Neural Processing Letters (2020) 52:1123–1153
https://doi.org/10.1007/s11063-020-10290-z

An Analysis of Activation Function Saturation in Particle Swarm Optimization Trained Neural Networks

Cody Dennis[1] · **Andries P. Engelbrecht**[2] · **Beatrice M. Ombuki-Berman**[1]

Published online: 25 July 2020

Abstract
The activation functions used in an artificial neural network define how nodes of the network respond to input, directly influence the shape of the error surface and play a role in the difficulty of the neural network training problem. Choice of activation functions is a significant question which must be addressed when applying a neural network to a problem. One issue which must be considered when selecting an activation function is known as activation function saturation. Saturation occurs when a bounded activation function primarily outputs values close to its boundary. Excessive saturation damages the network's ability to encode information and may prevent successful training. Common functions such as the logistic and hyperbolic tangent functions have been shown to exhibit saturation when the neural network is trained using particle swarm optimization. This study proposes a new measure of activation function saturation, evaluates the saturation behavior of eight common activation functions, and evaluates six measures of controlling activation function saturation in particle swarm optimization based neural network training. Activation functions that result in low levels of saturation are identified. For each activation function recommendations are made regarding which saturation control mechanism is most effective at reducing saturation.

Keywords Feed forward neural network · Particle swarm optimization · Saturation · Activation function

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11063-020-10290-z) contains supplementary material, which is available to authorized users.

✉ Beatrice M. Ombuki-Berman
bombuki@brocku.ca

Cody Dennis
cd15oy@brocku.ca

Andries P. Engelbrecht
engel@sun.ac.za

[1] Department of Computer Science, Brock University, St. Catharines, Canada

[2] Department of Industrial Engineering and Department of Computer Science, Stellenbosch University, Stellenbosch, South Africa

1 Introduction

A feedforward neural network (FFNN) is a simple model inspired by the mammalian brain [2]. A corpus of known information can be encoded within these models. The process of encoding information within the model is known as training the network. During training, the network can identify patterns and relationships in the data, which gives the network the ability to generalize. After successful training, the network is able to produce an appropriate response to novel inputs [48]. This ability has made the use of FFNNs popular for many real world problems in fields such as forecasting, medicine, finance, and others [38]. Gradient descent [50] is perhaps the most commonly used algorithm for training FFNNs [1,2,54]. However, a variety of works have shown that particle swarm optimization (PSO) [26] is an effective FFNN training algorithm [6,31,39,47,48]. In fact, one of the earliest proposed uses of PSO was as a FFNN training algorithm [26], and PSO offers some advantages relative to gradient descent [37]. Since a swarm of neural networks is trained performance is not dependent on a single set of randomly generated weights and biases. PSO is also more robust to rugged or highly variable error surfaces. Finally, PSO is more flexible, since it does not require gradient information.

Each node of the network encapsulates an activation function which defines the node's behavior with respect to an input. When this function is bounded, an issue known as activation function saturation can arise [41]. A saturated node is effectively reduced to a discrete output range; it will primarily output values close to the bounds of the activation function. This reduction damages the overall information capacity of the network [39,40]. Excessive saturation has also been linked to overfitting for PSO trained FFNNs [41] and can slow gradient descent based learning [40]. Several recent works have used PSO to analyze activation function saturation for a limited set of activation functions [38–41,48]. Saturation has been identified as both common and detrimental to PSO's ability as an FFNN training algorithm. Several mechanisms have been proposed for controlling saturation. Proposed methods include [41] scaling of input data, tuning the particle initialization range, the use of a bounded search space, velocity clamping, and the use of weight regularization [39]. Existing works have failed to evaluate commonly used activation functions such as the rectified linear function and the leaky rectified linear function with respect to saturation in PSO trained neural networks. Furthermore, little guidance exists as to which approach to controlling saturation should be chosen for the different activation functions. It has been established that each of the proposed saturation control mechanisms offers improvements relative to vanilla PSO. Unfortunately, little work exists comparing the different control mechanisms for different activation functions. In [38] only tuning initialization range and the use of search space boundaries are compared. In [41] only tuning initialization range, component-wise velocity clamping, and the use of search space boundaries are compared. Both of these works consider only a limited set of activation functions.

This study proposes a new numerical measure of saturation which enables common activation functions such as the rectified linear function, the leaky rectified linear function and softplus to be evaluated with respect to saturation. Although some numerical measures of saturation have been previously proposed [40,41], no existing measure can be used with these three activation functions. The proposed measure of saturation enables an empirical comparison of the saturation behavior of different activation functions and the effectiveness of different saturation control measures for the different activation functions. Simulations are performed using eight popular activation functions on a suite of common benchmark problems. For each function the baseline saturation behavior is established and the relationship

between saturation and performance is discussed. Additional simulations are performed to evaluate the effect of six saturation control measures on each activation function with respect to the baseline. The overall effectiveness of each saturation control measure is discussed and recommendations are made regarding which approach is most effective at reducing saturation for the different activation functions.

The rest of this work is organized as follows: Sect. 2 provides background information on the topics discussed in this work. Section 3 presents a new numerical measure of activation function saturation. Section 4 describes the experiments and measures of performance and behavior used in this study. Section 5 presents the empirical results of this work. Section 6 provides the conclusions of the works and future research directions.

2 Background

This section provides background on the topics discussed in this paper and describes a number of relevant works. Section 2.1 describes the standard PSO algorithm. Section 2.2 gives background information on FFNNs. Section 2.3 describes how the PSO algorithm can be used to train FFNNs and introduces some of the issues involved. Section 2.4 gives a detailed overview of the issue of activation function saturation and discusses a number of works which have examined activation function saturation in PSO based neural network training. Section 2.5 discusses the measures of saturation used previously in the literature. Section 2.6 describes a number of activation functions commonly used in the literature. Section 2.7 provides an overview of the techniques used to date to control activation function saturation in PSO based neural network training.

2.1 Particle Swarm Optimization

Particle swarm optimization [26] is a stochastic population based search algorithm commonly used for the optimization of real valued functions. The algorithm grew out of an attempt to simulate the flocking behavior of birds. Potential solutions to a problem are explored by simulating the movement of the swarm. At each time step, a particle is pulled towards both the best position it has found thus far and the best position found by any particle in its neighborhood. This causes the swarm to home in on more fruitful areas of the search space.

Each particle in the swarm has a state consisting of a position, velocity, personal best position, and neighborhood best position. The particle's position encodes a potential solution to the problem being optimized. This potential solution is evaluated using an objective function, f, which maps an n-dimensional vector to a real number. The value $f(\mathbf{x})$ is referred to as the quality of the solution. Typically a particle's initial position is a point chosen uniformly at random from the search space [26]. From this initial point, the movement of a particle is governed by its velocity, which is calculated using

$$\mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + c_1 \mathbf{r}_1(t)(\mathbf{y}_i(t) - \mathbf{x}_i(t)) + c_2 \mathbf{r}_2(t)(\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t)) \tag{1}$$

where $\mathbf{v}_i(t+1)$ is the velocity of particle i at time $t+1$, $\mathbf{v}_i(t)$ is the velocity of particle i at time t, $\mathbf{y}_i(t)$ is particle i's personal best position at time t, $\hat{\mathbf{y}}_i(t)$ is particle i's neighborhood best position at time t, and $\mathbf{x}_i(t)$ is particle i's position at time t. ω, c_1, and c_2 are referred to as the inertia weight, the cognitive coefficient, and the social coefficient respectively. The inertia weight controls the level of influence of particle i's previous velocity on its current velocity. The cognitive coefficient controls the influence of particle i's personal best position

on its velocity. The social coefficient controls the influence of particle i's neighborhood best position on its velocity. Finally, the vectors $\mathbf{r}_1(t)$ and $\mathbf{r}_2(t)$ are n-dimensional vectors with components sampled uniformly at random from the range [0, 1]. These vectors are re-sampled at each time step and add a stochastic element to the particle's movement [44].

At each time step the position of a particle is first evaluated and then updated according to

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{2}$$

where $\mathbf{x}_i(t+1)$ gives particle i's position at time $t+1$, and $\mathbf{x}_i(t)$ gives particle i's position at time t.

Particles within the same neighborhood share information, allowing the neighborhood as a whole to be directed to the best personal best position found thus far. Taken together, all the neighborhoods within the swarm define a neighborhood topology [27]. The flow of information between particles is governed by the way in which these neighborhoods overlap. The star topology is commonly used with the PSO algorithm [27] and connects all particles through a single global best particle. All particles in the swarm have knowledge of the best position found by the entire swarm. When the star topology is used, the algorithm is referred to as the gBest PSO.

2.2 Feedforward Neural Networks

A feedforward neural network is a mathematical model consisting of a collection of nodes organized into layers. Each node has an associated activation function, defining the node's behavior with respect to an input. Further discussion on activation functions is provided in Sect. 2.6. Each node also has a set of connection weights, connecting it to other nodes in the network. The input of a node is calculated by multiplying each of that node's connection weights by their associated output and taking the sum. The output of a node is found by passing this sum through the node's activation function. This model is inspired by the mammalian brain [2], with the activation function being analogous to action potential found in biological neurons [48] and connection weights analogous to synapses.

Given a network of sufficient size, an arbitrary mapping between inputs and outputs can be encoded in the network with arbitrary precision [28], provided that the correct set of weights and biases can be found. The process of finding the correct set of weights and biases is referred to as training the network. When a set of known inputs and desired outputs are used, it is referred to as supervised training. This sort of training can be viewed as an optimization problem. The goal of training is to minimize error, a measure of the difference between the responses of the network and the desired responses.

Two of the most popular ways of quantifying network error are the mean squared error [19],

$$MSE = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} (t_{kp} - o_{kp})^2}{KP} \tag{3}$$

and the cross entropy error [19],

$$CE = -\sum_{p=1}^{P} \sum_{k=1}^{K} t_{kp} log(o_{kp}) \tag{4}$$

where K is the number of output neurons, P is the number of examples, t_{kp} is the desired output of neuron k on example p, and o_{kp} is the actual output of neuron k on example p. Typically, cross entropy is used for classification problems with a softmax output layer, and

MSE is used for regression problems with a linear output layer. When softmax is used, the output of each node in the output layer is calculated using

$$o_{kp} = \frac{e^{net_{kp}}}{\Sigma_{c=1}^{K} e^{net_{cp}}} \tag{5}$$

where net_{kp} is the input to output node k on example p. When a linear output layer is used, each output node simply uses the function $g(x) = x$ as its activation function.

2.3 Training Neural Networks with Particle Swarm Optimization

One of the earliest proposed uses of PSO was neural network training [26]. PSO offers a number of advantages relative to gradient descent [37]. First, since a population of neural networks is trained performance is not dependent on a single set of randomly generated weights and biases. Second, gradient descent is essentially a hill climber. It is potentially vulnerable to local minima. Finally, PSO is more flexible since it does not require gradient information. Many previous works have applied PSO as a FFNN training algorithm [6,20, 31,37–41,46–49]. Training a neural network is simply an optimization problem. The goal is to find a set of real numbers, weight and bias values, which produce acceptably low error values over some dataset. Since PSO is an optimization algorithm intended for optimizing real valued functions it can be applied directly to the problem of FFNN training, without modification.

To train a FFNN with PSO, the weights and biases of the network are simply flattened to produce a single vector. This vector acts as the position of a particle [46]. This means that each particle in the swarm contains a distinct value for each weight and bias of the network. Having encoded a complete neural network in each particle, the objective function of the PSO is simply the error function of the FFNN. To evaluate the quality of a particle, a FFNN is reconstructed from the particle's position, and the error of the FFNN is calculated over the training set. The value of the error is used directly as the quality of the particle. The dimensionality of the resulting position vector is equal to the number of weights and biases in the network and is given by

$$W = (I + 1) * J + (J + 1) * K \tag{6}$$

where W is the total number of weights and biases, I is the number of input nodes, J is the number of hidden nodes, and K is the number of output nodes.

A number of previous works have shown PSO to be an effective neural network training algorithm [6,20,31,39,48]. However, swarm divergence is common when PSO is used to train neural networks [49]. This issue has been attributed to excessive activation function saturation [47,49]. Excessive saturation has also been linked to overfitting [41] and hypothesized to contribute to the poor performance of PSO in very high dimensional problems [38].

