Amsterdam University College

Scalability of Particle Swarm Optimization for Artificial Neural Network Training

Capstone

Jonas van der Ham
Amsterdam University College
Jonasvdham@gmail.com
Science major

Supervisor
Dr. Ana Oprescu
University of Amsterdam
A.M.Oprescu@uva.nl

Reader
Dr. Andy Pimentel
University of Amsterdam
A.D.Pimentel@uva.nl

Supervisor
MSc. Dolly Sapra
University of Amsterdam
D.Sapra@uva.nl

Tutor
Dr. Bart Verheggen
Amsterdam University College
B.Verheggen@auc.nl

June 1, 2020
Word count: 6640

University of Amsterdam

VU Vrije Universiteit Amsterdam

Abstract

With the growth of real world optimization problems, the scalability of optimization methods is a relevant area of research. A particularly interesting large scale optimization problem is artificial neural network training. Backpropagation, the standard method for training neural networks, can suffer from slow convergence and convergence to local optima. Particle swarm optimization (PSO) is a metaheuristic which balances exploration and exploitation of the solution space to escape local optima. PSO has been successfully applied for training small neural networks. This research investigates the scalability of backpropagation and PSO as neural network training methods. Both methods are implemented for the training of networks of increasing sizes. It is found that for the smallest neural networks in this study the final accuracy reached by backpropagation and PSO is comparable. For all other networks backpropagation is demonstrated to significantly outperform PSO in terms of both accuracy and training time. It is concluded that the current PSO implementation suffers from early convergence and is not suitable for training large scale neural networks. Backpropagation is deemed the superior training method in terms of accuracy, training time and scalability.

Keywords — Scalability, Artificial Neural Networks, Backpropagation, Particle Swarm Optimization, MNIST

Contents

CHAPTER 1

Introduction

The number of applications of artificial neural networks (ANNs) in society is steadily increasing [1]. Efficient training of these neural networks is essential for technologies like autonomous vehicles, speech recognition and semantic parsing [2, 3]. Historically, ANN training has been done through the backpropagation algorithm. Backpropagation can efficiently exploit the current solution but can converge to local optima and has a slow convergence rate [4, 5].

To solve these problems, metaheuristic approaches have been proposed, many of which balance exploration and exploitation [4, 6, 7]. Particle swarm optimization (PSO) is a swarm based metaheuristic in which multiple particles explore the search space [8]. The communication between particles allows them to break out of local optima and converge to a global optimum. Research has evaluated the promising performance of PSO in various applications [6, 7, 9]. ANN training through PSO (PSO-NN) has outperformed backpropagation in both mean squared error (MSE) and accuracy [4, 9]. However, in these works PSO was only used to train small ANNs for relatively simple applications [10, 11].

With the rising complexity of many real world problems the training of large scale neural networks is most relevant [12, 13]. Therefore, scalability is an important aspect when qualifying PSO for ANN training. Backpropagation has shown strong performance in terms of scalability [14]. PSO has shown weak performance in large scale optimization problems and modifications have been suggested to improve this [13, 15]. Research investigating the scalability of PSO when applied to ANN training has not shown conclusive results [16].

1.1 Problem statement

This research aims to compare the scalability of backpropagation and PSO with respect training ANNs. It is hypothesized that accuracies reached by the two methods are comparable for small networks but that backpropagation quickly outperforms PSO in terms of training time and accuracy with an increase in problem size. Considering the inconclusive results of previous research that has applied PSO to ANN training the following research question is formulated:

> *Is there a difference in the scalability of backpropagation and particle swarm optimization when applied to ANN training?*

1.2 Research methods

To answer the research question neural networks of increasing sizes are trained on handwritten digit classification tasks from MNIST and UCI Machine Learning repository [10, 17]. A popular set of measures from the PASCAL large scale learning challenge is employed to quantify scalability [14, 18, 19].

1.3 Outline

The rest of the paper is structured as follows. Chapter 2 covers the relevant background information. Related work is discussed in chapter 3. Next, chapter 4 presents the methodology. The experimental setup and results are presented in chapter 5 and the discussion and conclusion follow in chapters 6 and 7.

CHAPTER 2

Background

A sizeable amount of research has preceded this study. This chapter establishes scalability in the context of this study and covers the foundations of ANNs, backpropagation and PSO.

2.1 Scalability

With the growth of many real world problems, interest in the optimization of large scale continuous problems has increased [20]. However, many popularly adopted benchmarks for optimization algorithms only have around 10^2 to 10^3 decision variables [11, 13].

As performance of many optimization algorithms deteriorates quickly when the dimensionality of problems increases the evaluation of optimization algorithms in large scale problems is very important [20]. A large scale optimization problem that is particularly relevant is that of training artificial neural networks (section 2.2). Neural network training is a complex continuous optimization task, often requiring the concurrent optimization of thousands of weights.

Various metrics have been proposed to measure algorithmic scalability but there is no standard method in the literature [14]. An issue of Soft Computing on scalability provides a standard set of scalable function optimization problems which are tackled at 50-1000 dimensions [20]. However, considering the size of current real world optimization problems, 1000-dimensional problems do not necessarily classify as large scale. As image classification tasks in neural networks often require high-dimensional optimization, this research optimizes neural networks for the UCI and MNIST handwritten digit classification tasks (section 4.4) [10, 17].

Scalability measures in this study are applied following the PASCAL Large Scale Learning Challenge (section 4.5) for two reasons. Firstly, these measures are designed for neural network training applications and are therefore relevant to the aim of this research [18]. Secondly, previous research has shown that they allow for effective comparions of scalability [14, 19].

2.2 Artificial neural networks

The artificial neural network is a computational model inspired by the human brain. Neural networks are essential to many modern technologies and have long dominated the field of AI research [1, 3]. These networks consist of set of interconnected nodes known as perceptrons. These nodes take one or more real-valued inputs and output a single real value which they can pass on to other nodes [21]. Each perceptron takes an n-dimensional input vector $\vec{x}$ and n-dimensional weight vector $\vec{w}$. The value of that perceptron is the weighted sum of the input $\vec{x}$ with the weights $\vec{w}$. Its output is usually calculated by applying an activation function $f(x)$ (fig. 2.1).

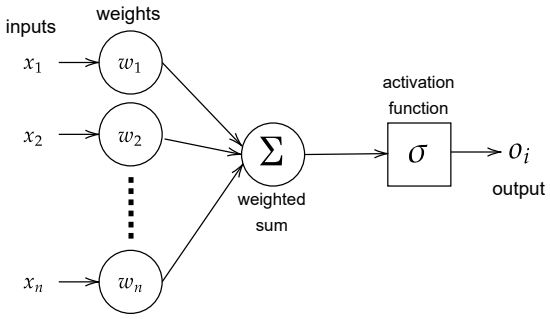

Figure 2.1: The perceptron

The combination of multiple perceptrons in a network structure can be used to solve complex problems (fig. 2.2) [2, 12]. The output of such a network is calculated by propagating the input through the network. The value of each node x in layers after the input layer follows from eq. (2.1). Each o_i is the output value from a node in the previous layer and each w_{ji} is the weight corresponding to that output. Typically each layer will have a bias node b_i connected to all the nodes in the following layer with constant value of 1. The weights of connections to each bias node are optimized during the training process.

