Combine Genetic Algorithm and Particle Swarm Optimization Approach for Neural Network Classification

DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's name:  
Signature: 
Date: 29/02/2016
Combine Genetic Algorithm and Particle Swarm Optimization Approach for Neural Network Classification

أساليب دمج بين الخوارزمية الجينية وخوارزمية السرب للتصنيف بطريقة الخلايا العصبية

Enas Ahmed Abu Hamra

Supervised by
Prof. Nabil M. Hewahi

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

February/2016
نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيك لجنة الحكم على أطروحة الباحثة/ة إنِّس أَمَّة عَتْة أَبُو حَمْرَة لنيل درجة الماجستير في كلية تكنولوجيا المعلومات برامج تكنولوجيا المعلومات وموضوعة:

أسلوب دمج بين الخوارزمية الجينية وخوارزمية السرب لتصنيف بطريقة الخلايا العصبية

Combine Genetic Algorithm and Particle Swarm Optimization Approach for Neural Network Classification

وبعد المناقشة التي تمت اليوم الثلاثاء 14 جمادى الأولى 1436هـ، الموافق 23/02/2016م الساعة الحادية عشرة صبحًا، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

أ.د. نبيل محمود حويحي

مشرفًا ورئيسًا

أ.د. أشرف بونس مغاري

مناقشة داخلية

أ.د. إيهاب صلاح الدين رزقوت

مناقشة خارجية

وبعد المداولة أوصت اللجنة بمنح الباحثة درجة الماجستير في كلية تكنولوجيا المعلومات/ برنامج تكنولوجيا المعلومات.

واللجنة إذ تمنح هذه الدرجة فإنها توصي بها بتقوى الله ولزوم طاعته وأن تسخر علماً في خدمة دينها ووطنها.

وَللهُ الطَّوْلُ وَالْحَرْقُ..."
Dedication

I dedicate my thesis' work to my family and many friends. A special feeling of gratitude to my loving parents, that the secret of my success is their dua’a. Special dedication for my mother; words of encouragement and push for tenacity ring in my ears. My sisters and brothers have never left my side and are very special.

I Love You Always and Forever

Enas Abu Hamra
Acknowledgements

I will forever be thankful to Allah, the merciful and the compassionate for giving me health and strength to start and complete this study. This master thesis is an actual educational experience to me. I would like to take this opportunity to express my sincere appreciation to all people who have been involved with this thesis and gave me their hands and support. It was impossible to complete this thesis without their contributions.

I would like to express my deepest gratitude and appreciation to my thesis’ supervisor Prof. Nabil M. Hewahi for his supervision, guidance, continued support and encouragement through all stages of my thesis. Thank you for introducing me to this interested topic.

I express my sincere appreciation to my friends; Reem Thabet and Sheonagh Schultz for their efforts in proofreading.

Also, I would like to acknowledge and thank the academic and administrative staff in the faculty of Information Technology at Islamic university of Gaza for their collaborations, supports and assistance.

Furthermore, I would like to express my heart-felt gratitude to my parents for their faith in me and allowing me to be as ambitious as I wanted. It was under the watchful eye that I gained so much drive and an ability to tackle challenges head on.

Finally, and most importantly, my great thanks to all the hidden hands whom share and assist me to deliver this thesis.
Abstract

Artificial Neural Network (ANN) has played a significant role in many areas because of its ability to solve many complex problems that mathematical methods failed to solve. However, it has some shortcomings that lead it to stop working in some cases or decrease the result accuracy. This research proposed a new approach combining the most famous optimization algorithms, namely the particle swarm optimization algorithm (PSO) and the genetic algorithm (GA), to increase the classification accuracy of ANN.

The proposed approach utilizes the advantages of both PSO and GA to overcome the local minima problem of ANN, which prevents ANN from improving the classification accuracy. It starts with finding out the best ANN using backpropagation algorithm through various attempts to use it as one of the population for the GA algorithm. If the solution is still not reached, PSO algorithm will start working with the half population which has the best fitness values. The process of keeping repeatedly applying GA followed by PSO with every time half of the last population with the best fitness values will be applied until the optimum solution is obtained.

In contrary to other approaches, the proposed approach is domain independent, and has been evaluated by applying it using nine datasets with various domains and characteristics. The testing was performed with three main different approaches, first is only using the ANN without any optimization algorithms, the second is applying our proposed approach and the third is applying various methods presented in previous researches; GA alone, ANN followed by GA, PSO alone, ANN followed by PSO and GA followed by PSO.

The comparison results show the superiority and the capability of our proposed approach for all the datasets to increase the classification accuracy whether the classification is high or low using other approaches.