2.4 Activation Function Saturation

Activation function saturation occurs when bounded activation functions are used in the hidden layer. A node is considered to be saturated when the majority of its outputs are close to the bounds of its activation function. When an FFNN is saturated, its nodes are effectively reduced to a discrete output range, damaging the overall information capacity of the network [40,41].

As the output of a bounded function approaches its asymptote, its gradient becomes shallow. As the gradient becomes shallow, the change in function output relative to the change in function input can become insignificant [49]. When this occurs, the magnitude of the values in a particle's position vector can grow arbitrarily with little impact on solution quality. This encourages divergence of the swarm, or causes particles to become trapped on a plateau, preventing solution quality from improving [49].

As a simple example, consider a classification task where the classic logistic function is used for all hidden and output nodes. For simplicity assume that there is a single output node, and that target values are either 0 or 1. Clearly, a certain level of saturation in the output layer is necessary in order to minimize error in this situation. This situation may even require saturation in the hidden layer, for example, if the problem is not linearly separable. However, the magnitude of an output node's output is not an indicator of the level of confidence of a classification [29,40,41]. Increasing saturation beyond the level required to correctly classify an input pattern may reduce error, but it does not represent a true gain in performance. In this particular example, it may be possible for node output to approach the bounds of the logistic function infinitely. Each step could reduce error, however, the proportion of correctly classified examples may never increase.

2.5 Measuring Saturation

A number of methods for evaluating the saturation of the activation functions within a network have been discussed in the literature. The first method is a manual examination of the output of nodes in the network for a set of input data [41]. While this does give an indication of the saturation of the network, it is a highly tedious process prone to human error and bias. Furthermore, this does not provide a means of easy comparison between different networks on different problems. The next option, which has been used in [18,38–41], is producing histograms describing the distribution of node output across a set of input data. This provides a quick and convenient way to examine the saturation of a network and even provides a means of comparing the saturation of different networks. However, comparison between networks still relies on visual inspection. Examining a number of different networks will be tedious, and is still subject to human error and bias. A numeric measure of saturation, which allows for automated comparisons and statistical testing would be ideal. Currently two numerical measures of saturation are available in the literature. The first saturation measure is [41]

$$\sigma_h = \frac{\Sigma_{p=1}^{P} \Sigma_{j=1}^{J} |net_{jp}|}{P\,H} \tag{7}$$

where h is the hidden layer number, P is the number of examples, J is the size of hidden layer h, and net_{jp} is the net input to hidden node j on pattern p.

Unfortunately, since the relationship between the magnitude of net input and saturation is dependent on the activation function used, σ_h cannot be used to compare saturation between two different activation functions. Furthermore, this measure of saturation requires that the activation function used has both an upper and lower bound. For an activation function like softplus, a net input with a large positive magnitude does not produce saturation, yet σ_h would indicate saturation if such net inputs were present.

A second numeric measure of saturation was later proposed to address issues with σ_h [40]. This measure is derived from the frequency distribution of node output and is calculated as follows:

$$\varphi_B = \frac{\Sigma_{b=1}^{B} |\bar{g}'_b| f_b}{\Sigma_{b=1}^{B} f_b} \tag{8}$$

$$\bar{g}'_b = \frac{2(\bar{g}_b - g_L)}{g_U - g_L} - 1 \tag{9}$$

where B is the number of bins, f_b is the weight assigned to bin b, g is an activation function with two boundaries, $\bar{g}_b$ is the average of the function outputs in bin b, or 0 if the bin is empty, g_L is the lower bound of the function, and g_U is the upper bound of the function.

It is assumed that, for a completely saturated network, the output of all nodes across all examples will fall into either the leftmost or rightmost bins, resulting in $\varphi_B = 1$. When node outputs tend to fall in the center bins, φ_B will tend toward 0. For a normal distribution of node outputs φ_B will be close to 0.5 [40]. φ_B allows saturation to be measured independently of the activation function used and was shown to be valid in [40]. While this is an improvement over σ_h, φ_B can still only be used to measure saturation in functions with two bounds.

2.6 Activation Functions

Each node in a neural network encapsulates an activation function which defines the way in which that node transforms input. The choice of activation function directly affects the error surface which a training algorithm must navigate. For example, an activation function with a steep gradient may require relatively small changes in the weights and biases of the network since each change has a relatively large effect on node output. Typically, a non-linear activation function is chosen for the hidden layer. Non-linear functions are used because non-linearity in the hidden layer allows a FFNN to approximate any non-linear mapping between inputs and outputs, provided that a large enough hidden layer is used [28].

The linear, or identity, activation function is given by $f_{lin}(x) = x$. This function has a range of $(-\infty, \infty)$, and a constant gradient of 1. As a result, this function cannot saturate. It has been suggested that the use of the linear activation function, or other unbounded functions, in the hidden layer may be an effective solution to the saturation problem [49]. However, if the linear function is used a larger hidden layer will be required in order to approximate non-linear functions [47,49]. Previous works have found that switching to the linear function in the hidden layer, without modifying hidden layer size, improves the convergence of the PSO algorithm [47], with the cost of increased training and generalization error [48].

The rectified linear [9] (ReLU) activation function is given by $f_{rel}(x) = max\{0.0, x\}$. This function is unbounded in the positive direction, with a range of $[0, \infty)$. The function has a gradient of 1 for inputs above 0, and a gradient of 0 for inputs below 0. When net input is less than 0, net input can be changed arbitrarily, provided it stays below 0, with no effect on function output. This may produce large, entirely flat, regions of the search space which could trap particles. The ReLU activation function has been successfully used in the field of deep neural networks to address an issue related to saturation known as the vanishing gradient problem [5,30]. The ReLU activation function has also been successfully used in PSO based neural network training [48].

The leaky rectified linear [30] (leaky ReLU) activation function is given by $f_{lrel}(x) = max\{0.01x, x\}$. This is another unbounded function, with a range of $(-\infty, \infty)$. For inputs below 0, the gradient of this function is 0.01, and for inputs above 0, the gradient is 1. This function was proposed as in improvement to the ReLU activation function, addressing the gradient of 0 for inputs less than 0 [30].

The softplus [9] activation function is given by $f_{sof}(x) = log(1 + e^x)$. This function is a smooth approximation of the ReLU activation function, proposed to address ReLU's discontinuity for an input of 0. Like ReLU, softplus is unbounded in the positive direction, with a range of $(0, \infty)$. This function has been successfully used in PSO based FFNN training, and was found to produce superior generalization performance to the leaky ReLU function [48].

The logistic activation function is given by $f_{log}(x) = \frac{1}{1+e^{-x}}$. This activation function is very commonly used in FFNNs [2] and has been shown to result in saturation in PSO based neural network training [40,41]. This function has a range of $(0, 1)$ and is approximately linear over the domain $[-2, 2]$. Outside of this domain the gradient of the function becomes shallow.

The hyperbolic tan (tanh) activation function is given by $f_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Tanh is another function very commonly used in FFNNs. The range of tanh, $(-1, 1)$, is wider than that of the logistic function, and unlike the logistic function, is symmetric around 0. This function has been shown to result in saturation in PSO based neural network training [40,41]. Over the domain of $[-2, 2]$ the tanh function is approximately linear, however, outside of this domain the gradient of the function becomes shallow.

LeCun's tanh [29] (Ltanh) function is given by $f_{ltan}(x) = 1.7159 f_{tanh}(\frac{2}{3}x)$. The Ltanh function was proposed as an improvement to the tanh function. It has a wider range, $(-1.7159, 1.7159)$, and a softer slope than tanh. Over the domain of $[-2, 2]$ the Ltanh function is approximately linear, however, outside of this domain the gradient of the function becomes shallow. This function has been shown to result in saturation in PSO based neural network training [40,41].

The Elliot [12] activation function is given by $f_{ell}(x) = \frac{x}{1+|x|}$. The Elliot function was proposed as a computationally efficient approximation of the tanh function and has similar properties. The Elliot function has a range of $(-1, 1)$, and a much softer slope than tanh. Over the domain of $[-1, 1]$ the Elliot function is approximately linear, however, outside of this domain the gradient of the function becomes shallow. This function has been shown to result in saturation in PSO based neural network training [40,41].

2.7 Controlling Saturation

A variety of techniques for controlling activation function saturation in PSO based neural network training have been proposed in the literature. Some examples include the use of unbounded activation functions [49], scaling of input data and targets [48], tuning of the search space size and particle initialization range [38,41], velocity clamping [41], weight regularization [39], or the use of adaptive activation functions [53]. Evidence has been presented in the literature showing improved performance, reduced saturation, or both for each of these approaches for controlling saturation in PSO based neural network training. Indeed, each of these approaches can be clearly related to saturation behavior and the potential effectiveness of the approach can be intuitively justified. Unfortunately, little data is available comparing the relative effectiveness of each approach on different activation functions. This leaves a practitioner with many plausible recommendations to follow, and little guidance on how to choose between them. This section describes and justifies the most common methods used in the literature.

2.7.1 Input Data Scaling

The scaling of input data has long been recognized as an important preprocessing step in neural network training. One long standing rule of thumb is that input data should be scaled such that the average of each input variable is close to 0, and target data should be scaled to be within the range of the activation function. This rule was given in [29] where it was shown that for gradient based learning violations of the rule can increase training time. It has also been shown that scaling input data to match the active range of the chosen activation function can improve performance [15]. Rakitianskaia and Engelbrecht [41] noted that saturation occurs when the magnitude of net input became too large. One factor which influences the magnitude of net input to a hidden node is the magnitude of the input data. If the magnitude of the input data is appropriately tuned during preprocessing, then the magnitude of net input to the hidden nodes at the beginning of training can be tuned. This can prevent immediate saturation, and may reduce or prevent saturation during training.

2.7.2 Initialization Range Tuning

The initial set of weights and biases directly determines the magnitude of the initial net inputs and the initial saturation of the network. Furthermore, by determining the state of the network when training begins, the initial weights and biases of the network play a role in determining the behavior of the training algorithm. Typically the weights and biases of the network are initialized to a small range around 0 to prevent the initial set of net inputs from having an unfavorably large magnitude [7]. Wessels and Barnard [51] showed that weights initialized to the range $[-\frac{1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$ help prevent a training algorithm from becoming trapped in local minima. It has previously been found that varying the initialization range can influence saturation [41]. Perhaps counter intuitively, it was found that a very small initialization range around 0 tended to encourage saturation. It was hypothesized that this is a result of the often flat error surface found around the origin.

2.7.3 Search Space Boundaries

By imposing search space boundaries, an absolute limit is placed on the magnitude of weights and biases in the network. Assuming that there is some finite limit on the magnitude of input values, then placing boundaries on the search space places an absolute limit on the magnitude of net input to a node. The search space is typically unbounded when gradient based methods are used to train neural networks. However, boundaries are common when PSO is used to optimize a problem. It should be noted that imposing search space boundaries has the potential to significantly alter the behavior of the swarm. It has been theoretically proven, under minimal assumptions, that when particles are initialized randomly within the search space a large proportion of the particles will leave the search space within the first few iterations of the search [23]. This means that a large proportion of the particles will be influenced by the chosen boundary handling mechanism. One simple approach is to reinitialize particles that violate the search space boundaries uniformly at random within the feasible search space. This approach is reasonable whenever no additional information is known about the problem since it does not bias the search [22]. Previous works have found that the use of search space boundaries can improve training performance, encourage convergence, [38] and reduce saturation [41] in PSO based FFNN training.

2.7.4 Component-Wise and Normalized Velocity Clamping

It has been suggested that the rate at which particles are able to move throughout the search space can influence the level of saturation in a network [41]. If a limit is placed on the rate at which a particle can travel, a limit is effectively placed on the possible magnitude of weights and biases in the network. If a weight with the initial value w can change by at most $\pm u$ per iteration, and training lasts at most q iterations, then the weight must fall within the range $[w-uq, w+uq]$ at the final iteration. Furthermore, imposing a maximum velocity can force particles to take a more detailed sampling of the search space. This will reduce the rate at which net input, and saturation, can increase.

Component-wise clamping [10] is one method of velocity clamping implemented as

$$v_{i,d}(t+1) = \begin{cases} v_{i,d}(t+1) & |v_{i,d}(t+1)| < V_{max,d} \\ V_{max,d} & V_{max,d} < v_{i,d}(t+1) \\ -V_{max,d} & v_{i,d}(t+1) < -V_{max,d} \end{cases} \tag{10}$$

where $v_{i,d}(t+1)$ gives the velocity of particle i in dimension d at time $t+1$ and $\mathbf{V}_{max}$ is the maximum velocity. A potential downside of component-wise velocity clamping is that when the velocity of each dimension is clamped independently, the original direction of the velocity can be lost. Unfortunately, information about which areas of the search space appear promising is encoded in the trajectory of the particle. As a result, the particle may lose information regarding which areas of the search space should be explored [42].