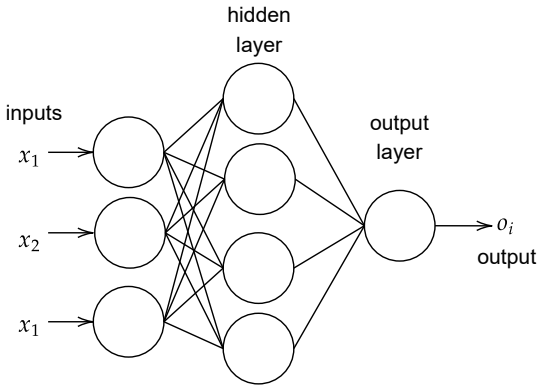

Figure 2.2: A fully connected neural network

$$x_j = \sum_i o_i w_{ji} + b_i \tag{2.1}$$

The activation function $f(x) = o$ used for calculating the output value of each node is classically the sigmoid function, eq. (2.2) [21].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

2.2.1 Network topologies

Through the years of neural network research, many network topologies have been suggested. A fully connected neural network, or multi-layer perceptron, is the most simple neural network topology (fig. 2.2). It consists of a set of hierarchical layers of nodes which are strictly connected to all the nodes in the layers before and after it. Each connection between two nodes represents a weight. The rising complexity of problems sparked the creation of deep neural networks. These networks make use of multiple hidden layers and have become the standard for any non-trivial task [12].

An important problem specific neural network topology is the convolutional neural network (CNN). CNNs were designed for image recognition tasks [22]. In two-dimensional pattern recognition these networks have attained state of the art performance [23]. The convolutional layers included in these networks can recognize corners, shapes, and other features through the feature extraction process.

2.2.2 Neural network optimization

Optimizing a neural network consists of the search for a suitable set of hyper parameters and weights. The tuning of hyper parameters includes the selection of activation functions, network layout and network size. Finding an appropriate set of weights is known as training the neural network. Given a network, these weights will determine the output. As this research aims to evaluate ANN training methods, the hyper parameters are predetermined and not optimized.

Neural network training is usually divided into supervised, unsupervised or hybrid learning paradigms [3]. In supervised learning, each training sample which is propagated through the network has an associated correct answer which the network should output. The weights are directed to give outputs as close to the correct output as possible. Supervised learning problems can be subdivided into regression and classification problems [5]. Regression problems have some continuous valued optimal output associated with the inputs. In classification problems the input is assigned to one or more members of the set of output classes with some probability. Each prediction is a probability that the input is a member of the corresponding class. Unsupervised learning does not have correct outputs associated with training inputs. The network is expected to uncover underlying structures, patterns or correlations in the data [3]. The hybrid learning paradigm employs a combination of both supervised and unsupervised learning.

Early on it was shown that neural network training is an NP-complete problem [24]. Neural network architectures have evolved from using 3 perceptrons [24] to having up to 10^9 free parameters [25]. This evolution has greatly increased the range of applications and performance: from learning the XOR-function, to handwritten digit recognition, to modern image recognition tasks [12, 26, 27]. Synchronously, the growth of neural networks gave rise to the complexity of training them. Considering this trend, efficient ANN training is increasingly important. This is reflected by the large number of studies dedicated to scalability of optimization methods [13–15, 18, 28].

Various methods exist for training neural networks. The two methods applied in this study are explained in the following sections.

2.3 The backpropagation algorithm

Backpropagation, as proposed by Rumelhart, Hinton, and Williams [29], has long been the standard training method for artificial neural networks [12, 21]. The algorithm attempts to minimize the squared error between the desired output values and the network predictions through gradient descent. This is done in two phases: the forward pass and the backward pass [21].

In the forward pass the network is initialized and an input vector is propagated through the network following eq. (2.1) and eq. (2.2). The received output layer is used in the backward pass. The error between target and output values t_k and o_k is calculated for each output node k (eq. (2.3)). This error is propagated back through the network to find the error term δ_h for each hidden unit h (eq. (2.4)).

$$\delta_k = o_k(1 - o_k)(t_k - o_k) \tag{2.3}$$

$$\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{kh}\delta_k \tag{2.4}$$

Through the error term 'blame' is assigned to each node depending on its weight. The weights are adjusted according to learning rate parameter η and their blame for the error term (eq. (2.5), eq. (2.6)). Iteratively this process converges to optimal weights for the network (algorithm 1).

$$w_{ji} = w_{ji} + \Delta w_{ji} \tag{2.5}$$

$$\Delta w_{ji} = \eta \delta_j x_{ji} \tag{2.6}$$

The convergence strategy of backpropagation relies on the gradient of the error or loss function. Its speed of convergence is highly dependent on the choice of the learning rate parameter η [29]. As backpropagation is sensitive to changes in η finding the right parameter value for it can be a difficult task. If the learning rate is too low convergence will be slow but if the learning rate is too high backpropagation may not converge at all [5].

When working with complex optimization problems, those which are not linearly separable, backpropagation can get stuck in local optima [30]. This problem has long been identified and many improvements have been proposed [31–33]. Whilst it does not fully eliminate the problem, this research uses the stochastic gradient descent algorithm in backpropagation which has the possibility of escaping local optima [14]. In stochastic gradient descent, the error function is calculated for each individual sample.

Algorithm 1 Backpropagation learning algorithm [21]

```
Initiate weights w⃗
for x⃗ in training data do
  Forward Pass
    Propagate x⃗ through the network using eq. (2.1) and eq. (2.2)
  Backward Pass
    for each output unit k do
      Calculate the corresponding error term δ_k
    end for
    for each hidden unit h do
      Calculate the corresponding error term δ_h
    end for
    for each w_ji ∈ w⃗ do
      Update weight w_ji
    end for
end for
```

Weight updates are applied iteratively following eq. (2.6) and the average weight updates are expected to approximate the local gradient descent.

2.4 Particle swarm optimization

Particle swarm optimization is a metaheuristic optimization algorithm proposed by Kennedy and Eberhart [8]. This swarm intelligence metaheuristic is inspired by flocking behavior in animals. It uses multiple particles which represent candidate solutions [34]. The swarm balances global exploration of the search space with local exploitation of the current solutions.

Given an n-dimensional problem space, each particle is represented by an n-dimensional vector $\vec{x}$. Each iteration of the algorithm moves particle i with location $\vec{x}_i$ according to its velocity $\vec{v}_i$ (eq. (2.7)), if that movement improves the particle's fitness. The particle's velocity $\vec{v}_i$ determines the rate and direction of change of its position (eq. (2.8)).

$$\vec{x}_i = \vec{x}_i + \vec{v}_i \tag{2.7}$$

$$\vec{v}_i = \omega\vec{v}_i + c_1\vec{r}_1(\vec{x}_{i,best} - \vec{x}_i) + c_2\vec{r}_2(\vec{x}_{g,best} - \vec{x}_i) \tag{2.8}$$

The inertia ω scales the weight of the previous velocity [35]. Constant c_1 determines the weight of the cognitive term, in which vector $(\vec{x}_{i,best} - \vec{x}_i)$ points to the location of best fitness $(\vec{x}_{i,best})$ reached by particle i. Constant c_2 determines the importance of the social term. Vector $(\vec{x}_{g,best} - \vec{x}_i)$ points to the location of best fitness $(\vec{x}_{g,best})$ reached by any particle. Vectors $\vec{r}_1$ and $\vec{r}_2$ with uniformly random entries are used to add noise, with the aim of discovering new candidate solutions. Through a number of iterations the swarm closes in on an optimal solution (algorithm 2).

