	45.7576 (3.40942)	45.5195 (2.42576)	43.9827 (4.79281)	43.0303 (4.26638)	**42.7273** (4.80029)
Mackey-Glass	E_T	0.000581 (0.000101)	0.000795 (0.000107)	0.000647 (9.8e-05)	0.000527 (9.4e-05)	0.000804 (0.000102)	0.000625 (9.1e-05)	0.000709 (0.0001)	0.000363 (9.2e-05)	0.000716 (0.000342)	**0.000334** (6.9e-05)
	E_G	0.000798 (0.000347)	0.000992 (0.000449)	0.000732 (0.000444)	0.000791 (0.000434)	0.000977 (0.000364)	**0.000683** (0.000337)	0.000861 (0.000443)	0.00098 (0.00076)	0.00109 (0.000694)	0.000835 (0.000438)
	ρ_F	1.49278 (0.888448)	1.34596 (0.783277)	1.23024 (0.923314)	1.65646 (1.24358)	1.27888 (0.606906)	1.15197 (0.678856)	1.3173 (0.870121)	2.783 (2.1551)	1.6877 (1.16746)	2.61587 (1.4827)
Sunspots	E_T	0.003556 (0.000457)	0.003293 (0.000357)	0.003437 (0.000382)	**0.003191** (0.000387)	0.003259 (0.00032)	0.003332 (0.000498)	0.003508 (0.000417)	0.003539 (0.000469)	0.003734 (0.000591)	0.003575 (0.000626)
	E_G	0.004078 (0.000868)	0.004007 (0.000912)	**0.003853** (0.001162)	0.004073 (0.000936)	0.003887 (0.00101)	0.003957 (0.000925)	0.004178 (0.000991)	0.004431 (0.000971)	0.004413 (0.001088)	0.004322 (0.001274)
	ρ_F	1.16147 (0.279772)	1.22907 (0.307294)	1.13736 (0.370569)	1.30308 (0.382229)	1.20007 (0.31497)	1.21039 (0.351198)	1.2094 (0.330108)	1.26335 (0.286433)	1.19766 (0.302919)	1.2457 (0.436452)



(a) Iris data set



(b) Cancer data set

Fig. 3. Hidden unit output values frequency distributions

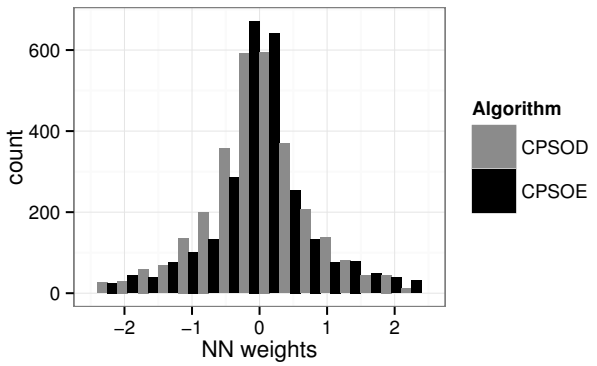


Fig. 4. Frequency distribution of NN weights, Glass

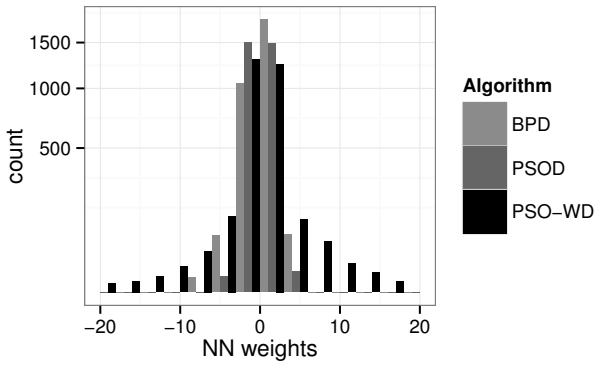


Fig. 5. Frequency distribution of NN weights, Glass

shows that PSO-WD indeed performed competitively and produced the lowest E_G on three of the seven problems considered. However, Figure 5 shows that PSO-WD resembled the behaviour of BPD much less than PSOD did: both BPD and PSOD reduced NN weights to a smaller range than PSO-WD. The dissimilarity between PSO-WD and PSOD shows that these two algorithms are not identical. Applying the regularisation term directly to the objective function of the PSO (i.e., PSOD) may result in more drastic weight decay, which may often be desirable to prevent saturation and divergent swarm behaviour.