Normalized velocity clamping was proposed to address the loss of information caused by component-wise velocity clamping [33], and is described by

$$v_{i,d}(t+1) = \begin{cases} v_{i,d}(t+1) & ||v_{i,d}(t+1)|| \leq V_{max,d} \\ \frac{V_{max,d}}{||v_{i,d}(t+1)||} v_{i,d}(t+1) & v_{i,d}(t+1) > V_{max,d} \end{cases} \tag{11}$$

where $||.||$ denotes the Euclidean norm. Instead of restricting the magnitude of each dimension of the velocity vector, normalized velocity clamping restricts the magnitude of the velocity vector as a whole. Whenever the magnitude of the vector exceeds V_{max}, it is reduced to V_{max}. This preserves the trajectory of the particle, and any information encoded in that trajectory. However, this method of velocity clamping is sensitive to outliers. A large velocity in one dimension can force all other dimensions to a very low velocity. This may prevent the particle from fully exploring certain areas of the search space [33].

2.7.5 Weight Decay

Weight decay is a popular form of weight regularization [39] which penalizes the overall magnitude of weights in the network. The goal of weight regularization is to reduce both the error of the FFNN and the complexity of the FFNN [39]. To achieve this goal, a complexity penalty term is added to the objective function as follows

$$f(\mathbf{x}) = E_{nn} + \lambda E_p \tag{12}$$

where E_{nn} is the error of the FFNN, E_p is the penalty term, and λ is a hyper-parameter controlling the strength of regularization. The weight decay penalty term is given by

$$E_p = \frac{1}{2} \Sigma_{l=1}^{W} w_l^2 \tag{13}$$

where W is the total number of weights and biases in the network and w_l is the lth weight or bias in the network.

Previous work has shown that the use of weight decay to control the growth of weights and biases in the network and drive irrelevant weights to 0 can improve generalization when gradient based training algorithms are used [32]. In addition to improving generalization, weight decay could also be expected to reduce saturation, since it tends to reduce the magnitude of weights in the network. Based on this hypothesis the use of weight decay in PSO based FFNN training was examined in [39], where it was found that weight decay significantly reduced saturation. Weight decay has also been found to significantly improve generalization in PSO based FFNN training [3].

3 A New Saturation Measure

Several techniques have been used in the literature to examine and quantify activation function saturation in a neural network. Unfortunately, the authors of this paper are not aware of an existing technique which is sufficient for use in this work. A survey of existing techniques can be found in Sect. 2.5. A visual examination of hidden node output histograms does any allow for hypothesis testing and does not provide a means to quantifying saturation. The measure σ_h [41] quantifies saturation, but it cannot be used to compare different activation functions. Furthermore, σ_h cannot be used with activation functions having a single asymptote since σ_h makes no distinction between the magnitude of hidden node output, and saturation. According to σ_h a node with relatively large output is saturated, even if the the node's output is not approaching an asymptote.

The most promising candidate is φ_B [40]. However, this measure can only be used with activation functions having two bounds. This work examines the saturation behavior of functions such as ReLU and softplus, which each have one boundary, therefore φ_B is not suitable. A secondary issue exists with φ_B, which results from differences in the way in which different activation functions approach their asymptotes. The measure φ_B provides a simple indicator of the typical distance between node output and the closest asymptote of the activation function. As φ_B tends to 1, all nodes in the network tend to output values close to the upper or lower asymptote of their activation function for all examples. Since activation function saturation occurs as node outputs approach an asymptote of the function, the relationship between φ_B and network saturation can be seen intuitively. Unfortunately, not all activation functions approach their asymptotes in the same way. Figure 1 provides a plot of the Elliot and tanh activation functions. The tanh function provides an example of typical behavior. From Fig. 1 it can be seen that the tanh function is approximately linear close to the origin, and beyond the active domain of tanh its gradients very quickly approach 0. Similar behavior is displayed by the logistic and Ltanh activation functions.

In contrast, the gradient of the Elliot function begins to become shallow much earlier than that of tanh, and the function approaches its boundaries extremely slowly. Since the gradient of the Elliot function becomes shallow early, extremely large inputs are required to push its output close to its asymptote, relative to functions such as tanh, Ltanh, and logistic. As a result, it is likely that φ_B will indicate that the Elliot function will saturate relatively little under most circumstances. Indeed, previous work using φ_B has indicated that the Elliot function is relatively resistant to saturation [40].

The main issue associated with activation function saturation is that the gradient of the function becomes shallow as saturation occurs. Shallow gradients produce a state where

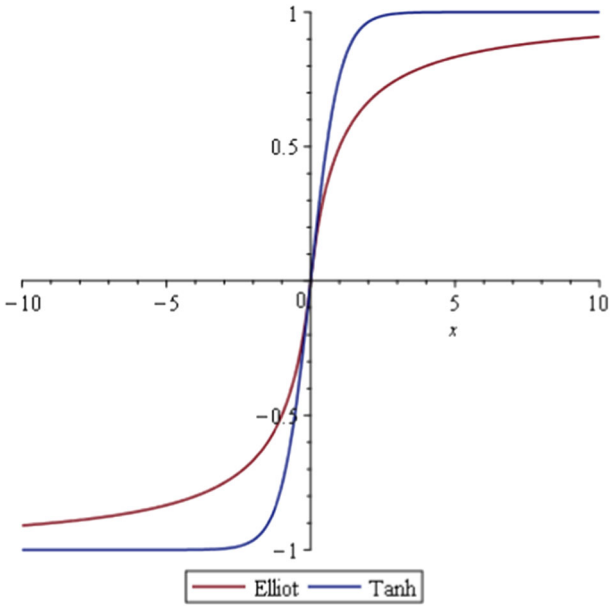


Fig. 1 A plot of the Elliot function and the tanh function

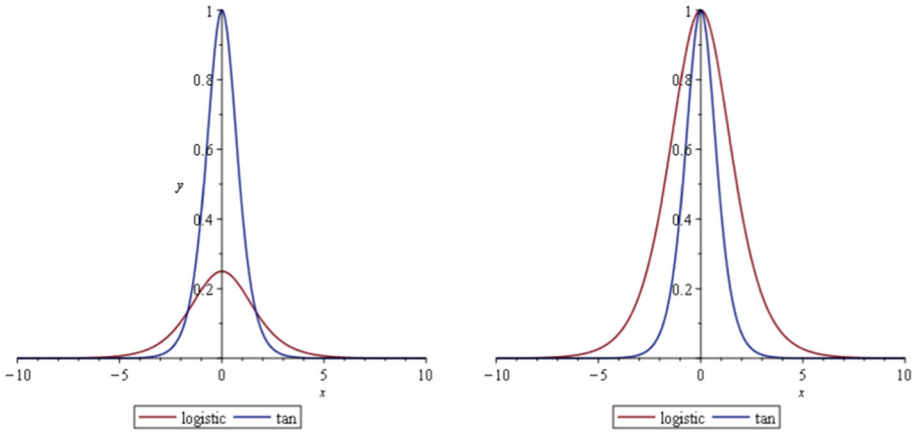
change in function output is negligible relative to change in function input. This means that changes to the weights going into a node may have little to no effect on node output. As a result, a training algorithm may have difficulty improving on a solution or may increase the magnitude of a weight endlessly producing continuous but ultimately negligible improvement. The measure φ_B provides an indirect measurement of the gradients of activation functions within a neural network. Since different functions approach their asymptotes differently, the relationship between φ_B and the gradient will be different for different functions. It is believed that a more direct measure of the gradients of activation functions within a neural network will provide a more reliable measure of activation function saturation, particularly when comparing functions which exhibit different behavior near their asymptotes.

This study proposes a modification of φ_B to address the issues described above, and enable comparison between functions with differing numbers of asymptotes. The proposed measure is referred to as ϵ_B. Unlike σ_h, ϵ_B is independent of the range of an activation function, allowing for comparison and hypothesis testing between different functions. Unlike φ_B, ϵ_B directly measures the gradients of activation functions. Since ϵ_B measures the gradient directly, it does not require that two asymptotes exist. Furthermore, if the gradient of an activation function becomes shallow relatively far from an asymptote ϵ_B can detect it. ϵ_B, is calculated as

$$\epsilon_B = \frac{\Sigma_{b=1}^{B} \bar{K}_b c_b}{\Sigma_{b=1}^{B} c_b} \tag{14}$$

where

$$\bar{K}_b = \frac{\Sigma_{e=1}^{E_b} |G(o_e)|}{E_b} \tag{15}$$



(a) Gradients of the logistic function and tan.

(b) G for the gradients of the logistic function and tan.

Fig. 2 Line plots illustrating the effects of G on the gradients of the logistic function and tan

and

$$G(x) = \begin{cases} = \frac{f'(x)}{max(f')} & \text{for } max(f') < 1 \\ = 1 & \text{for } f'(x) > 1 \\ = f'(x) & \text{otherwise} \end{cases} \tag{16}$$

The values of $G(x)$ for all nodes for all training examples are divided into B bins. c_b gives the number of gradients in bin b, E_b gives the number of gradients in bin b, o_e is a net input in bin b, $f'(x)$ gives the gradient of the activation function at x, and $f'(x) \geq 0 \; \forall x$. Note that this assumes all nodes use the same activation function. If different activation functions are used in different nodes, then f' must be calculated appropriately. In the case where $E_b = c_b$ the number of bins becomes irrelevant, and ϵ_B is simply the arithmetic mean of G of each input to each node of the network over all training examples. Alternatively, c_b may be given different values in order to change the weighting of each bin, this is similar to φ_B.

Figure 2 illustrates the effect of G on the gradients of the logistic function and the tan function. From Fig. 2 it can be seen that the logistic function has a relatively small maximum gradient. Suppose we have a set of inputs whose values fall in the range $[-0.25, 0.25]$. These inputs are within the active range of both the logistic function, and the tan function, but the tan function will produce gradients close to 1 and the logistic function will produce gradients close to 0.25. For such inputs the gradients of both functions will be close to their steepest, but if only the gradients are considered then the tan function may appear to saturate less than the logistic function. G addresses this issue by scaling the gradients of the logistic function, as can be seen in Fig. 2.

The function G does not simply give the gradient of the activation function. As a node becomes saturated, the gradient of its activation function tends to 0. The measure ϵ_B must reflect when gradients are tending to 0. It does not need to indicate when gradients become large. In fact, a few very large gradients may hide the fact that the network is becoming saturated. To prevent this, large gradients are simply given a value of 1. Similarly, the fact that a function has a soft slope does not necessarily mean that the function has saturated. Saturation occurs as the magnitude of the net input becomes large, the function approaches an optimum, and the gradient tends to 0. A function with a soft slope in its active domain

should not automatically be considered to saturate more than a function with a steeper slope. To prevent this G scales the gradient of a function with a soft slope to the range [0, 1]. This scaling causes a proportional increase in the rate of change of the gradient of the function. As the activation function saturates, G approaches 0 at an increased rate of $\frac{1}{max(f')}$ times faster than f'. Since G's rate of change increases by the same factor that f' is scaled, G will accurately reflect when f is saturating. A detailed empirical analysis of ϵ_B is performed in Sect. 5.1.

4 Experimental Procedure

This section describes the experimental procedures used in this study. Section 4.1 summarizes the datasets used and the simulations performed in this study. Section 4.2 summaries the hyper-parameters used for the simulations and how these parameters were optimized. Section 4.3 describes the measures used to quantify PSO performance and behavior.