Algorithm 2 Particle Swarm Optimization

```
Initiate x⃗_i and v⃗_i for each particle i
for number of iterations do
  for each particle i do
    Evaluate fitness and update x⃗_i,best
    if x⃗_i,best > x⃗_g,best then
      Update x⃗_g,best
    end if
    Update x⃗_i and v⃗_i
  end for
end for
Return x⃗_g,best
```

2.4.1 PSO for training neural networks

Because of its simple structure, PSO is easily applicable to neural network training. In this application (PSO-NN), each particle represents a candidate solution to the neural network training problem [16]. The particle is a vector of all the weights and biases in the network. Each particle moves through the n-dimensional problem space where, as usual, the location is updated if the fitness is increased.

2.4.2 PSO variations

Numerous variations of PSO have been proposed with different purposes. An overview of the most important modifications is given here. All modifications applied in this research are explained in chapter 4.

The inertia ω was an early addition to PSO suggested by Shi and Eberhart [35]. It can be used to inhibit the velocity growth, combatting 'velocity explosion' where velocities grow indefinitely [36]. Additionally, the size of inertia influences the balance between exploration ($\omega > 1$) and exploitation ($\omega < 1$) [36]. Inertia was initially proposed as a constant but various dynamic inertia strategies have later been suggested [37]. Linearly decreasing inertia (eq. (2.9)) is a popular strategy for shifting from initial global exploration to eventual local exploitation.

$$\omega_i = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{i_{max}} i \tag{2.9}$$

Another technique for avoiding velocity explosion is the use of a constriction factor [38]. The constriction factor χ dampens the position update (eq. (2.10)). It is classically dependent on c_1 and c_2 (eq. (2.11)) but any other value for χ could be chosen. It should be noted that the constriction factor and inertia are connected through (eq. (2.11)) and that PSO with a constriction factor can be seen as a special case of PSO with inertia [38].

$$\vec{x}_i = \vec{x}_i + \chi \vec{v}_i \tag{2.10}$$

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad where \quad \varphi = c_1 + c_2 \quad and \quad \varphi > 4 \tag{2.11}$$

Avoidance of velocity explosion is certain with the use of velocity clamping [36, 39]. This introduces a threshold to prevent a particle from moving far beyond the bounds of the intended search space (eq. (2.12)). Similarly, a threshold on each particle's location is often implemented. Every entry x_{ij} is then bounded between some x_{min} and x_{max} [36].

$$\begin{cases} v_{ij} = v_{max} & \text{if } v_{ij} > v_{max} \\ v_{ij} = -v_{max} & \text{if } v_{ij} < -v_{max} \end{cases} \tag{2.12}$$

Network topology is an important factor in the convergence behavior of PSO [40]. It influences the way information spreads through the swarm by determining communication connections between particles [40]. This can change the weight of the social term in the velocity updates. In the classical PSO model, each particle is allowed to 'communicate' with any other particle. Consequently, in this 'gbest' topology there is always one global optimum which all particles converge to. Another popular topology is the ring or 'lbest' topology, in which each particle is only connected to its K immediate neighbors. Generally, the more connected the swarm is, the faster PSO converges [41]. However, with fast convergence to one solution in the gbest topology, the swarm is sensitive to local optima. In a less connected network with a ring topology information travels more slowly. A higher chance of discovering the global optimum thus comes at the expense of slower convergence [40, 41].

Another proposed solution to combat the problem of local minima is that of restarting particles [42]. This PSO modification labels particles which have become trapped in a local optimum as inactive. A lower bound on the magnitude of velocity updates is used to determine inactivity. The location of inactive particles is reset, allowing them to explore the search space again and thereby increasing the chance of finding a global optimum [13, 43].

CHAPTER 3

Related work

Backpropagation and PSO have both been applied to numerous optimization problems in various fields. Important backpropagation and PSO applications and results from previous studies are summarized here.

3.1 Backpropagation applications

Early on the value of backpropagation for AI and further optimization research became apparent [44]. Despite the local minima problem, state of the art performance has been reached using backpropagation in various optimization problems. Record low error rates (0.35%) were set on the MNIST dataset [17] among many other high-dimensional problems [12, 45, 46].

Some previous research has aimed to examine the scalability of backpropagation. Lawrence, Giles and Tsoi show that a gradient descent solution reached within reasonable training time is often significantly worse than the global optimum [46]. Additionally, they demonstrate a significant deterioration of solutions found by gradient descent as the problem complexity increases. Dauphin et al. show that backpropagation performance falls off in high-dimensional problems because of saddle points [47]. The number of saddle points, at which the gradient of the error function vanishes, increases exponentially with the dimensionality of the problem. Thus, for high-dimensional problems saddle points are the main obstacle for convergence, not local minima.

Peteiro-Barral and Guijarro-Berdiñas compare various gradient descent based methods with respect to scalability [14]. Their research shows that these methods can be used effectively to train ANNs on problem sets from the UCI Machine Learning Repository [10]. The neural networks they train are fairly small however, with a maximum of 6158 weight parameters. This could influence their results with respect to scalability.

3.2 Particle Swarm optimization applications

A substantial amount of research into PSO has been published over the years examining various aspects of the metaheuristic. Many modifications of PSO and other derived metaheuristics have been proposed [1, 5]. Marginal differences in performance depending on the problem follow from the 'No free lunch-theorem' [48]. Regardlessly, stochastic optimization methods like these have been shown to suffer from the 'curse of dimensionality' [49]. Therefore the performance of many metaheuristic likely deteriorates similar to PSO's with increasing problem dimensions. Consequently it is expected that findings on the scalability of PSO in this study can be extrapolated to other metaheuristics.

A selection of works has already focused on the performance of PSO with respect to scalability. In a special issue of Soft Computing on scalability, two studies are dedicated to the examination of scalability of PSO modifications [13, 15]. García-Nieto and Alba [13] raise the issue of a lack of large scale benchmarks. This is evident as both works evaluate the scalability of PSO using optimization problems between 50 and 1000 dimensions [13, 15]. With the growing dimensionality of real world problems it is questionable whether 1000-dimensional problems are large enough to be considered 'large scale'.

Varying results have been found when PSO was applied to neural network training. Compared to backpropagation PSO has some possible advantages. It is independent of the activation function gradient,

has a smaller chance of converging to local optima and is less sensitive to weight initialization [34]. When applied to some networks, PSO has shown optimization performance superior to backpropagation [4, 9]. However, this was only found for networks of small sizes.

The performance of PSO-NN on larger datasets like MNIST has also been examined with varying results. Syulistyo, Purnomo, Rachmadi, and Wibowo [50] take a set of convolutional neural network weights and apply PSO to improve the previously found weights. Although they do not optimize the whole CNN with PSO, their research demonstrates the potential for using PSO in this application. Kösten, Barut, and Acir [51] claim to have reached up to 95% accuracy on MNIST using CNNs optimized with PSO. However the applied methods are not readable as the research is written in Turkish. Additionally they seem to have run tests for up to 10^5 iterations on a GPU. Rakitianskaia and Engelbrecht [16] have applied a PSO-NN to MNIST but with insufficient results. Their research does not specify the reached accuracy as classification errors saw little to no improvement. They suggest that an investigation of the additional PSO parameters maximum velocity, neighborhood topology and constriction factor could improve their results.