Table III shows that regularised PSO and CPSO performed competitively compared to regularised BP and PSO-WD. Regularised PSO and CPSO produced lower E_T, E_G, and E_C values on Iris, Glass, Mackey-Glass, and Sunspots data sets. Regularised PSO and CPSO produced significantly lower values of ρ_F than the BP approaches on the Glass, Heart, Diabetes, and Mackey-Glass data sets. Note that Heart and Mackey-Glass have the largest NN architectures among the seven problems considered (refer to Table I), thus PSO's resistance to overfitting on these problems is especially valuable. Figure 6 shows the E_G profiles obtained on the Heart data set for PSO, PSOD, PSOE, and BP, BPD, and BPE, respectively. Figure 6 illustrates that although the BP approaches converged much faster than the PSO approaches in the beginning of the algorithm run, all BP approaches fell prone to severe overfitting, alleviated but not eliminated by BPE. PSO approaches, on the other hand, took longer to converge, but exhibited

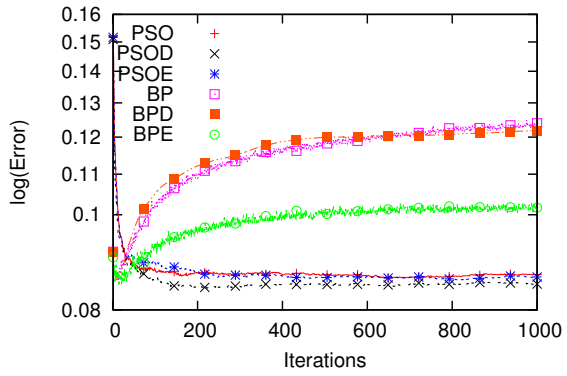


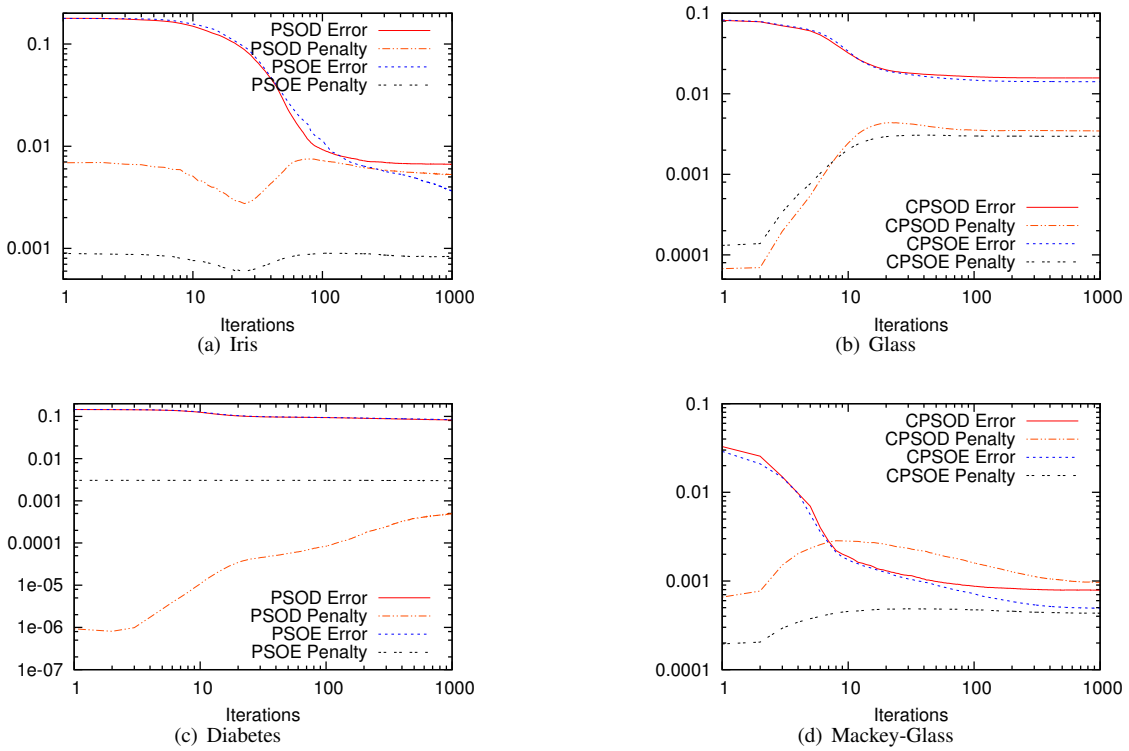
Fig. 6. E_G, Heart data set

less overfitting. Figure 6 also shows that regularised PSO approaches performed better than the non-regularised PSO.