4.1 Datasets and Experiments

Table 1 summarizes the suite of common benchmark datasets used in this study. For all datasets all attributes are real valued. Equations (17), (18), (19), and (20) give the equations used to generate the Henon map, logistic map, TS5, and Mackey-Glass datasets respectively:

$$z_t = 1 + 0.3z_{t-2} - 1.4z_{t-1}^2 \qquad z_0, z_1 \in U(-1, 1) \tag{17}$$

$$z_t = rz_{t-1}(1 - z_{t-1}) \qquad z_0 = 0.01, r = 4 \tag{18}$$

$$z_t = 0.3z_{t-6} - 0.6z_{t-4} + 0.5z_{t-1} + 0.3z_{t-6}^2 - 0.2z_{t-4}^2 + n_t \qquad n_t \in N(0, 0.05) \tag{19}$$

$$z_t = (1 - b)z_{t-1} + a\frac{z_{t-30}}{1 + z_{t-30}^{10}} \qquad a = 0.2, b = 0.1 \tag{20}$$

These four datasets were generated as described in [48]. For the logistic map, Henon map, and TS5 the following procedure was used: First, 1000 points were sampled from each time series. Then a training example was constructed for each possible value of z_t such that $t > I$, where I is the number of input nodes for the problem. For each target value z_t, the inputs are the I preceding values. For the Mackey-Glass equation the inputs for each example were z_{t-1}, z_{t-7}, z_{t-13} and z_{t-18}. All datasets have been min-max normalized to the range [0, 1].

For each dataset, 30 independent runs were performed for each activation function described in Sect. 2.6. This set of experiments was used to establish the baseline behavior and performance of the activation functions. After establishing a baseline, 30 independent runs were performed for each activation function, on each data set, using each of the saturation control measures described in Sect. 2.7. Section 4.2 describes the hyper-parameters and the optimization process used.

4.2 Algorithm Configuration

This section describes the hyper-parameters used and the optimization process used to select those parameters. Section 4.2.1 describes the hyper-parameters used when establishing the baseline performance and behavior of each activation function and Sect. 4.2.2 describes the hyper-parameters used when evaluating each saturation control measure.

Table 1 A summary of the datasets used in this study

	Classification	Problems		
DataSet	Attributes	Classes	Patterns	Dimensionality
BRCA[52]	30	2	569	299–1454
Wine [17]	13	3	178	156–309
Diabetes [36]	8	2	768	90–365
Iris [16]	4	3	150	43–91
Glass [8]	9	6	214	166–342
	Regression	Problems		
DataSet	Attributes	Targets	Patterns	Dimensionality
Henon map [48]	2	1	998	9–69
Logistic map [48]	4	1	996	13–103
Mackey-Glass [35]	4	1	981	43–145
Sunspot [4]	4	1	311	31–109
TS5 [48]	10	1	990	25–109

4.2.1 Vanilla PSO

A standard gBest PSO was used for all experiments. The gBest PSO was selected based on [14], where it was shown that neither of two very common PSOs, gBest or lBest, consistently outperformed the other. This work concluded that the choice of neighborhood topology is not significant when the goal is to compare the effects of changes to the PSO algorithm. Each swarm has 20 particles and all particles have been initialized uniformly at random in the range $[-1, 1]$ based on the recommendations in [7]. By default no bounds were placed on the search space since one goal of this study is to investigate the effect of search space boundaries. The initial velocity of each particle was set to 0. This has been shown to promote convergence [13]. The hyper-parameters c_1, c_2, ω, and hidden layer size have been optimized using the hyper-parameter tuning tool SMAC (Sequential Model-based Algorithm Configuration) [24]. SMAC is a state of the art algorithm configuration tool [25,45] which has been found to significantly improve the performance of other state of the art algorithms [11]. These four hyper-parameters have been optimized over 1000 simulations for each combination of dataset and activation function to minimize validation error. More detail regarding this measure can be found in Sect. 4.3. Incumbent configurations were evaluated over a maximum of 30 simulations. The default values of c_1, c_2, ω, and hidden layer size provided to SMAC were taken from [48], where these parameters were optimized for the same problems, but only for one activation function. The allowable range for c_1, c_2 and ω was chosen based on the range of values suggested for general purpose use in the literature. A survey of the values suggested can be found in [21]. The range of allowable values for hidden layer size was selected based on the optimized sizes in [48]. The lower bound was set to $\frac{1}{4}$ of the optimized size and the upper bound was set to $\frac{7}{4}$ of the optimized size. Online Resource 1 provides the optimized hyper-parameters used with each activation function and dataset. For all classification problems, CE was used as the objective function with a softmax output layer. For all regression problems, MSE was used as the objective function with a linear output layer.

4.2.2 Saturation Control Measures

The hyper-parameters of each saturation control measure were optimized over 300 simulations using SMAC for each combination of activation function and problem to minimize validation error. More information regarding this measure can be found in Sect. 4.3. Incumbent configurations were evaluated over a maximum of 30 simulations. Any hyper-parameters or configurations not described in this section are set as described in Sect. 4.2.1. An extensive evaluation of both normalized and component-wise velocity clamping was performed in [33]. This work considered values of V_{max} in the range [0, 0.5]. Larger values were not tested because no significant difference was detected between unclamped swarms and swarms with $V_{max} = 0.5$. Based on these results, the range $[0, 0.5]$ was used when optimizing V_{max}, and the default value was set to 0.25. When optimizing λ the allowable range used was [0.0, 0.0001] with a default value of 0.00005. This range was selected based on the values of λ tested in other works [3,32,39]. The value 0.0001 was the largest value considered among several other works. When optimizing the initialization range and search space boundaries, the allowable range was $[-100, 100]$ and the default range was $[-1, 1]$. The allowable range was intended to be nonrestrictive and was chosen based on the magnitude of particle positions observed in an unbounded search space. The default was selected to be consistent with the default presented in Sect. 4.2.1. Finally, the default range for input data scaling was [0, 1], which was selected to be consistent with the default from Sect. 4.2.1 and the allowable range was $[-10, 10]$. Online Resource 1 provides the optimized hyper-parameters used with each saturation control measure. Whenever a particle leaves the boundaries of the search space, its position is reinitialized uniformly at random within the search space and its velocity is reset to 0.

4.3 Algorithm Measurements

For each simulation, the available data was split with 70 percent used for training and 30 percent used for testing. Stratified K-fold cross validation was used with $k = 5$ to generate a training error and a validation error. This value of k was selected based on the relatively small number of examples available for some of the tested datasets, Iris for example. The testing data was used to generate a generalization error. Training, validation, and generalization errors are referred to as E_T, E_V, and E_G respectively. ϵ_B as described in Sect. 3 was primarily used to quantify saturation. The relationship between ϵ_B and φ_B, as described in Sect. 2.5, is considered in Sect. 5.1. Swarm diversity calculated using

$$D = \frac{1}{|S|} \sum_{i=1}^{|S|} \sqrt{\sum_{j=1}^{W} (x_{ij} - \bar{x}_j)^2} \tag{21}$$

where $|S|$ is the total number of particles, W is the dimensionality of the problem, x_{ij} is the position of particle i in dimension j, and $\bar{x}_j$ is the position of the swarm center in dimension j, has been used to examine swarm behavior over the course of a simulation. This measure was shown to be valid in [34]. Finally, $\rho = \frac{E_G}{E_T}$ was used to quantify overfitting, shown in [43] to be valid.

Table 2 Correlation coefficients and p values from correlation tests between ϵ_B and φ_B over all datasets using the logistic, tanh, Ltanh, and Elliot activation functions

Function	Kendall's τ	p value	Spearman's ρ	p value
Logistic	0.9758449	< 0.0001	0.9983843	< 0.0001
tanh	0.9786697	< 0.0001	0.9990784	< 0.0001
Ltanh	0.963755	< 0.0001	0.9973938	< 0.0001
Elliot	0.9409604	< 0.0001	0.9934969	< 0.0001

5 Results and Discussion

This section presents and discusses the experimental results obtained in this study. Section 5.1 provides an evaluation of the measure of saturation ϵ_B, proposed in Sect. 3. The experiments in this section were performed with no saturation control measure. Section 5.2 establishes the baseline performance and behavior of the activation functions. No saturation control measure was used in these experiments so that any differences in behavior between functions could be identified. After establishing the baseline behavior of each function an evaluation each of each appraoch for controlling saturation is performed.

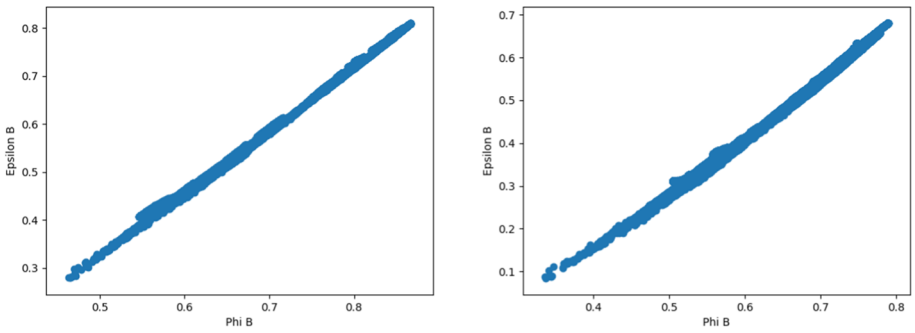
5.1 Analysis of ϵ_B

This section establishes the validity of ϵ_B, the measure of saturation proposed in Sect. 3. To establish the validity of ϵ_B, a comparison of activation function saturation as measured by ϵ_B and φ_B is performed. A visual inspection of the distribution of the output of hidden nodes for different activation functions and problems using histograms is also performed. This sort of visual inspection is commonly performed in the literature [18,38,40].

Table 2 provides correlation coefficients and p values from correlation tests between ϵ_B and φ_B across all datasets for the logistic, tanh, LTanh, and Elliot activation functions. These experiments were performed using the optimized algorithm parameters as described in Sect. 4.2.1. No saturation control measures were used during these tests.

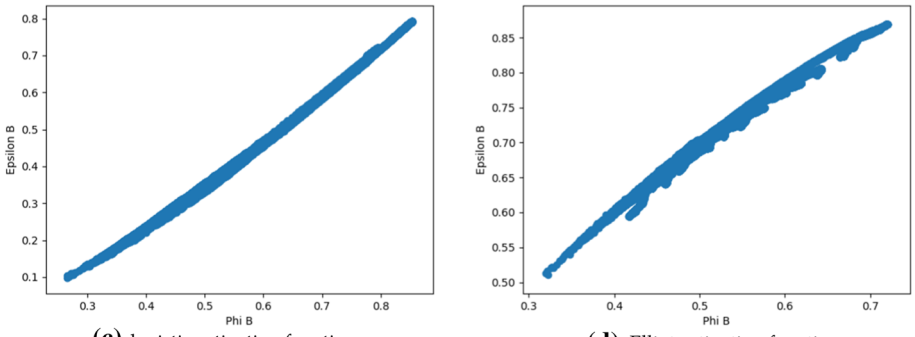
From Table 2 it can be seen that there is a highly significant, strong positive correlation between ϵ_B and φ_B. Figure 3 contains plots with ϵ_B on the x-axis and φ_B on the y-axis for these four activation functions. Figure 3 shows that the relationship between ϵ_B and φ_B is highly linear. This linearity suggests that the two measures tend to differ by an approximately constant factor. An approximately constant difference suggests that if both measures are used to rank algorithms with respect to saturation, the rankings will tend to agree. In general, an agreement in ranking between the two measures is the desired result since φ_B has been shown to be a valid measure of saturation [40].

Figure 4 presents critical difference plots ranking the logistic, tanh, Ltanh, and Elliot activation functions with respect the φ_B and ϵ_B. As is expected, it can be seen that the two measures of saturation tend to agree. Both measures indicate no significant difference in saturation between Ltanh, tanh, and the logistic activation functions, and both measures rank these three functions in the same order. However, φ_B indicates that Elliot saturates significantly less than tanh, and ϵ_B indicates that Elliot saturates significantly more than both Ltanh, and the logistic function. φ_B fails to account for the fact that the gradient of the Elliot function becomes shallow relatively far from the functions asymptotes.



(a) tanh activation function.

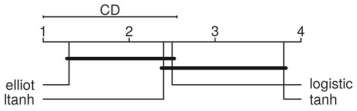
(b) Ltanh activation function.



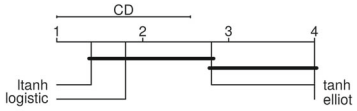
(c) logistic activation function.

(d) Elliot activation function.