Embedding in current research - This study assembles results from previous works as a basis for new research. For its demonstrated performance on the MNIST classification problem, backpropagation is used to represent the state of the art in neural network optimization [45]. Backpropagation and scalability measures are implemented according to existing research, however testing is done on significantly more complex datasets [14]. Furthermore a unique combination of PSO modifications is implemented to improve performance for ANN training. This application of PSO is of substantial complexity when compared to other applications [4, 9, 13, 15]. The study is distinguished from other works that have applied PSO to ANN training on MNIST for two reasons. In relation to the work of Kösten, Barut, and Acir [51] this study provides a hardware-agnostic application of PSO-NN with a clear and reproducible methodology. Compared to Rakitianskaia and Engelbrecht's research [16] additional PSO modifications are employed and a significant improvement in results is observed.

CHAPTER 4

Methodology

This chapter covers the implementation of the evaluated algorithms and the measures used to answer the research question. This study aimed to quantify the scalability of backpropagation and PSO for neural network training. It was hypothesized that accuracy reached by backpropagation and PSO would be similar for small network sizes and that backpropagation would outperform PSO with respect to training time and scalability.

4.1 Neural network implementation

For investigating the scalability of the training methods, neural networks of varying sizes were trained. In particular this research implemented fully connected neural networks for three reasons. Firstly, since the main objective was to evaluate the scalability of training methods the neural networks did not need to attain state of the art performance. Although the implementation of CNNs would likely have improved the final accuracies, it was not expected to improve the quality of the comparison made between backpropagation and PSO. Secondly, the time limitations of this research project motivated the choice for keeping the implementation as simple as possible. Lastly it was expected that a simple network topology would improve the reproducibility of this study.

For the implementation of neural networks of varying complexities, two different network architectures and two different datasets were used (table 4.1). The images in the UCI and MNIST datasets (section 4.4) consist of 64 and 784 pixels respectively. These two input sizes were combined with two different hidden layer structures to get four different neural networks. For the smaller networks one hidden layer with 15 hidden units was chosen, denoted {15}. The larger networks featured two hidden layers with 64 and 32 hidden units respectively, denoted {64, 32}. As each neural network was applied to a digit classification task each output layer consisted of 10 nodes. The number of parameters in table 4.1 was calculated with the inclusion of one bias node for all layers except the output layer.

Table 4.1: Evaluated neural networks

Dataset	Network size	Network structure	Number of parameters
UCI	small	{64, 15, 10}	1135
	large	{64, 64, 32, 10}	6570
MNIST	small	{784, 15, 10}	11935
	large	{784, 64, 32, 10}	52650

4.2 Backpropagation implementation

For the backpropagation implementation a stochastic gradient descent algorithm was used. This algorithm has previously shown strong performance in ANN training and has the potential of escaping local optima [6, 14].

4.3 Particle swarm optimization implementation

The large number of published PSO modifications demanded substantial empirical testing to find an approriate PSO implementation for neural network training. Initially the basic PSO algorithm was implemented but results were not satisfactory. Modifications were gradually added to improve convergence in the ANN training task.

Firstly, inertia ω was implemented to control the balance between exploration and exploitation, and to inhibit velocity growth. In particular the linearly decreasing inertia strategy was used to shift the focus from exploration of the search space to exploitation of the found solutions. A constriction factor χ was implemented to further dampen the position updates.

Despite the velocity restrictions in place empirical testing showed that particles' position and velocity vectors could still grow indefinitely in some dimensions. The high dimensionality of the problem and the probabilistic nature of velocity updates allowed for this explosive behavior. Although technically neural network weights can take up any real value, overshooting the search space is not desirable [34, 36]. This motivated restrictions on particle location and velocity which were applied through x_{max} and v_{max}.

To aid convergence a variable v_{max} was implemented, analogous to a variable learning rate in back-propagation. It was found that particles were more likely to discover new solutions when v_{max} was large. However, local improvement of solutions was most notable when v_{max} was small. This insight motivated a cyclic v_{max} implementation as defined in algorithm 3. The cycle ensured that parts of the search space were thoroughly exploited before further exploration was performed.

Algorithm 3 Cyclic v_{max}

```
Initialize v_max = 2
for number of iterations k do
  if iteration k % 10 = 0 then
    if v_max > 0.1 then
      v_max = 0.75 * v_max
    else
      v_max = 2
    end if
  end if
  for each particle i do
    Limit v_i to v_max in all dimensions
  end for
end for
```

After the implementation of the aforementioned adaptations PSO was noticed to suffer from early convergence. This problematic behavior motivated the use of a neighborhood topology, in which each particle communicates its best position $\vec{x}_{iKbest}$ only with its K nearest neighbors. In the classical star or g_{best} topology where K is equal to the number of particles, all particles converge to the location of g_{best}. Although this results in fast convergence, the probability of particles converging to local optima is relatively high [41]. A ring topology, where $K = 2$, was implemented which slowed down convergence but improved the eventual solutions. The neighborhoods were assigned at initialization and were kept static throughout the iterations. This results in a minimal computational overhead when compared to distance-based neighborhoods which have to be recalculated after each iteration. All modifications considered, the final PSO implementation is shown in algorithm 4.

4.4 Datasets

For the aim of this research neural networks needed to be applied to sufficiently large datasets. A popular application of neural networks is that of image classification. Following this, neural networks were applied to two different image classification tasks. The two tasks, both handwritten digit classification, were used to test the neural networks on problems of varying dimensions.

Firstly, testing was performed on the MNIST handwritten digits dataset [17]. Each sample in this dataset encodes an image of 28x28 pixels of a handwritten digit. This results in a dimensionality of 784 where each input is a grey-value between [0, 255]. The task is to classify images as one of the integers from 0-9. The large number of samples in MNIST allowed for the evaluation of networks trained with

Algorithm 4 Modified PSO

for each particle i **do**
  Initiate $\vec{x}_i$ following a truncated normal distribution [-1, 1]
  Initiate $\vec{v}_i$ following a uniform distribution [-1, 1]
end for
for number of iterations **do**
  for each particle i **do**
    Evaluate fitness and update $\vec{x}_{i,best}$
    if $\vec{x}_{i,best} > \vec{x}_{iKbest}$ **then**
      Update $\vec{x}_{iKbest}$
    end if
    Update $\vec{x}_i$: $\vec{x}_i = \vec{x}_i + \chi\vec{v}_i$
    Apply x_{max}
    Update $\vec{v}_i$: $\vec{v}_i = \omega_i\vec{v}_i + c_1\vec{r}_1(\vec{x}_{i,best} - \vec{x}_i) + c_2\vec{r}_2(\vec{x}_{iKbest} - \vec{x}_i)$
    Update and apply v_{max} following algorithm 3
  end for
  Update ω_i following eq. (5.1)
end for
Return MAX($\vec{x}_{i,best}$)

training sets of varying sizes. The change in performance of these networks with increasing training set sizes is one of the measures used to qualify scalability of training methods, as described in section 4.5.