Keywords: Genetic Algorithm, Particle Swarm Optimization, Artificial Neural Networks, Classification.
ملخص: أسلوب دمج بين الخوارزمية الجينية وخوارزمية السرب للتصنيف بطريقة الخلايا العصبية

تلعب الخلايا العصبية دورًا هامًا في الكثير من المجالات. ومع ذلك، فإنها تعاني من بعض أوجه القصور التي تؤدي إلى توقف العمل في بعض الحالات أو تقلص دقة النتائج. وبالتالي، اقترح هذا البحث نهجًا جديدًا يجمع بين خوارزميات التحسين الأكثر شهرة، وهي خوارزمية السرب والخوارزمية الجينية (GA) لزيادة دقة تصنيف الخلايا العصبية.

يستخدم النهج المقترح مزايا كل من خوارزمية السرب والخوارزمية الجينية لتغلب على مشكلة الحدود الدنيا المحلية للخلية العصبية، والذي يمنعها من تحسين دقة التصنيف. فهي تبدأ بإيجاد أفضل خلية عصبية باستخدام خوارزمية الانتشار الخلفي من خلال عدة محاولات لاستخدامها كأحد عناصر الخوارزمية الجينية. في حال لم يتم الوصول إلى حل فسيتم الانتقال إلى خوارزمية السرب من خلال استخدام نصف عينة الخوارزمية الجينية والتي تحتوي أفضل القيم. سيتم الاستمرار في تنفيذ الخوارزمية الجينية تليها خوارزمية السرب مع استخدام نصف العينة والتي تحتوي أفضل القيم إلى أن يتم الوصول لأفضل حل.

خلافاً للاستراتيجيات الأخرى، فإن النهج المقترح لا يعتمد على نوع محدد من البيانات، وتم تقييمها من خلال تنفيذها باستخدام تسع قواعد البيانات المختلفة في المجال والخصائص. وتم تقسيم الفحص إلى ثلاث مراحل وهي تطبيق قواعد البيانات على الشبكة العصبية بدون استخدام أي تحسين، تطبيق قواعد البيانات على الشبكة العصبية باستخدام النهج المقترح وتطبيق قواعد البيانات على بعض الطرق المختلفة والتي تم ذكرها في أبحاث سابقة وهي الخوارزمية الجينية، الخلايا العصبية يليها الخوارزمية الجينية، خوارزمية السرب، الخلايا العصبية يليها خوارزمية السرب، الخوارزمية الجينية يليها خوارزمية السرب.

وأظهرت النتائج تفوق وقوة النهج المقترح على تحسين دقة التصنيف لجميع مصادر البيانات التي تم استخدامها. مقارنة بداء الطرق الأخرى سواء كانت دقة التصنيف منخفضة أو مرتفعة بالطرق الأخرى.

الكلمات المفتاحية: الخوارزمية الجينية، خوارزمية السرب، الخلايا العصبية، التصنيف.
Table of contents

Dedication .................................................................................................................. I
Acknowledgements ...................................................................................................... II
Abstract ....................................................................................................................... III
Abstract in Arabic ....................................................................................................... IV
Table of contents ......................................................................................................... V
List of figures ............................................................................................................... VII
List of tables ................................................................................................................. VIII
List of abbreviations ................................................................................................... IX
Chapter 1 : Introduction .......................................................................................... 1
  1.1 Artificial Neural Network .................................................................................... 2
  1.2 Backpropagation Algorithm .............................................................................. 3
      1.2.1 Forward phase ............................................................................................. 4
      1.2.2 Backward phase .......................................................................................... 4
      1.2.3 Problems of BP algorithm ........................................................................... 5
  1.3 Genetic Algorithm ............................................................................................. 6
  1.4 Particle Swarm Optimization Algorithm ........................................................... 8
  1.5 PSO vs. GA ......................................................................................................... 10
  1.6 Statement of the Problem ................................................................................. 11
  1.7 Objectives ......................................................................................................... 11
      1.7.1 Main Objective ............................................................................................ 11
      1.7.2 Specific Objectives .................................................................................... 12
  1.8 Significance of the Thesis ................................................................................. 12
  1.9 Scope and Limitations ....................................................................................... 12
  1.10 Research Methodology .................................................................................... 12
  1.11 Thesis Outline ................................................................................................ 13
Chapter 2 : Related Works ...................................................................................... 14
Chapter 3 : Methodology and Implementation ....................................................... 17
  3.1 Overall Methodology ....................................................................................... 17
      3.2.1 Crossover operator .................................................................................... 20
      3.2.2 Mutation operator .................................................................................... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 ANN representation as a particle</td>
<td>24</td>
</tr>
<tr>
<td>3.4 Choosing problem domains</td>
<td>26</td>
</tr>
<tr>
<td>3.5 Data encoding</td>
<td>28</td>
</tr>
<tr>
<td>3.6 Network structure</td>
<td>28</td>
</tr>
<tr>
<td>3.7 Parameters</td>
<td>31</td>
</tr>
<tr>
<td>3.8 Tools, Equipment and Methods</td>
<td>32</td>
</tr>
<tr>
<td>Chapter 4 : Experimental Results and Evaluation</td>
<td>33</td>
</tr>
<tr>
<td>4.1 Learning Based ANN</td>
<td>33</td>
</tr>
<tr>
<td>4.2 The proposed approach</td>
<td>34</td>
</tr>
<tr>
<td>4.3 Other Approaches</td>
<td>35</td>
</tr>
<tr>
<td>4.4 Discussion</td>
<td>40</td>
</tr>
<tr>
<td>Chapter 5 : Conclusion and Future Directions</td>
<td>41</td>
</tr>
<tr>
<td>5.1 Conclusion</td>
<td>41</td>
</tr>
<tr>
<td>5.2 Future Directions</td>
<td>42</td>
</tr>
<tr>
<td>References</td>
<td>43</td>
</tr>
<tr>
<td>Appendix A</td>
<td>49</td>
</tr>
</tbody>
</table>
List of figures