Fig. 3 Scatter plots with φ_B on the x axis and ϵ_B on the y axis, showing saturation across all problems



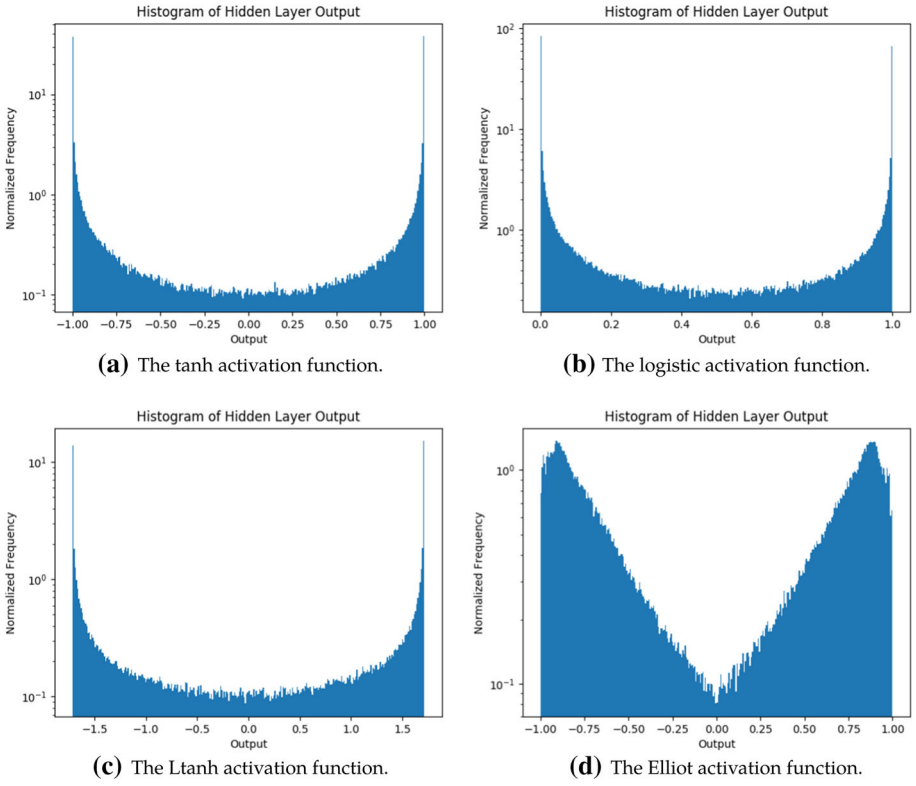
(a) A critical difference plot ranking the logistic, tanh, Ltanh, and Elliot activation functions with respect to φ_B over all datasets.



(b) A critical difference plot ranking the logistic, tanh, Ltanh, and Elliot activation functions with respect to ε_B over all datasets.

Fig. 4 Rankings of the logistic, tanh, Ltanh, and Elliot activation functions with respect to saturation

Figure 5 provides histograms of hidden layer output for each of these four activation functions on the Wine dataset. From Fig. 5 it can be seen that all four functions became saturated, and at first glance, it appears that the Elliot function saturated less than the other three. A similar trend occurs with the other datasets used in this study. However, the gradients of these functions must be considered. As a node in a network becomes saturated, the gradient of the function becomes extremely shallow, and the weights going into that node can become arbitrarily large. This can result in divergence of the swarm, and in effect leaves particles on a large mostly flat plateau from which they cannot escape. As can be seen in Fig. 1, the gradient of the Elliot function becomes shallow relatively early, and relatively large inputs are required to cause the Elliot function to output values close to its boundaries. For these reasons, the difference between the output of the Elliot function, and the closest asymptote of the Elliot function does not provide a reliable measure of saturation.



(a) The tanh activation function. **(b)** The logistic activation function.

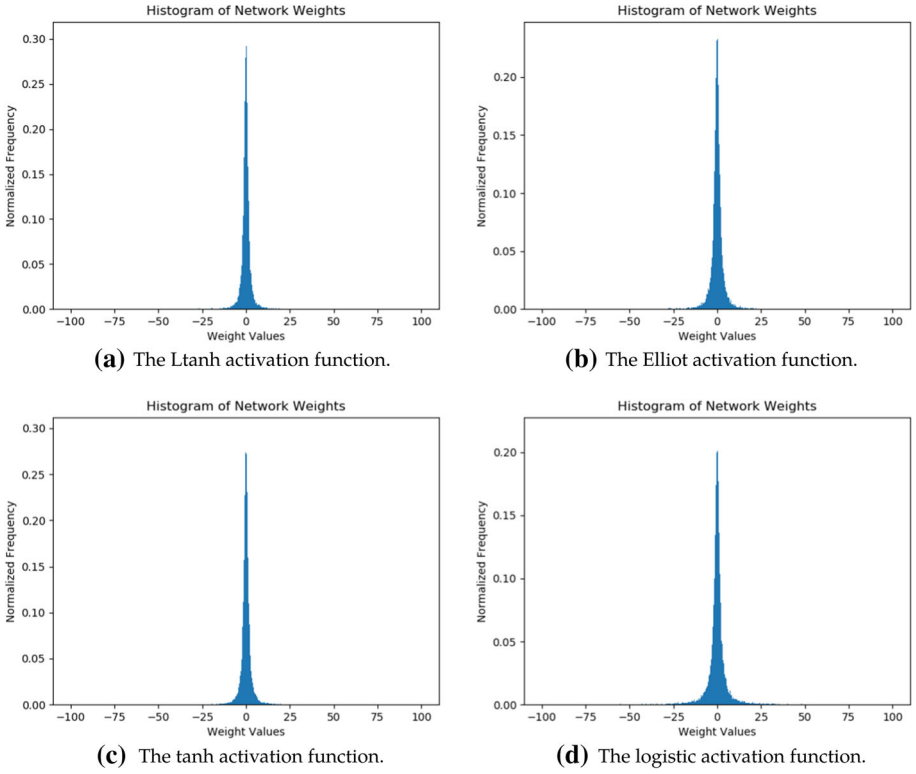
(c) The Ltanh activation function. **(d)** The Elliot activation function.

Fig. 5 Hidden node output distributions for the Wine dataset

Figure 6 presents histograms of network weights after training on the Wine dataset with the Elliot, logistic, tanh, and Ltanh activation functions. From Fig. 6 it can be seen that all functions result in at least some large weights. However, the Elliot function has a greater proportion of large weights than the Ltanh activation. This is worth noting since saturation is likely to produce larger weights in the network and ϵ_B indicates that Elliot has saturated more than Ltanh. Interestingly, the logistic function has also produced a relatively large number of large weights. This is likely due to the shallow gradient of the logistic function. Although the logistic function has a shallow gradient relative to tanh and Ltanh, it approaches its boundaries much more quickly than Elliot. This can be seen in Fig. 7. As a result, both ϵ_B and φ_B provide a similar evaluation of the saturation of the logistic function.

5.2 Analysis of Saturation Behavior

This section examines each activation function to evaluate network saturation and PSO behavior when that function is used. It was found that behavior varied considerably for different datasets. In order to illustrate the range of behaviors, plots of E_T, E_V, ρ, ϵ_B, and diversity over 1000 iterations are presented for the linear, ReLU and logistic functions for two datasets: One dataset where the function exhibited high saturation, and one dataset where the function exhibited low saturation. In order to display all five of these measures within the same plot, some normalization was necessary. Both diversity and ρ have been min-max normalized



(a) The Ltanh activation function. **(b)** The Elliot activation function.

(c) The tanh activation function. **(d)** The logistic activation function.

Fig. 6 Network weight distributions for the Wine dataset

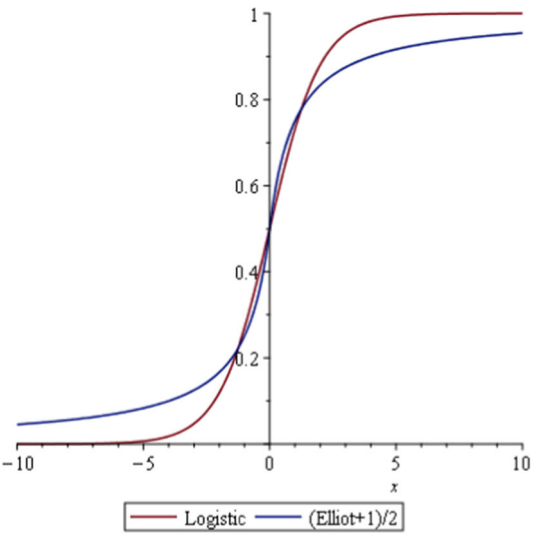
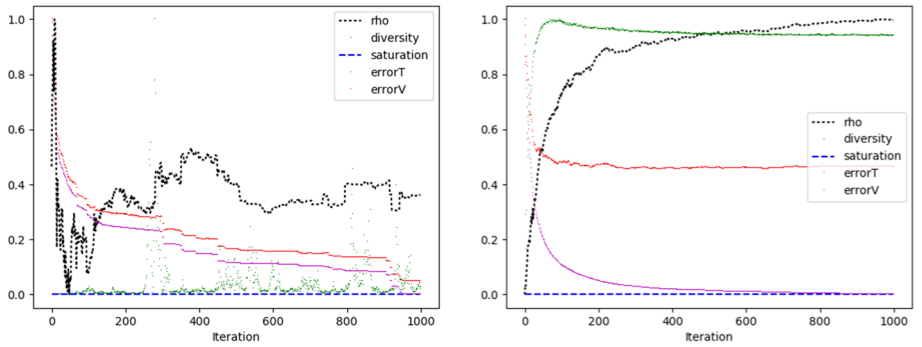


Fig. 7 A plot of the Elliot function and the logistic function



(a) The Henon map dataset. **(b)** The Diabetes dataset.

Fig. 8 Performance and behavior of the linear activation function over 1000 iterations

so that the behavior of these values over the run can be observed. In order to more easily examine overfitting throughout a run, ρ is calculated as $\frac{E_V}{E_T}$ for the presented plots. Training error and validation error have been min-max normalized together. The minimum used is the minimum training or validation error observed, and the maximum is the maximum training or validation error observed. As a result, the included plots correctly reflect the difference in training and validation error. Finally, ϵ_B has not been normalized since it is already in the range [0, 1]. The distribution of hidden node outputs is also examined.

Figure 8 presents E_T, E_V, ϵ_B, ρ, and diversity of PSO over 1000 iterations when the linear activation function was used on the Henon map dataset and the Diabetes dataset. From Fig. 8a it can be seen that diversity varied greatly throughout the run. On this dataset diversity became large early in the run and the swarm failed to converge. The magnitude and standard deviation of diversity at the final iteration are given in Table 3. Although the swarm failed to converge, it was able to continually improve both training, and validation error throughout the run, and appears to have avoided overfitting. From Table 3 it can be seen that ρ for the linear function on the Henon map dataset is less than 1. Figure 8b shows a case where diversity reached is maximum early in the run and remained relatively constant. On this dataset, the magnitude of diversity was both low and consistent compared to diversity on the Henon map dataset, as can be seen in Table 3. In further contrast to the Henon map dataset, Fig. 8b shows that overfitting occurred. It can clearly be seen that validation error reach its minimum early in the run, while training error continued to improve.

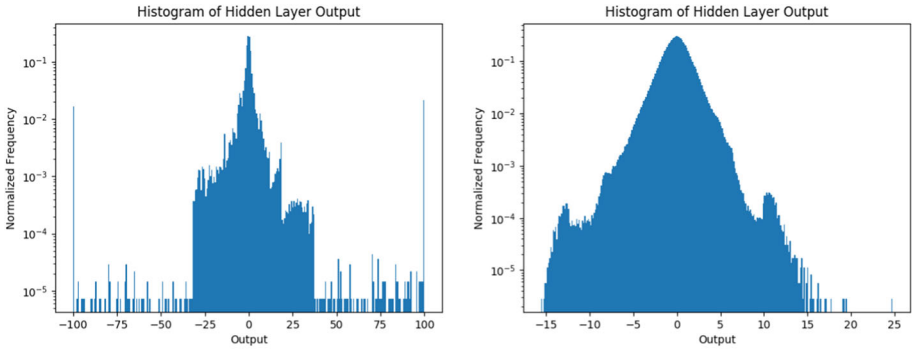
Figure 9 presents the distribution of hidden node outputs for the Henon map and Diabetes datasets when the linear activation function is used. From Fig. 9 it can be seen that hidden node output was centered around 0 for both problems. However, the magnitude of hidden node output was larger on the Henon map dataset.