Additionally, the handwritten digits dataset from the UCI Machine Learning Repository was used [10]. This dataset is of the same nature, but each image sample consists of 8x8 pixels resulting in a dimensionality of 64. As a result ANN training methods could be applied to networks of varying complexities, thereby aiding the evaluation of scalability. Because of the small number of samples in this dataset, it was not applied at varying training set sizes.

4.5 Measurements

The measurements in this research were performed to differentiate the scalability of backpropagation and PSO for ANN training. To qualify scalability six measures from the PASCAL Large Scale Learning challenge were applied [14, 18]:

1. *Err*: The minimum mean squared error reached on the test set
2. *AuTE*: Area under the *training time* vs *test error* curve
3. *Te-n%*: The time t at which test error falls below threshold $n\%$ of the final error
4. *AuSE*: Area under *training set size* vs *test error* curve
5. *Se-n%*: Training set size for which test error falls below threshold $n\%$ of the final error
6. *Eff*: Slope b of the *training set size* vs *training time* curve by using a least squares fit to ax^b

The PASCAL measures, specifically [4-6] required varying training set sizes. As the UCI dataset is relatively small these measures were only applied to the training on MNIST. For a reasonable balance between training times and the number of available training samples, training set sizes of 1000, 2000, 4000 and 8000 were used. For the measures independent of training set size, networks were trained on 8000 samples. For each network trained on the MNIST dataset a test set was used of $\frac{1}{4}$ the size of the training set.

Measurements were performed over discrete time intervals, which led to the results being discontinuous. To calculate the area under curves for measures 2 and 4 a univariate spline was used to interpolate the data. Thereafter the data could simply be integrated over.

In addition to the relation between performance and training set size, tests were conducted to evaluate performance with increasing network complexity. To achieve this neural networks were trained using the topologies from table 4.1. Again, for the MNIST dataset 8000 training samples were used. For the UCI Machine Learning repository dataset 75% of all 5620 samples were used for the training set, resulting in 4215 training samples. Instead of applying the PASCAL measures, a direct comparison of the errors and training times was made between backpropagation and PSO.

4.6 Statistical testing

To ensure statistical significance of the difference in results between backpropagation and PSO the two-tailed non-parametric Mann-Whitney U test was employed. This test does not assume normally distributed data and was chosen following related work [16].

For this test a null hypothesis H_0 and an alternative hypothesis H_1 were defined. The null hypothesis was that the two distributions of results followed from the same population. The alternative hypothesis stated that the two distributions of results did not follow from the same distribution. Both hypotheses were formalized as follows:

$$H_0 : \mu_0 = \mu_1 \tag{4.1}$$

$$H_1 : \mu_0 \neq \mu_1 \tag{4.2}$$

Here μ_0 and μ_1 denote the mean results for backpropagation and PSO respectively. The null hypothesis H_0 would be rejected if the p-value was smaller than significance level α.

CHAPTER 5

Experimental setup and results

All tests in this research were run on and Intel Core i5-4590 quad-core processor with a clock speed of 3.7GHz. For statistical validity all tests were repeated 20 times. For the Mann-Whitney U test the significance level α was set to 0.05.

Following other works in the area of PSO-NN research and empirical results, a selection of hyper parameters for the neural networks was made. The mean squared error was used as the loss function and the sigmoid activation function was used [4, 9, 16]. Similarly for backpropagation the learning rate η was set to 0.01 and weights were initialized following a truncated normal distribution between [-1, 1].

All PSO parameters used are displayed in table 5.1 and were based on previous research and empirical testing. The linearly decreasing inertia was set to decrease from 0.9 to 0.4 such that the inertia at iteration i follows from eq. (5.1) [52]. The constriction factor χ was set to 0.7.

$$w_i = 0.9 - \frac{0.5}{i_{max}} i \tag{5.1}$$

Evualation of backpropagation trained networks showed that the final weights ranged between $[-1.5, 1.5]$. Further experimentation with this result motivated the choice for a value of 3 for the x_{max} threshold used in PSO. The number of particles used in the literature ranged between 10 and 50 and was rounded to 25 and constants c_1 and c_2 were set to 1 and 3 respectively [13, 34, 39]. The particles were initialized according to a truncated normal distribution between [-1, 1] with the aim of replicating backpropagation's convergence behavior and the algorithm was ran for 4000 iterations.

Table 5.1: PSO parameter values

Parameter	Value
iterations	4000
particles	25
c_1	1
c_2	3
ω_{max}	0.9
ω_{min}	0.4
χ	0.7
v_{max}	[0.1, 2]
x_{max}	3
K	2

A total of 10 different tests were run using each training method. For the PASCAL scalability measures backpropagation and PSO were used to train two neural networks on MNIST with 1000, 2000, 4000 and 8000 training samples. One network with a single hidden layer containing 15 nodes and another with two hidden layers containing 64 and 32 nodes (denoted {15} and {64, 32}), resulting in 8 tests for each training method. Additionally both methods were used to train networks with the same structures on the UCI dataset with 4215 training samples, resulting in 2 more tests for both backpropagation and PSO.

5.1 PASCAL results

The measures from the PASCAL large scale learning challenge were only applied to MNIST as the UCI dataset did not provide enough samples. The results of the corresponding tests using both training methods and both hidden layer structures ({15} and {64, 32}) are displayed in figs. 5.1 to 5.3. For fig. 5.1 MNIST was trained on 8000 samples, and for figs. 5.2 and 5.3 training set sizes were varied. Following these tests, the graphs were used to calculate the PASCAL measures as shown in table 5.2. The best results for each category are displayed in bold font.