The influence of the regularisation term on the performance of the PSO can be visualised by plotting the penalty value profile together with the E_T profile. Figure 7 displays four such graphs for four different problems considered. Quite curiously, it becomes evident from Figure 7 that the penalty value is not necessarily minimised during the course of optimisation. Indeed, in Figure 7(b) both weight decay and weight elimination penalties grow in the beginning of the algorithm run, and stagnate soon thereafter. The corresponding final E_T and E_G values in Table III confirm that for the Glass problem, depicted in Figure 7(b), adding a regularisation term did not significantly improve PSO performance. The same lack of definite regularisation-aided improvement can be observed for the Diabetes problem in Table III, and Figure 7(c) explains this behaviour by showing that the weight elimination penalty stagnated throughout the algorithm run, and the weight decay penalty kept growing. Table III shows that adding a penalty term to the objective function of the PSO proved beneficial for the Iris and the Mackey-Glass problems. The corresponding error and penalty profiles in Figures 7(a) and 7(d) reveal that for these problems, the penalty value grew in the beginning, but started decreasing when the magnitude of the penalty became comparable with the magnitude of error. Indeed, if the value of the penalty term is much smaller than the error value, the error can be expected to overshadow the penalty, thus causing the penalty to be disregarded. Although the penalty is scaled by the optimised λ hyperparameter, the scaling might need to adjust itself during the algorithm run, as suggested by Carvalho and Ludermir [3]. The application of an adaptive λ to the entire swarm as opposed to the per-particle approach of Carvalho and Ludermir [3] is the immediate topic of future research.

VII. Conclusion

This paper suggested adding a weight regularisation term to the objective function of the PSO in order to improve NN training. The performance of the proposed method was benchmarked against BP with regularisation and PSO-WD on five classification problems and two time-series prediction problems. The results showed that adding a penalty term has a visible effect on PSO performance, and often aids NN training by either driving irrelevant weights to zero (weight elimination), or alleviating saturation by reducing the weight range



(a) Iris (b) Glass

(c) Diabetes (d) Mackey-Glass

Fig. 7. Penalty versus training error

(weight decay). PSO and CPSO with regularisation performed competitively compared to BP with regularisation and PSO-WD. PSO-WD was shown to exhibit different behaviour than PSOD, even though both are implementations of weight decay. Further analysis of penalty values revealed that the penalty term is most beneficial in PSO NN training when the penalty values are of the same scale as the training error values. Further research will investigate different techniques to dynamically scale the penalty term up to the training error term in order to further improve performance.

REFERENCES

[1] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioural sciences," PhD thesis, Harvard University, Boston, USA, 1974.

[2] 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.

[3] M. Carvalho and T. B. Ludermir, "Particle swarm optimization of feed-forward neural networks with weight decay," in *Sixth International Conference on Hybrid Intelligent Systems (HIS'06)*. IEEE, 2006, pp. 5–8.

[4] 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.

[5] 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.

[6] 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.

[7] G. E. Hinton, "Learning translation invariant recognition in a massively parallel networks," in *PARLE Parallel Architectures and Languages Europe*. Springer, 1987, pp. 1–13.

[8] J. Moody, S. Hanson, A. Krogh, and J. A. Hertz, "A simple weight decay can improve generalization," *Advances in neural information processing systems*, vol. 4, pp. 950–957, 1995.

[9] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight elimination with application to forecasting," *Advances in Neural information processings systems*, vol. 3, 1991.

[10] 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.

[11] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE International Conference on Evolutionary Computation*. IEEE, 1998, pp. 69–73.

[12] A. P. Engelbrecht, "Particle swarm optimization: Global best or local best?" in *BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence (BRICS-CCI & CBIC)*. IEEE, 2013, pp. 124–135.

[13] A. B. van Wyk and A. P. Engelbrecht, "Overfitting by PSO trained feedforward neural networks," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[14] A. Gupta and S. M. Lam, "Weight decay backpropagation for noisy data," *Neural Networks*, vol. 11, no. 6, pp. 1127–1138, 1998.

[15] J. Larsen, C. Svarer, L. N. Andersen, and L. K. Hansen, "Adaptive regularization in neural network modeling," in *Neural Networks: Tricks of the Trade*. Springer, 1998, pp. 113–132.

[16] A. Röbel, "The dynamic pattern selection algorithm: Effective training and controlled generalization of backpropagation neural networks," Technische Universität Berlin, Tech. Rep., 1994.