Tables 4, 5, and 6 present the performance of the ReLU, leaky ReLU, and softplus activation functions over all datasets. Figure 10 presents E_T, E_V, ϵ_B, ρ, and diversity of PSO over 1000 iterations when the ReLU activation function was used with the Wine dataset and the logistic map dataset. Figure 10a presents an example where the ReLU function produced relatively low values for ϵ_B, and Fig. 10b presents an example of relatively high values of ϵ_B. As can be seen from Fig. 10a and b the ReLU activation function produced similar values of ϵ_B across all datasets. However, considerable differences in behavior do exists. For example, the diversity is more than two orders of magnitude larger for the Logistic map dataset than than the Wine dataset. Despite this divergent behavior, the error rates found in Table 4 indi-

Table 3 Performance of the linear activation function on all datasets

	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1415 (0.0202)	0.7121 (0.1269)	0.8491 (0.0916)	0.0000 (0.0000)	37.49 (4.683)
Sunspot	0.0303 (0.0001)	0.0322 (0.0005)	0.0301 (0.0001)	0.0000 (0.0000)	11.12 (2.835)
Logistic map	0.4261 (0.0411)	0.4319 (0.0413)	0.4481 (0.0432)	0.0000 (0.0000)	32807 (62093)
Mackey-Glass	0.0007 (0.0000)	0.0007 (0.0000)	0.0009 (0.0000)	0.0000 (0.0000)	8.761 (0.5880)
Glass	0.7525 (0.0339)	1.1194 (0.0763)	1.1233 (0.0710)	0.0000 (0.0000)	201.4 (243.2)
TS5	0.0053 (0.0000)	0.0055 (0.0001)	0.0056 (0.0000)	0.0000 (0.0000)	14.93 (12.99)
Diabetes	4.0034 (0.0491)	5.1966 (0.2480)	4.4202 (0.1485)	0.0000 (0.0000)	20.34 (2.607)
Henon map	0.2894 (0.0171)	0.2909 (0.0175)	0.2689 (0.0158)	0.0000 (0.0000)	5568 (9959)
Wine	0.0014 (0.0039)	0.3147 (0.1180)	0.0793 (0.0527)	0.0000 (0.0000)	137.2 (85.16)
Iris	0.0727 (0.0026)	0.1403 (0.0379)	0.0484 (0.0085)	0.0000 (0.0000)	105.5 (41.84)

Standard deviation is given in parentheses



(a) Hidden node outputs on the Henon map dataset.

(b) Hidden node outputs on the Diabetes dataset.

Fig. 9 Hidden node output distributions for the linear activation function

cate that ReLU performed competitively on the logistic map dataset. The leaky ReLU and softplus activation functions behaved similar to the ReLU function, exhibiting similar levels of saturation across all datasets. The leaky ReLU activation function promoted convergent behavior relative to the ReLU function.

Note that, in the case of the leaky ReLU activation function, ϵ_B does not measure saturation since leaky ReLU has no asymptotes and cannot saturate. For this function, ϵ_B indicates the magnitude of the gradients of the hidden nodes. A high value of ϵ_B indicates relatively shallow gradients, meaning that a relatively high proportion of net inputs to hidden nodes were below 0. A low value of ϵ_B indicates relatively large gradients, meaning that a relatively high proportion of net inputs to hidden nodes were above 0.

Figure 11 presents the distribution of hidden node outputs for the Wine and logistic map datasets when the ReLU activation function is used. From Fig. 11 it can be seen that the majority of hidden node outputs are 0 or close to 0 for both datasets. The hidden node outputs followed a similar pattern when the leaky ReLU and softplus activation functions were used.

Tables 7, 8, 9 and 10 present the performance of the logistic, tanh, LTanh, and Elliot activation functions across all datasets. Figure 12 presents E_T, E_V, ϵ_B, ρ, and diversity

Table 4 Performance of the ReLU activation function on all datasets

	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1696 (0.0253)	0.7585 (0.1567)	0.8561 (0.1037)	0.5243 (0.0500)	147.7 (46.93)
Sunspot	0.0051 (0.0003)	0.0066 (0.0005)	0.0059 (0.0003)	0.5258 (0.0443)	9.289 (5.310)
Logistic map	0.0026 (0.0008)	0.0029 (0.0009)	0.0028 (0.0008)	0.5526 (0.0375)	2.6e4 (4.8e4)
Mackey-Glass	0.0002 (0.0000)	0.0002 (0.0003)	0.0002 (0.0001)	0.5048 (0.0591)	265.6 (1053)
Glass	0.5924 (0.0361)	1.2267 (0.1086)	1.1958 (0.1195)	0.5101 (0.0442)	599114 (2.2e6)
TS5	0.0015 (0.0005)	617e15 (332e16)	953e10 (513e11)	0.4066 (0.0965)	136e5 (557e5)
Diabetes	3.9041 (0.0809)	5.1968 (0.2116)	4.4646 (0.1513)	0.5116 (0.0353)	29.34 (4.187)
Henon map	0.0005 (0.0002)	0.0005 (0.0003)	0.0005 (0.0003)	0.5346 (0.0429)	1.0e11 (3.1e11)
Wine	0.0218 (0.1128)	0.4263 (0.1740)	0.2065 (0.1793)	0.4982 (0.0424)	104.1 (58.88)
Iris	0.0283 (0.0070)	0.5664 (0.2049)	0.1784 (0.1237)	0.5096 (0.0667)	2.5e4 (7.9e4)

Standard deviation is given in parentheses

Table 5 Performance of the leaky ReLU activation function on all datasets

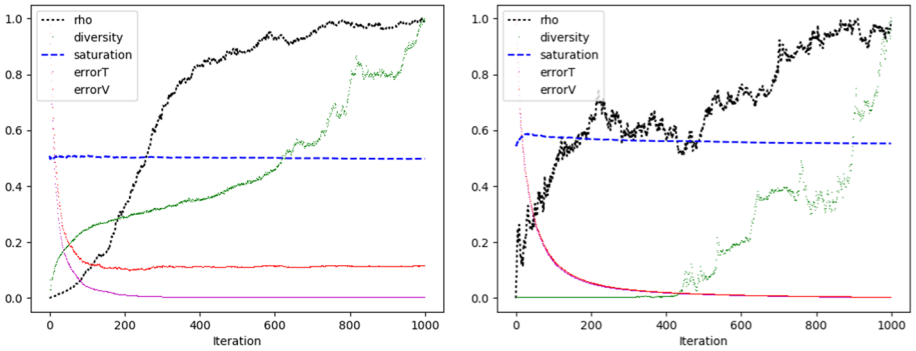
	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1210 (0.0231)	0.7719 (0.1448)	0.9481 (0.1053)	0.5175 (0.0651)	57.19 (18.38)
Sunspot	0.0191 (0.0011)	0.0282 (0.0055)	0.0260 (0.0041)	0.4968 (0.0456)	32.53 (17.44)
Logistic map	0.0075 (0.0026)	0.0080 (0.0027)	0.0079 (0.0027)	0.6606 (0.0478)	29.07 (11.17)
Mackey-Glass	0.0006 (0.0000)	0.0007 (0.0000)	0.0009 (0.0000)	0.5079 (0.0501)	12.00 (2.972)
Glass	0.6112 (0.0409)	1.2303 (0.0913)	1.1859 (0.1132)	0.4832 (0.0411)	197.7 (75.41)
TS5	0.0052 (0.0001)	0.0301 (0.0701)	0.0815 (0.2955)	0.3570 (0.0980)	40.09 (32.69)
Diabetes	3.8793 (0.0729)	5.2709 (0.2765)	4.5720 (0.1508)	0.4919 (0.0443)	35.92 (9.25)
Henon map	0.0018 (0.0007)	0.0019 (0.0007)	0.0020 (0.0007)	0.5258 (0.0506)	35.47 (15.06)
Wine	0.0227 (0.1090)	0.5164 (0.1785)	0.2433 (0.0935)	0.5362 (0.0682)	169.88 (99.79)
Iris	0.0366 (0.0066)	0.3323 (0.1564)	0.1329 (0.0649)	0.5322 (0.0714)	132.1 (88.36)

Standard deviation is given in parentheses

Table 6 Performance of the softplus activation function on all datasets

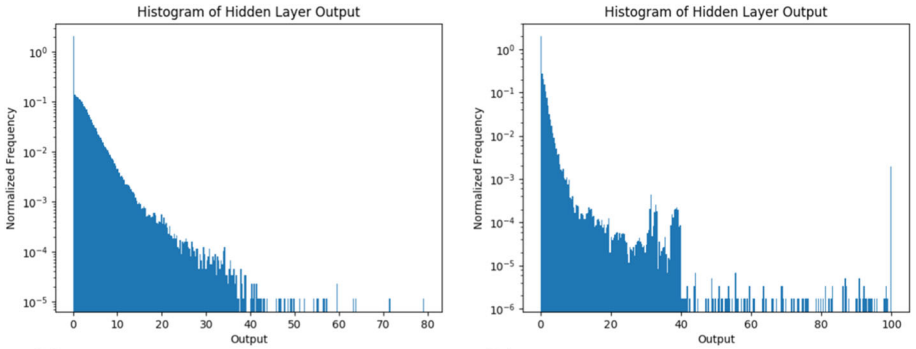
	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1323 (0.0273)	0.7704 (0.1441)	0.9380 (0.0988)	0.5072 (0.0247)	66.34 (9.218)
Sunspot	0.0049 (0.0001)	6.2353 (33.5449)	5.9801 (32.1726)	0.5530 (0.0301)	7.6e5 (3.9e5)
Logistic map	0.0012 (0.0004)	0.0013 (0.0004)	0.0013 (0.0004)	0.5154 (0.0449)	172.2 (595.3)
Mackey-Glass	0.0002 (0.0000)	0.0002 (0.0000)	0.0002 (0.0000)	0.5393 (0.0450)	850.0 (4028)
Glass	0.6323 (0.0427)	1.1644 (0.0892)	1.1506 (0.0835)	0.5169 (0.0473)	5.5e5 (2.9e6)
TS5	0.0012 (0.0000)	0.0014 (0.0003)	0.0016 (0.0005)	0.5607 (0.0475)	12.86 (5.278)
Diabetes	3.9745 (0.0420)	5.1470 (0.1868)	4.3960 (0.0999)	0.5330 (0.0299)	12.01 (2.418)
Henon map	0.0004 (0.0002)	0.0004 (0.0002)	0.0003 (0.0002)	0.4990 (0.0475)	12.09 (2.729)
Wine	0.0025 (0.0116)	0.3156 (0.1212)	0.1800 (0.0823)	0.5318 (0.0407)	7443 (22631)
Iris	0.0476 (0.0054)	0.2501 (0.0942)	0.0644 (0.0282)	0.5586 (0.0597)	144.2 (197.8)

Standard deviation is given in parentheses



(a) The Wine dataset. **(b)** The logistic map dataset.

Fig. 10 Performance and behavior of the ReLU activation function over 1000 iterations



(a) Hidden node outputs on the Wine dataset. **(b)** Hidden node outputs on the logistic map dataset.

Fig. 11 Hidden node output for the ReLU activation function

of PSO over 1000 iterations when the logistic activation function was used on the Sunspot dataset and the Wine dataset. Figure 12a presents an example where the logistic function produced relatively low values for ϵ_B, and Fig. 12b presents an example of relatively high values of ϵ_B. By comparing the magnitude of ϵ_B in Fig. 12a, b it can be seen that ϵ_B varied considerably between datasets with the logistic function. The tanh, Ltanh, and Elliot functions also produced a relatively high variation in ϵ_B between datasets as can be seen in Tables 7, 8, 9, and 10. Although it appears that the logistic function behaved similarly in Fig. 12a, b, the magnitudes of diversity and ρ are considerably different.

Figure 13 presents the distribution of hidden node outputs for the Sunspot and Wine datasets when the logistic activation function is used. From Fig. 13 it can be seen that saturation occurred for both datasets, but considerably more outputs were close to the boundaries of the logistic function for the Wine dataset. The tanh and Elliot functions also exhibited relatively high saturation in all datasets. In their associated histograms the majority of hidden node outputs were close to the function boundaries. In contrast, the Ltanh activation function was relatively resistant to saturation. For some datasets, Diabetes for example, the majority of hidden node outputs were centered on zero.