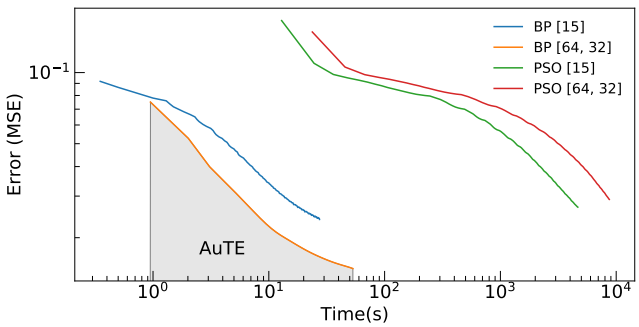

Figure 5.1: Training time vs Error

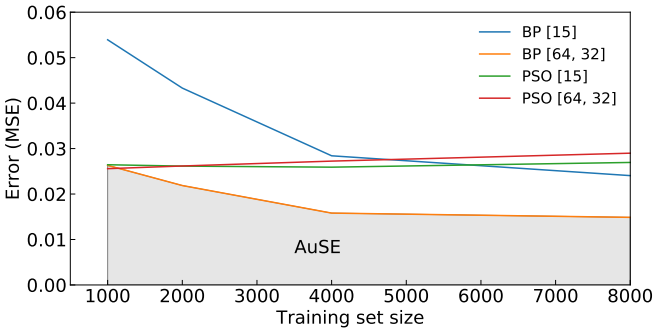

Figure 5.2: Training set size vs Error

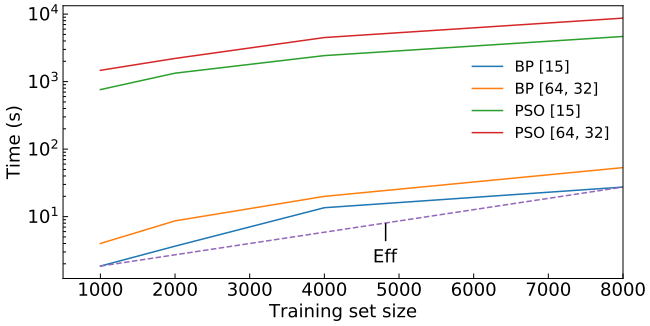

Figure 5.3: Training set size vs Time

Table 5.2: PASCAL measures applied to BP and PSO for training ANNs on MNIST

Method	Structure	*Err*	*AuTE*	*Te*-5%	*AuSE*	*Se*-5%	*Eff*
BP	{15}	0.024 ± 0.001	**0.99**	**23.2**	212.37	8000	1.21
PSO	{15}	0.027 ± 0.003	212.07	4244.1	183.30	**1000**	**0.90**
BP	{64, 32}	$\mathbf{0.015 \pm 0.001}$	1.11	36.6	**117.44**	8000	1.35
PSO	{64, 32}	0.029 ± 0.004	425.97	8106.4	191.94	**1000**	0.94

5.2 MNIST and UCI results

The results of tests ran on both the MNIST and UCI datasets can be seen in table 5.3. These results demonstrate the performance of the training methods for training ANNs of varying complexities. In particular, this table reports the means and standard deviations of final accuracies and times reached by backpropagation and PSO.

The convergence behavior of networks trained by backpropagation and PSO can be seen in figs. 5.4 to 5.7. The mean convergence is graphed and the area between the best and worst convergence of all runs is shaded.

Table 5.3: Performance of BP and PSO with increasing network complexity

Dataset	Structure	#Params	BP Accuracy	PSO Accuracy	BP Time(s)	PSO Time(s)
UCI	{15}	1135	0.92 ± 0.009	0.91 ± 0.021	9.75 ± 0.4	404.2 ± 0.4
UCI	{64, 32}	6570	0.94 ± 0.004	0.89 ± 0.042	14.3 ± 0.5	2122.7 ± 1.6
MNIST	{15}	11935	0.86 ± 0.008	0.78 ± 0.020	27.4 ± 0.5	4663.7 ± 23.2
MNIST	{64, 32}	52650	0.90 ± 0.005	0.75 ± 0.036	53.2 ± 0.7	8716.2 ± 38.7

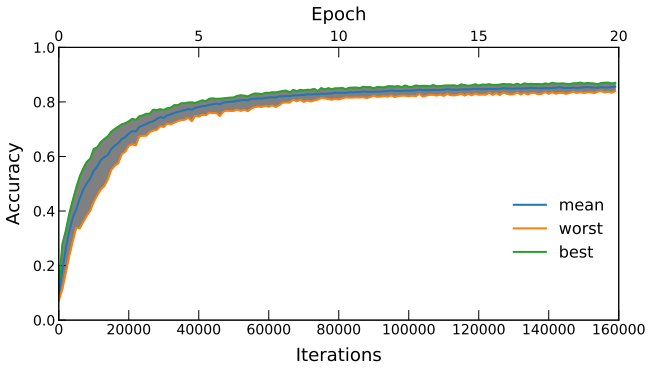

Figure 5.4: BP {15} convergence

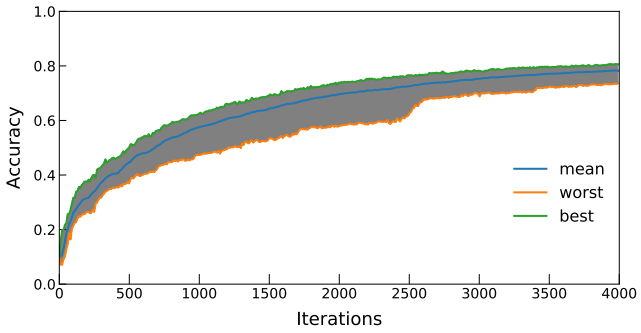

Figure 5.5: PSO {15} convergence

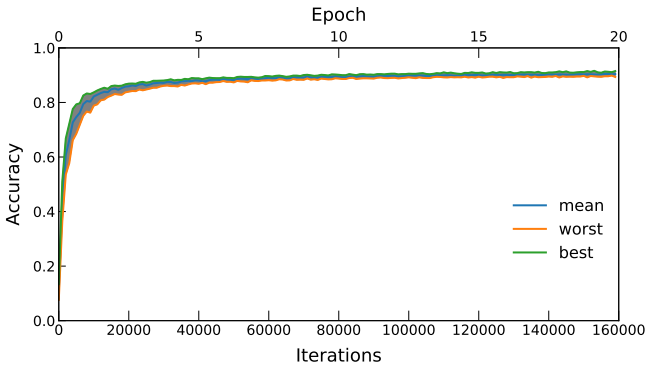

Figure 5.6: BP {64, 32} convergence

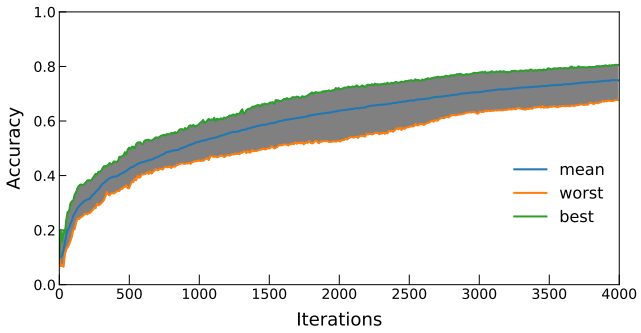

Figure 5.7: PSO {64, 32} convergence

CHAPTER 6

Discussion

This research hypothesized that reached accuracies of backpropagation and PSO would be comparable for the training of small ANNs. It was expected that, with an increase of problem dimensionality, backpropagation would quickly outperform PSO with respect to training time and accuracy. The following sections analyze and interpret the results to evaluate the hypothesis.

6.1 UCI

To evaluate the first part of the hypothesis the results obtained by the smallest neural networks in this study are analyzed. In particular this section examines the UCI networks with the {15} and {64, 32} hidden layer structures from table 5.3.

UCI {15} - The accuracies reached by backpropagation and PSO on the UCI network with the {15} hidden layer structure are very similar. The corresponding Mann-Whitney U test gave a p-value of 0.20, which is larger than the significance level α of 0.05. Consequently the null hypothesis H_0 could not be rejected, indicating that there is no statistically valid difference between the final accuracies. A significant difference can be seen in terms of training times which were over 40 times higher for PSO than for backpropagation. This is a direct result of the training process of PSO. Candidate solutions have to be evaluated for each particle in the swarm resulting in a long training time.

UCI {64, 32} - Where backpropagation's performance improves for training the {64, 32} structure network, PSO's performance deteriorates with the extra hidden layer. The Mann-Whitney U test results in a p-value of 3.33×10^{-8} in this case and H_0 is rejected. The observed decrease in accuracy for PSO implies that the current implementation struggles in high dimensional problems and that it is not sufficiently scalable. This is confirmed by the observed training times which are over 150 times higher for PSO than for backpropagation in this case.