Figure 1.1: MLP Model ................................................................. 3
Figure 1.2: Gradient descent stuck at local minima ..................................... 6
Figure 1.3: GA structure ..................................................................... 8
Figure 1.4: PSO algorithm flowchart ...................................................... 10
Figure 3.1: Proposed methodology flowchart ........................................ 18
Figure 3.2: GA population .................................................................. 20
Figure 3.3: Single point crossover technique .......................................... 21
Figure 3.4: Before crossover .................................................................. 21
Figure 3.5: After crossover ................................................................... 22
Figure 3.6: Chromosome values before and after mutation ......................... 23
Figure 3.7: Weights of ANN in the car domain ......................................... 24
Figure 3.8: Example of calculating new position and new velocity ............... 25
Figure 3.9: Example of calculating new velocity & new position (continued) .......................................... 26
Figure 3.10: Car dataset file before encoding .......................................... 27
Figure 3.11: Encoding data .................................................................. 28
Figure 3.12: Best number of hidden nodes of Car dataset ......................... 29
Figure 4.1: A comparison between the results of the Glass Identification dataset .......................................... 36
Figure 4.2: A comparison between the results of the Nursery dataset .................. 36
Figure 4.3: A comparison between the results of the Balance Scale dataset .......... 37
Figure 4.4: A comparison between the results of the Car dataset .................. 37
Figure 4.5: A comparison between the results of the User Knowledge Modeling dataset ....... 38
Figure 4.6: A comparison between the results of the Contraceptive Method Choice dataset . 38
Figure 4.7: A comparison between the results of the Ecoli dataset .................... 39
Figure 4.8: A comparison between the results of the Seeds dataset .................... 39
Figure 4.9: A comparison between the results of the Draw dataset .................... 40
List of tables

Table 3.1: Characteristics of datasets ................................................................. 27
Table 3.2: Number of hidden layers & Network structure for datasets .............. 30
Table 3.3: Proposed algorithm's parameters ....................................................... 31
Table 4.1: Accuracy percentage on ANN without optimization algorithms ......... 33
Table 4.2: Comparison between ANN results and our approach results .......... 34
Table 4.3: Accuracy percentage for previous methods ........................................ 35
List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>BP</td>
<td>Backpropagation</td>
</tr>
<tr>
<td>UCI</td>
<td>UC Irvine Machine Learning Repository</td>
</tr>
</tbody>
</table>
Chapter 1 : Introduction

As life becomes increasingly complex, new techniques must be developed to facilitate these added complications, resolving all the problems that traditional methods cannot solve. Artificial Neural Network (ANN or NN) is an information processing system that mimics the structure and function of the human brain. It consists of neurons arranged in layers, connecting with each other using weights that determine the strength of their relationship (Jia & Zhu, 2009) (Azzam-ul-Asar, et al. 2007). It has proved its success in solving a great number of problems across different fields. However, it also has some shortcomings related to the setting of the weights of the neural connections during the training phase that restricts its ability to solve specific problems. The weights of the ANN have to be optimized in order to reach a good and accurate output. Therefore, the training process should result in finding the weights associated to the minimum output error (Caputo, et al. 2010). One of the main shortcomings with ANN that prevents ANN from reaching a high accuracy is local minima. When an ANN learning algorithm causes the total error of the net to descend into a valley of the error surface, that valley may or may not lead to the lowest point on the entire error surface. If it does not, the minimum into which the total error will eventually fall is termed, “a local minima” (Wilson, 2012). The common objective in ANN classification is to reach the global optima.