Comparing the rows from any of Tables 3, 4, 5, 6, 7, 8, 9, or 10 make it clear that even a single activation function allows for noticeable variation in swarm behavior. Levels of

Table 7 Performance of the logistic activation function on all datasets

	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1374 (0.0243)	0.7535 (0.1903)	0.9458 (0.1108)	0.4002 (0.0423)	40.4779 (4.7865)
Sunspot	0.0046 (0.0001)	0.0060 (0.0004)	0.0056 (0.0004)	0.3589 (0.0393)	460.3 (1708)
Logistic map	0.0009 (0.0004)	0.0009 (0.0005)	0.0009 (0.0004)	0.6351 (0.0641)	9946 (31332)
Mackey-Glass	0.0002 (0.0000)	0.0002 (0.0000)	0.0002 (0.0000)	0.3509 (0.0538)	61.73 (111.1)
Glass	0.6504 (0.0315)	1.2020 (0.0822)	1.1837 (0.0935)	0.7010 (0.0433)	25016 (119071)
TS5	0.0012 (0.0000)	0.0013 (0.0000)	0.0015 (0.0001)	0.3829 (0.0405)	22.81 (17.24)
Diabetes	3.9774 (0.0728)	5.2218 (0.2609)	4.5328 (0.1499)	0.4084 (0.0589)	18.82 (3.988)
Henon map	0.0003 (0.0001)	0.0003 (0.0002)	0.0003 (0.0001)	0.5051 (0.0494)	40.75 (64.82)
Wine	0.0018 (0.0018)	0.4248 (0.2059)	0.2036 (0.0872)	0.7842 (0.0369)	1096 (1008)
Iris	0.0354 (0.0055)	0.4366 (0.1849)	0.1141 (0.0699)	0.6381 (0.0465)	278.1 (352.8)

Standard deviation is given in parentheses

Table 8 Performance of the tanh activation function on all datasets

	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1432 (0.0286)	0.8321 (0.1788)	0.9099 (0.1114)	0.5403 (0.0441)	28.2641 (4.0755)
Sunspot	0.0180 (0.0004)	0.0253 (0.0068)	0.0221 (0.0016)	0.5001 (0.0459)	706.1 (2194)
Logistic map	0.0035 (0.0013)	0.0038 (0.0013)	0.0037 (0.0013)	0.7065 (0.0427)	24817 (113557)
Mackey-Glass	0.0007 (0.0000)	0.0007 (0.0001)	0.0009 (0.0001)	0.4370 (0.0478)	11.52 (12.58)
Glass	0.6405 (0.0383)	1.1966 (0.1116)	1.1488 (0.0746)	0.7780 (0.0354)	16959 (74275)
TS5	0.0050 (0.0001)	0.0055 (0.0001)	0.0061 (0.0002)	0.5382 (0.0472)	19.14 (22.08)
Diabetes	3.8215 (0.0737)	5.2699 (0.2012)	4.4750 (0.1437)	0.6137 (0.0409)	30.32 (9.982)
Henon map	0.0011 (0.0004)	0.0012 (0.0004)	0.0011 (0.0004)	0.5512 (0.0319)	550.4 (1848)
Wine	0.0065 (0.0178)	0.4392 (0.1588)	0.2077 (0.0939)	0.8023 (0.0303)	1481 (3491)
Iris	0.0412 (0.0037)	0.2909 (0.1203)	0.0711 (0.0392)	0.4712 (0.0477)	10.36 (1.751)

Standard deviation is given in parentheses

Table 9 Performance of the Ltanh activation function on all datasets

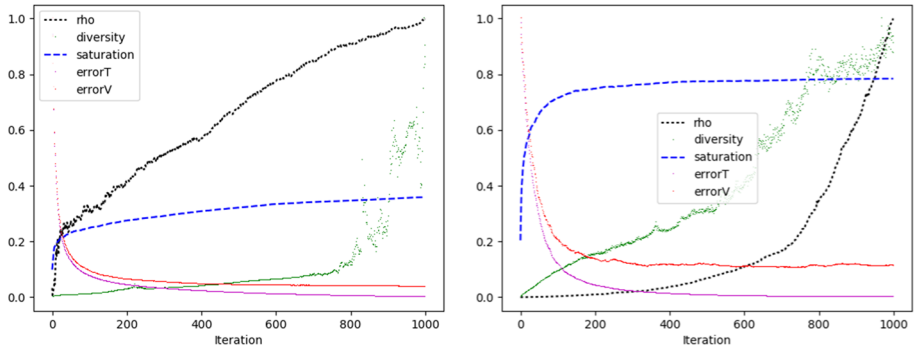
	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1796 (0.0362)	0.7991 (0.1616)	0.9365 (0.0931)	0.3727 (0.0600)	22.95 (3.716)
Sunspot	0.0527 (0.0013)	0.0704 (0.0042)	0.0686 (0.0227)	0.3950 (0.0375)	455.1 (796.1)
Logistic map	0.0130 (0.0053)	0.0141 (0.0062)	0.0136 (0.0059)	0.5733 (0.0485)	11497 (46636)
Mackey-Glass	0.0019 (0.0001)	0.0020 (0.0001)	0.0025 (0.0001)	0.3267 (0.0556)	23.57 (31.85)
Glass	0.6655 (0.0428)	1.1856 (0.0868)	1.1712 (0.0874)	0.5327 (0.0568)	532.1 (1431)
TS5	0.0144 (0.0001)	0.0157 (0.0004)	0.0178 (0.0019)	0.3849 (0.0569)	32.24 (42.13)
Diabetes	3.9699 (0.0560)	5.2417 (0.2233)	4.4260 (0.1643)	0.2376 (0.0381)	14.25 (1.514)
Henon map	0.0038 (0.0011)	0.0039 (0.0011)	0.0038 (0.0011)	0.4516 (0.0384)	148.7 (502.5)
Wine	0.0040 (0.0031)	0.4115 (0.1350)	0.1706 (0.0654)	0.6579 (0.0469)	138.7 (84.50)
Iris	0.0627 (0.2248)	0.7161 (0.3272)	0.2753 (0.3543)	0.6574 (0.0492)	266,354 (1,041,377)

Standard deviation is given in parentheses

Table 10 Performance of the Elliot activation function on all datasets

	E_T	E_V	E_G	ϵ_B	Diversity
BRCA	0.1428 (0.0273)	0.8981 (0.1703)	0.9637 (0.1365)	0.8071 (0.0254)	84.14 (240.3)
Sunspot	0.0171 (0.0005)	0.0240 (0.0013)	0.0217 (0.0007)	0.6558 (0.0233)	172.3 (364.2)
Logistic map	0.0064 (0.0028)	0.0069 (0.0027)	0.0067 (0.0028)	0.7713 (0.0248)	22548 (115248)
Mackey-Glass	0.0008 (0.0001)	0.0008 (0.0001)	0.0010 (0.0001)	0.6332 (0.0213)	12.80 (3.045)
Glass	0.6250 (0.0386)	1.1841 (0.0853)	1.1367 (0.0913)	0.8265 (0.0163)	1487 (3925)
TS5	0.0050 (0.0001)	0.0055 (0.0002)	0.0067 (0.0019)	0.6591 (0.0433)	30.26 (47.77)
Diabetes	3.7890 (0.0890)	5.2639 (0.2527)	4.5535 (0.1559)	0.7457 (0.0269)	35.98 (9.322)
Henon map	0.0013 (0.0006)	0.0014 (0.0007)	0.0014 (0.0006)	0.6936 (0.0300)	43.82 (66.71)
Wine	0.0013 (0.0040)	0.3974 (0.2061)	0.1891 (0.1035)	0.8631 (0.0191)	450.7 (689.7)
Iris	0.0335 (0.0067)	0.3930 (0.1550)	0.0918 (0.0613)	0.7281 (0.0354)	40.45 (34.17)

Standard deviation is given in parentheses

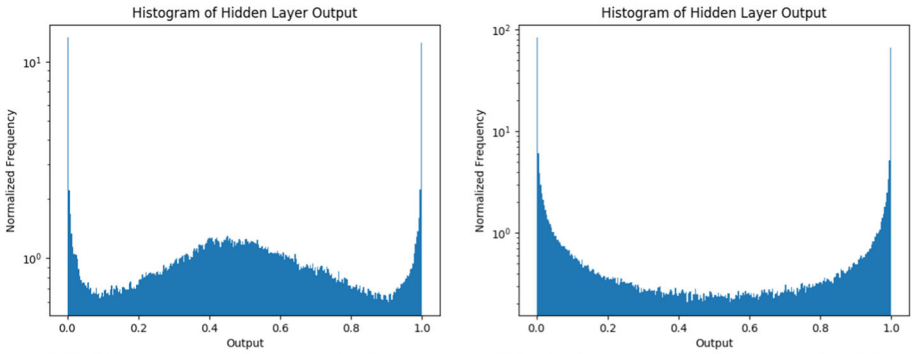


(a) The Sunspot dataset. **(b)** The Wine dataset.

Fig. 12 Performance and behavior of the logistic activation function over 1000 iterations

saturation and diversity vary considerably between datasets, as does performance. Since ϵ_B can vary considerably for the same activation function on different datasets, the magnitude of gradients within the network must vary. This reinforces the need to optimized hyper-parameters for both the dataset and architectural features of the network, like activation functions used. The activation function gradients provide an indication of how much effect changing a weight will have on node output. It is possible that the magnitude of gradients can be related to desirable particle behavior. Since ϵ_B provides a measure of the magnitude of gradients within the network perhaps algorithm hyper-parameters can be adapted at run time based on the value of ϵ_B. For example, it may be reasonable to reduce c_1, c_2, or ω when increasing saturation is detected. This may prevent the magnitude of a particle's position from growing too quickly. Similarly, if weight decay is used then perhaps λ could be increased when unacceptably high levels of saturation are detected.

Finally, the effects of input data scaling, weight decay, both methods of velocity clamping, search space boundaries, and the initialization range on swarm performance and behavior are examined and the most effective measure for controlling saturation is identified. In order to analyze the effects of each method for controlling saturation, Friedman tests were performed, and critical difference plots were generated for each activation function. This produced rank-



(a) Hidden node outputs on the Sunspot dataset.

(b) Hidden node outputs on the Wine dataset.

Fig. 13 Hidden node output distributions for the logistic activation function

ings of the saturation control measures with respect to E_T, E_G, ϵ_B, and ρ. The main results of this analysis are summarized here. A full listing of the results and rankings can be found in Online Resource 1.

This study found that normalized velocity clamping was the overall most effective measure for controlling saturation. For all activation functions normalized velocity clamping produced a significant reduction in saturation relative to vanilla PSO. Normalized velocity clamping also tended to produce low values of ρ, indicating low overfitting. However, ρ was not reduced significantly in any case. Although overfitting was low, for the majority of functions normalized velocity clamping produced the worst training and generalization performance of any saturation control measure. In all cases normalized velocity clamping performed among the worst. The main motivation behind velocity clamping is to prevent step sizes from exploding and reduce the rate at which weights can grow. This results in a more detailed sampling of the search space. In a bounded search space this makes sense. A particle with too much velocity can simply leave the feasible space, resulting in an invalid solution. However, in FFNN training the search space is typically unbounded. While too many large weights may lead to saturation and damage performance, a few large weights may be required to represent the target function. It is possible that in this unbounded space normalized velocity clamping simply prevented particles from fully exploiting valleys in the search space.

Component wise velocity clamping also typically produce low levels of saturation as well. However, it was also generally accompanied by competitive training and generalization performance. This difference in performance may provide further evidence that normalized clamping prevented particles from exploiting fruitful regions. Normalized velocity clamping limits the total magnitude of the velocity vector and component-wise only limits the magnitude of components. As a result normalized clamping could produce relatively smaller step sizes when there are correlations between weights, leading to less ability to exploit relative to component-wise velocity clamping. Another factor which could explain this difference in performance is the fact that component-wise clamping does not preserve the velocity vector's original direction. When a component with large magnitude is clamped, the direction of the particle is changed. Since a local minimum could lead to increased velocity it is possible that this change in direction helps particles escape from local minima with small basins of attraction. In this sort of situation normalized clamping would be more likely to leave a particle trapped in the local minimum.

Weight decay did not perform significantly better or worse than any other approach with respect to controlling saturation. Although it only significantly reduce saturation for leaky ReLU, no significant difference could be detected between weight decay and normalized clamping for any function. Even though the differences were not significant, weight decay was typically near the top of the pack with respect to reducing saturation and generally produced competitive training and generalization error. Finally, the scaling of input data consistently performed poorly as a method for controlling saturation. It did not significantly reduce saturation for any function and in the case of the logistic function it actually increased saturation. Along with scaling of input data, optimizing the initialization range or the search space bounds consistently performed poorly as a saturation control measure. In no case did any of these three methods significantly reduce saturation.