6.2 MNIST

The neural networks applied to MNIST were significantly more complex than those applied to UCI and will therefore be used to evaluate the second part of the hypothesis regarding scalability.

MNIST {15} - Although both algorithms attain lower accuracies on the MNIST dataset than on the UCI dataset, the decrease in performance is largest for PSO. The Mann-Whitney U test for the reached errors and accuracies (tables 5.2 and 5.3) report p-values of 5.54×10^{-6} and 2.31×10^{-8} respectively. Consequently both null hypotheses are rejected and significant differences in performance are concluded.

Table 5.2 shows huge discrepancies in terms of *AuTE* and *Te-5%* as a result of PSO's much longer training time (table 5.3). In terms of *AuSE* PSO slightly outperforms backpropagation. This can be explained by the extensive exploration of the search space performed by PSO. During this process PSO will evaluate many candidate solutions even when few training samples are available, resulting in a relatively low error. In contrast backpropagation struggles to extract sufficient information from the small set of samples as a result of its short training time. The *Se-5%* results for PSO cannot meaningfully be interpreted. In fig. 5.2 error for PSO rises with an increase of training samples. This indicates a flawed

implementation as more information should not lead to less knowledge. To extract sufficient information from the supplied training samples PSO would need to perform a further exploration of the search space. Consequently PSO would need more training time, making its slight benefit over backpropagation in terms of *Eff* score insignificant.

MNIST {64, 32} - The differences in performance between the MNIST {15} and {64, 32} networks are similar to those observed between the UCI {15} and {64, 32} networks. Again backpropagation's error and accuracy significantly improve whereas PSO's deteriorate. Accordingly the differences in results are statistically valid with p-values for the errors and accuracies of 3.40×10^{-8} and 3.36×10^{-8} respectively.

As a result of this trend PSO's *AuSE* stays similar where it improves for backpropagation. This clearly indicates that the current PSO implementation is not suitable to be scaled up further. Another result of PSO's stagnated error together with an increased training time is a significant increase in $AuTE$ and *Te-5%* compared to a marginal increase for backpropagation. The *Eff* score results are again deemed to be insignificant.

Finally, when examining figs. 5.4 to 5.7 the reliability of each of the training methods can be observed. The figures clearly show that backpropagation's convergence is consistent, where PSO's convergence has sizeable margins. This discrepancy can be ascribed to the stochastic nature of the PSO algorithm.

6.3 Interpretation of results

Following the previous sections which analysed the results an answer can be formulated to the research question:

> *Is there a difference in the scalability of backpropagation and particle swarm optimization when applied to ANN training?*

This study hypothesized that accuracies reached by the two methods would be comparable for small networks but that, with an increase in problem size, backpropagation would quickly outperform PSO in terms of training time and accuracy.

Considering the results, the hypothesis can be confirmed. For the single hidden layer network (denoted {15}) trained on the UCI dataset no statistically significant differences were found in the final accuracy between the two methods. Furthermore, the results on the MNIST dataset support the hypothesized behavior of PSO with increasing problem dimensionality. All tests, except those on the aforementioned UCI {15} network, demonstrated backpropagation's superiority over PSO in terms of both training time and accuracy.

The discrepancy in reached accuracies between the two methods can be ascribed to the early convergence of PSO. The invariable error with increasing training set sizes for PSO (fig. 5.2) indicates this early convergence. It shows that PSO does not explore the problem space further when fed with more training samples. Because of limited time resources additional testing with even more samples was not possible. However, the current results were consistent over the repeated experiments. Despite all the implemented modifications to aid PSO's convergence this problematic behaviour remained.

The large difference in training times is inherent to PSO as it evaluates a large number of candidate solutions each iteration. In the following chapter the research is concluded and possible modifications to further improve PSO's convergence and training time are discussed.

CHAPTER 7

Conclusion

Real world applications of artificial neural networks are becoming increasingly complex. With this trend the scalability of neural network training methods is a relevant area of research. This study examined the scalability of backpropagation and particle swarm optimization for this application. To qualify their scalability both training methods were applied to neural networks of increasing sizes. Handwritten digit classification tasks from MNIST and the UCI machine learning repository were used as benchmarks of varying complexities. For the aim of the research PSO was implemented with a set of modifications to improve convergence on the neural network training task. Most importantly a neighborhood topology, a linearly decreasing inertia weight and restrictions on velocity and location were used.

All test results demonstrated a clear difference in the scalability of backpropagation and PSO for ANN training. For small neural networks accuracies reached by backpropagation and PSO were similar, but backpropagation consistently outperformed PSO as neural network sizes were increased. Additionally backpropagation's training time was significantly lower than that of PSO in all tests. Despite PSO's reasonable accuracies, backpropagation is concluded to be the superior ANN training method in terms of accuracy, training time and scalability.

Future research could work on improving the exploration of the problem space in the current PSO implementation. To negate the problem of early convergence the weight of the social term in the velocity equation may need to be decreased. Additionally, experimentation with restarting particles could result in improved convergence for PSO. The training time required by PSO could benefit from the implementation of minibatches, in which weights are only updated after a fixed number of iterations.

Bibliography

[1] A. K. Kar, "Bio inspired computing – A review of algorithms and scope of applications," *Expert Systems with Applications*, vol. 59, pp. 20–32, Oct. 2016. DOI: 10.1016/J.ESWA.2016.04.018.

[2] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 305–313. DOI: 10.5555/89851.89891.

[3] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, Mar. 1996. DOI: 10.1109/2.485891.

[4] N. Mohammadi and S. J. Mirabedini, "Comparison Of Particle Swarm Optimization And Backpropagation Algorithms For Training Feedforward Neural Network," *Journal of Mathematics and Computer Science*, vol. 12, no. 02, pp. 113–123, 2014. DOI: 10.22436/jmcs.012.02.03.

[5] V. K. Ojha, A. Abraham, and V. Snášel, "Metaheuristic design of feedforward neural networks: A review of two decades of research," *Engineering Applications of Artificial Intelligence*, Apr. 2017. DOI: 10.1016/j.engappai.2017.01.013. eprint: 1705.05584.

[6] F. Ye, "Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data," *PLOS ONE*, vol. 12, no. 12, pp. 1–36, Dec. 2017. DOI: 10.1371/journal.pone.0188746.

[7] J. R. Zhang, J. Zhang, T. M. Lok, and M. R. Lyu, "A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training," *Applied Mathematics and Computation*, vol. 185, no. 2, pp. 1026–1037, Feb. 2007. DOI: 10.1016/j.amc.2006.07.025.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Nov. 1995, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

[9] K. Khan and A. Sahai, "A Comparison of BA, GA, PSO, BP and LM for Training Feed forward Neural Networks in e-Learning Context," *International Journal of Intelligent Systems and Applications*, vol. 4, no. 7, pp. 23–29, Jun. 2012. DOI: 10.5815/ijisa.2012.07.03.

[10] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[11] L. Prechelt, "PROBEN 1 - a set of benchmarks and benchmarking rules for neural network training algorithms," Tech. Rep., 1994.

[12] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, Jan. 2015. DOI: 10.1016/j.neunet.2014.09.003.