6 Conclusions

This study examined a set of common activation functions used in FFNNs and empirically evaluated network saturation and PSO swarm behavior for each function. The considered activation functions include the linear, logistic, tanh, Ltanh, Elliot, ReLU, leaky ReLU, and softplus functions. This work placed a particular focus on activation function saturation. Activation function saturation is often discussed in the literature and often associated with poor performance and swarm divergence. In order to objectively compare activation function saturation between all examined activation functions, a new measure of saturation has been proposed and validated. After establishing a baseline for the activation functions six measures for controlling saturation were examined. These measures include scaling of the input data, tuning particle initialization range, tuning the search space boundaries, normalized and component-wise velocity clamping, and weight decay. Each approach was evaluated for each activation function to determine its effectiveness at controlling activation function saturation.

Section 5.1 analyzed the measure of saturation proposed in Sect. 3. A highly significant, highly linear relationship between ϵ_B and φ_B was identified. This suggests that given the same algorithms and the same data, the two measures will provide the same ranking of algorithms with respect to saturation. Friedman tests and critical difference plots were used to rank tanh, Ltanh, logistic and Elliot with respect to ϵ_B and φ_B over all datasets. It was shown that ϵ_B produced a similar ranking of algorithms with respect to that of φ_B and that ϵ_B addresses issues identified with φ_B's evaluation of the Elliot function.

Section 5.2 analyzed the saturation behavior of each activation function and evaluated each approach to controlling saturation. Although the behavior of all functions varied depending on the dataset, an evaluation of the linear activation function suggests that when no saturation is possible, both hidden node output and the weights and biases of the network will have a distribution centered on zero. All activation functions produced a similar distribution of weights and biases. However, there was some variation in the distribution of hidden node outputs between activation functions. In general, all functions with at least one bound produced a distribution of hidden node outputs consistent with saturation for the majority of datasets. That is, a large number of outputs were close to the function's bound(s). All functions also regularly exhibited divergent behavior and overfitting. The Ltanh, logistic, leaky ReLU, and ReLU activation functions were found to produce the lowest values of ϵ_B. These values were significantly less than those for the Elliot activation function. It should be noted that leaky ReLU and ReLU are special cases with respect to saturation. For these two functions the gradient of the function is either one of two values, it does not slowly approach zero as the

magnitude of input grows. However, it is possible for the magnitude of weights and biases in the network to grow arbitrarily with little or no effect on node output, so these two functions can exhibit behavior similar to typical saturation.

Finally, the evaluation of each measure for controlling saturation revealed that both methods of velocity clamping and weight decay were effective at controlling saturation. However, normalized velocity clamping consistently produced poor performance in the performed simulations, while component-wise clamping and weight decay produced competitive performance. Two factors which may contribute to the difference in performance between normalized velocity clamping and component-wise velocity clamping were identified. The first factor related to the unbounded nature of the neural network error surface. In certain cases when many weights and biases are changing simultaneously normalized clamping could produce smaller step sizes than component-wise clamping. This may prevent particles from fully exploiting fruitful regions of the error surface. Second, since component-wise clamping does not preserve a particle's original direction, it introduces variation into the particle trajectory when the magnitude of a component in the velocity vector becomes large. It is possible that this variation in trajectory helps particles to escape local minima with small basins of attraction. Overall, component-wise velocity clamping and weight decay were found to be the most effective measures for controlling activation function saturation.

Future work involves expanding the suite of benchmark problems to include both large scale and dynamic problems. Such problems typically involve operating on a stream of data, rather than a single set of known examples. That is, the batch of data on which the network is training changes with time. The effects of a shifting dataset on activation function saturation should be examined. When training on a stream of data, optimal values for saturation control measure hyper-parameters may change over time. Strategies to dynamically adapt parameter values should be examined. The use of adaptive activation functions can also be examined.

References

1. Al Hazza MH, Adesta EY (2013) Investigation of the effect of cutting speed on the surface roughness parameters in CNC end milling using artificial neural network. In: IOP conference series: materials science and engineering, vol 53, IOP Publishing. https://doi.org/10.1088/1757-899X/53/1/012089
2. Bishop C (1995) Neural networks for pattern recognition. Oxford University Press, New York
3. Carvalho M, Ludermir R (2006) Particle swarm optimization of feed-forward neural networks with weight decay. In: Proceedings of the international conference on 3D digital imaging and modeling, pp 1–5
4. Center NGD (2019) Boulder sunspot number data. https://www.sws.bom.gov.au/Educational/2/3/6. Accessed 16 Mar 2019
5. Dahl G, Sainath T, Hinton G (2013) Improving deep neural networks for LVCSR using rectified linear units and dropout. In: Proceedings of the conference on acoustics, speech and signal processing, pp 8609–8613
6. Das M, Dulger L (2009) Signature vecification (SV) toolbox: applications of PSO-NN. Eng Appl Artific Intel 22(4):688–694
7. Dreyfus G (2005) Neural networks: methodology and applications. Springer, Berlin
8. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml. Accessed 16 Mar 2019
9. Dugas C, Bengio Y, Belisle F, Nadeau C, Garcia R (2001) Incorporating second-order functional knowledge for better option pricing. In: Proceedings of the conference on advances in neural information processing systems, pp 472–478
10. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, pp 39–43
11. Eggensperger K, Lindauer M, Hoos H, Hutter F, Leyton-Brown K (2018) Efficient benchmarking of algorithm configurators via model-based surrogates. Mach Learn 107(1):15–41

12. Elliott D (1993) A better activation function for artificial neural networks. Technical report T.R. 93-8, University of Maryland
13. Engelbrecht A (2012) Particle swarm optimization: velocity initialization. In: Proceedings of the congress on evolutionary computation, pp 1–8
14. Engelbrecht A (2013) Particle swarm optimization: global best or local best? In: BRICS congress on computational intelligence and 11th Brazilian congress on computational intelligence. IEEE, pp 124–135
15. Engelbrecht A, Cloete I, Geldenhuys J, Zurada J (1995) Automatic scaling using gamma learning for feedforward neural networks. In: Proceedings of the international workshop on artificial neural networks. Springer, pp 374–381
16. Fisher R (1936) Iris data set. https://archive.ics.uci.edu/ml/datasets/Iris. Accessed 2 Aug 2018
17. Forina M et al (1991) Wine data set. https://archive.ics.uci.edu/ml/datasets/Wine. Accessed 2 Aug 2018
18. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: International conference on artificial intelligence and statistics, pp 249–256
19. Golik P, Doetsch P, Ney H (2013) Cross-entropy vs. squared error training: a theoretical and experimental comparison. Interspeech 13:1756–1760
20. Gudise V, Venayagamoorthy G (2003) Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In: Proceedings of the swarm intelligence symposium, pp 110–117
21. Harrison K (2018) An analysis of parameter control mechanisms for the particle swarm optimization algorithm. Ph.D. thesis, University of Pretoria
22. Helwig S, Wanka R (2007) Particle swarm optimization in high dimensional bounded search spaces. In: Proceedings of the swarm intelligence symposium, pp 198–205
23. Helwig S, Wanka R (2008) Theoretical analysis of initial particle swarm behavior. In: Rudolph G, Jansen T, Beume N, Lucas S, Poloni C (eds) Proceedings of the international conference on parallel problem solving from nature, pp 889–898
24. Hutter F, Hoos H, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Coello CA (ed) Learning and intelligent optimization. Springer, Heidelberg, pp 507–523
25. Hutter F, Lücke J, Schmidt-Thieme L (2015) Beyond manual tuning of hyperparameters. KI-Künstliche Intelligenz 29(4):329–337
26. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of the international conference on neural networks, vol 4, pp 1942–1948
27. Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: Proceedings of the international congress on evolutionary computation, vol 2, pp 1671–1676
28. Lawrence S, Tsoi A, Back A (1996) Function approximation with neural networks and local methods: bias, variance and smoothness. In: Proceedings of the Australian conference on neural networks, australian national university, vol 1621
29. LeCun Y, Bottou L, Orr G, Müller K (2012) Efficient BackProp. Springer, Berlin, pp 9–48
30. Maas A, Hannun A, Ng A (2013) Rectifier nonlinearities improve neural network acoustic models. in: Proceedings of the workshop on deep learning for audio, speech, and language processing, vol 30, pp 3–8
31. Mendes R, Cortez P, Rocha M, Neves J (2002) Particle swarms for feedforward neural network training. In: Proceedings of the international joint conference on neural networks, pp 1895–1899
32. Moody J, Hanson S, Krogh A, Hertz JA (1995) A simple weight decay can improve generalization. Adv Neural Inf Process Syst 4:950–957
33. Oldewage E (2018) The perils of particle swarm optimization in high dimensional problem spaces. Master's thesis, Department of Computer Science, University of Pretoria
34. Olorunda O, Engelbrecht A (2008) Measuring exploration/exploitation in particle swarms using swarm diversity. In: Proceedings of the international congress on evolutionary computation, pp 1128–1134
35. Platt J (1991) A resource-allocating network for function interpolation. Neural Comput 3(2):213–225
36. Prechelt L (1994) PROBEN1—a set of benchmarks and benchmarking rules for neural network training algorithms. https://github.com/jeffheaton/proben1. Accessed 16 Mar 2019
37. Rakitianskaia A, Engelbrecht A (2012) Training feedforward neural networks with dynamic particle swarm optimisation. Int J Uncertain Fuzziness Knowl Based Syst 6(3):233–270
38. Rakitianskaia A, Engelbrecht A (2014) Training high-dimensional neural networks with cooperative particle swarm optimiser. In: Proceedings of the international joint conference on neural networks, pp 4011–4018
39. Rakitianskaia A, Engelbrecht A (2014) Weight regularisation in particle swarm optimisation neural network training. In: Proceedings of the symposium on swarm intelligence, pp 1–8
40. Rakitianskaia A, Engelbrecht A (2015) Measuring saturation in neural networks. In: Proceedings of the symposium series on computational intelligence, pp 1423–1430

41. Rakitianskaia A, Engelbrecht A (2015) Saturation in PSO neural network training: good or evil? In: Proceedings of the international congress on evolutionary computation, pp 125–132
42. Rini DP, Shamsuddin SM, Yuhaniz SS (2011) Particle swarm optimization: technique, system and challenges. Int J Comput Appl ' 14(1):19–27
43. Röbel A (1994) The dynamic pattern selection algorithm: effective training and controlled generalization of backpropagation neural networks. Technical report, Technische Universität Berlin
44. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: Proceedings of the international congress on evolutionary computation, pp 69–73. https://doi.org/10.1109/ICEC.1998.699146
45. Stützle T, López-Ibáñez M (2019) Automated design of metaheuristic algorithms. Springer, Cham, pp 541–579
46. van den Bergh F, Engelbrecht A (2000) Cooperative learning in neural networks using particle swarm optimizers. S Afr Comput J 26:84–90
47. van Wyk A, Engelbrecht A (2010) Overfitting by PSO trained feedforward neural networks. In: Proceedings of the international congress on evolutionary computation, pp 1–8
48. van Wyk A, Engelbrecht A (2016) Analysis of activation functions for particle swarm optimized feedforward neural networks. In: Proceedings of the international congress on evolutionary computation, pp 423–430
49. Volschenk A, Engelbrecht A (2016) An analysis of competitive coevolutionary particle swarm optimizers to train neural network game tree evaluation functions. In: Tan Y, Shi Y, Niu B (eds) Advances in swarm intelligence. Springer, Cham, pp 369–380
50. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioural sciences. Ph.D. thesis, Harvard University
51. Wessels L, Barnard E (1992) Avoiding false local minima by proper initialization of connections. Trans Neural Netw 3(6):899–905
52. Wolberg W (1990) Breast cancer wisconsin (original) data set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29. Accessed 2 Aug 2018
53. Wyk A, Engelbrecht A (2011) Lambda-gamma learning with feedforward neural networks using particle swarm optimization. In: Proceedings of the symposium on swarm intelligence, pp 1–8
54. Xiao X, Wang Z, Li Q, Xia S, Jiang Y (2017) Back-propagation neural network on markov chains from system call sequences: a new approach for detecting android malware with system call sequences. IET Inf Secur 11(1):8–15