[13] J. García-Nieto and E. Alba, "Restart particle swarm optimization with velocity modulation: A scalability test," *Soft Computing*, vol. 15, no. 11, pp. 2221–2232, Sep. 2011. DOI: 10.1007/s00500-010-0648-1.

[14] D. Peteiro-Barral and B. Guijarro-Berdiñas, "A study on the scalability of artificial neural networks training algorithms using multiple-criteria decision-making methods," in *Artificial Intelligence and Soft Computing*, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds., Berlin, Germany: Springer, 2013, pp. 162–173. DOI: 10.1007/978-3-642-38658-9_15.

[15] M. A. Montes de Oca, D. Aydın, and T. Stützle, "An incremental particle swarm for large-scale continuous optimization problems: An example of tuning-in-the-loop (re) design of optimization algorithms," *Soft Computing*, vol. 15, no. 11, pp. 2233–2255, Sep. 2011. DOI: 10.1007/s00500-010-0649-0.

[16] A. Rakitianskaia and A. Engelbrecht, "Training high-dimensional neural networks with cooperative particle swarm optimiser," in *2014 International Joint Conference on Neural Networks (IJCNN)*, 2014, pp. 4011–4018. DOI: 10.1109/IJCNN.2014.6889933.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. DOI: 10.1109/5.726791.

[18] S. Sonnenburg, V. Franc, E. Yom-Tov, and M. Sebag, "Pascal large scale learning challenge," *25th International Conference on Machine Learning (ICML2008) Workshop. J. Mach. Learn. Res*, vol. 10, pp. 1937–1953, Jan. 2008.

[19] D. Peteiro-Barral, V. Bolón-Canedo, A. Alonso-Betanzos, B. Guijarro-Berdiñas, and N. Sánchez-Maroño, "Toward the scalability of neural networks through feature selection," *Expert Systems with Applications*, vol. 40, no. 8, pp. 2807–2816, Jun. 2013. DOI: https://doi.org/10.1016/j.eswa.2012.11.016.

[20] M. Lozano, D. Molina, and F. Herrera, "Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems," *Soft Computing*, vol. 15, no. 11, pp. 2085–2087, Sep. 2011. DOI: 10.1007/s00500-010-0639-2.

[21] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989. DOI: 10.1162/neco.1989.1.4.541.

[23] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," eng, in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, IEEE, 2010, pp. 317–324. DOI: 10.1109/PDP.2010.43.

[24] A. L. Blum and R. L. Rivest, "Training a 3-node neural network is np-complete," *Neural Networks*, vol. 5, no. 1, pp. 117–127, 1992. DOI: 10.1016/S0893-6080(05)80010-3.

[25] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 8595–8598. DOI: 10.1109/ICASSP.2013.6639343.

[26] M. Minsky and S. Papert, *Perceptrons*. Oxford, England: M.I.T. Press, 1969.

[27] L. Jackel, H. Graf, W. Hubbard, J. Denker, D. Henderson, and I. Guyon, "An application of neural net chips: Handwritten digit recognition," in *IEEE 1988 International Conference on Neural Networks*, vol. 2, 1988, pp. 107–115. DOI: 10.1109/ICNN.1988.23918.

[28] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017. arXiv: 1712.06567.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[30] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76–86, Jan. 1992. DOI: 10.1109/34.107014.

[31] J. M. McInerney, K. G. Haines, S. Biafore, and R. Hecht-Nielsen, "Back propagation error surfaces can have local minima," in *International 1989 Joint Conference on Neural Networks*, vol. 2, 1989, p. 627. DOI: 10.1109/IJCNN.1989.118524.

[32] X. Wang, Z. Tang, H. Tamura, and M. Ishii, "A modified error function for the backpropagation algorithm," *Neurocomputing*, vol. 57, pp. 477–484, Mar. 2004. DOI: 10.1016/j.neucom.2003.12.006.

[33] K. Burse, M. Manoria, and V. P. S. Kirar, "Improved back propagation algorithm to avoid local minima in multiplicative neuron model," in *International Conference on Advances in Information Technology and Mobile Communication*, Springer, 2011, pp. 67–73. DOI: 10.1007/978-3-642-20573-6_11.

[34] A. Rakitianskaia and A. Engelbrecht, "Saturation in PSO neural network training: Good or evil?" In *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Sep. 2015, pp. 125–132. DOI: 10.1109/CEC.2015.7256883.

[35] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 1998, pp. 69–73.

[36] F. Marini and B. Walczak, "Particle swarm optimization (pso). a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.

[37] J. C. Bansal, P. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization," in *2011 Third world congress on nature and biologically inspired computing*, IEEE, 2011, pp. 633–640.

[38] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, 2000, pp. 84–88. DOI: 10.1109/ICEC.1998.699146.

[39] J. Kennedy, "Swarm intelligence," in *Handbook of nature-inspired and innovative computing*, Springer, 2006, pp. 187–219. DOI: 10.1007/0-387-27705-6.

[40] J. Kennedy, "Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 3, 1999, 1931–1938 Vol. 3.

[41] A. Muñoz-Zavala, A. Hernandez-Aguirre, and E. Villa Diharce, "A new neighborhood topology for the particle swarm optimization algorithm," in *Particle Swarm Optimization: Theory, Techniques and Applications*. Nova Science Publishers, Jan. 2011, pp. 227–247.

[42] K. Tatsumi, T. Yukami, and T. Tanino, "Restarting multi-type particle swarm optimization using an adaptive selection of particle type," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 923–928. DOI: 10.1109/ICSMC.2009.5346746.

[43] J. Zhang, X. Zhu, W. Wang, and J. Yao, "A fast restarting particle swarm optimizer," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 1351–1358. DOI: 10.1109/CEC.2014.6900427.

[44] P. J. Werbos, "Backpropagation: Past and future," in *IEEE 1988 International Conference on Neural Networks*, vol. 1, 1988, pp. 343–353. DOI: 10.1109/ICNN.1988.23866.

[45] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, Dec. 2010. DOI: 10.1162/neco_a_00052.

[46] S. Lawrence, C. L. Giles, and A. C. Tsoi, "What size neural network gives optimal generalization? convergence properties of backpropagation," Tech. Rep., Jun. 1996. [Online]. Available: http://hdl.handle.net/1903/809.

[47] Y. N. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *CoRR*, 2014. arXiv: 1406.2572.

[48] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997. DOI: 10.1109/4235.585893.

[49] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004. DOI: 10.1109/TEVC.2004.826069.

[50] A. R. Syulistyo, D. M. J. Purnomo, M. F. Rachmadi, and A. Wibowo, "Particle swarm optimization (pso) for training optimization on convolutional neural network (cnn)," *Jurnal Ilmu Komputer dan Informasi*, vol. 9, no. 1, pp. 52–58, 2016. DOI: 10.21609/jiki.v9i1.366.

[51] M. M. Kösten, M. Barut, and N. Acir, "Deep neural network training with ipso algorithm," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, 2018, pp. 1–4. DOI: 10.1109/SIU.2018.8404721.

[52] C. Chen, C. Chuang, and M. Wu, "Combining concepts of inertia weights and constriction factors in particle swarm optimization," in *2012 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA) Proceedings*, 2012, pp. 73–76. DOI: 10.1109/CIMSA.2012.6269606.