There are many algorithms that are used for ANN’s training in order to optimize its performance. Most famously, the particle swarm optimization algorithm (PSO) and the genetic algorithm (GA). The PSO concept is inspired by the social behavior of a bird flocking or a fish schooling in its search for food, while GA is based on the concepts of natural selection and genetics (Hongbiao & Genwang, 2012) (Zang & Yu, 2011). PSO and GA are similar in many ways. Both algorithms start with a randomly generated population, using fitness function to evaluate the population and search for the optimal solution, with neither of them guaranteeing success (Mishra & Patra, 2008). Moreover, both algorithms have advantages that make them successful in many areas, but at the same time, they suffer from some shortcomings (Chang & Xu, 2008) (Dan, 2013) (Mishra & Patra, 2008) (Xin-hui, et al. 2010).

Until now only few research projects have combined both PSO and GA to optimize the performance of ANN but for specific structure and domain (Juang, 2004) (Chen, et al. 2008)
This research proposes a new approach for combining PSO and GA to increase the ANN classification for any domain.

1.1 Artificial Neural Network

Artificial neural network is an important part of artificial intelligence (Wang & Li, 2010). In addition, it is a mathematical model that mimics the structure and function of a human brain (Chogumaira & Hiyama, 2009) (Jia & Zhu, 2009). The human brain consists of billions of neurons that are connected, communicating with each other by the use of electrical signals (Haron, et al. 2012). ANN, like a human brain, consists of simple processing units, which are called neurons, organized in layers and connected to each other through connection weights and bias value for information transmission and processing. The weights and biases are adjusted automatically in the learning process (Zhang, et al. 2009) (Shenglong & Tonghui, 2012). The concept of learning from inputs to outputs in ANN is similar to the way that the human brain learns from experience (Miao, et al. 2010). Most learning rules can be categorized into either; supervised or unsupervised methods (Haron, et al. 2012). In supervised method, inputs and outputs are provided and the neural network is being trained based on that. Once the neural network is trained, it is expected to be able to classify unknown cases as well. On the other hand, in unsupervised method, only inputs are given and then clustered based on their internal relation. In our research we are concerned with supervised neural networks.

There are many different types of ANN structures. One common structure is the Multi-Layer Perceptron (MLP), which is a feed forward type of the neural network. The typical MLP network model consists of a group of neurons with three categories, which are input neurons, hidden neurons, and output neurons. Each neuron is in one layer and is connected with all neurons of the adjusted layer. Every connection is having a weight that measures the importance of the link. As shown in Figure 1.1, each circle represents a neuron. From left to right, the three layers are called input layer, hidden layer and output layer respectively (Fu, et al. 2011). The operation of a typical MLP network can be divided into two phases, which are training and testing phases. The MLP network must be trained for its specific purpose using learning algorithms like backpropagation algorithm. After the step of training, the MLP network can be used to generate the outputs (Al-Shareef & Abbod, 2010).
Over nearly 20 years, neural network has achieved great success and progress in many research areas such as pattern recognition, forecasting, automatic control, signal processing, decision support, robotics, etc. (Jia & Zhu, 2009) (Yong, et al. 2010) (Ye Qian & Pengfei Zhang, 2010). It also has many advantages compared to the statistical classification methods, such as parallel processing, self-organization, self-learning, adaptive, nonlinear dynamic mechanisms and association ability, etc. (Wang & Li, 2010) (Ye Qian & Pengfei Zhang, 2010).

1.2 Backpropagation Algorithm

Backpropagation algorithm (BP) is the most popular supervised learning method for multilayer feed-forward neural networks and is based on the gradient descent method (Islam, et al. 2011) (Feng, et al. 2011). In 1974, it was discovered by Werbos in his PhD. thesis and was later developed by Rumelhart, Hinton, and Williams in 1986 (Rumelhart, et al. 1986).

The process of BP learning algorithm can be divided into two stages: a forward phase and a backward phase (Lu, 2011). First, the input pattern is propagated from the input layer to the output layer and, as a result of this forward flow of data, it produces output. The error signals resulting from the difference between output pattern and an actual output are back-propagated from the output layer to the previous layers to update their weights (Al-Zubaidy & Shambour, 2011). This process will continue until the network error decreases within the acceptable range or the network reaches the maximum learning times (Leng, et al. 2013).
1.2.1 Forward phase

The principle of forward phase can be described as follows: Assuming the input of some neural cell is real continuous variable $x=(x_1, x_2, \ldots, x_n)$, and the connecting weight between the cell and its superior cell is $w=(w_1, w_2, \ldots, w_n)$.

The neural cell processes the input data through two steps: the first is the weighted summation of input data, and the second is the adding of a bias variable $b$ that results in the net input of the neural cell

$$net = \sum w_i x_i + b \quad (1)$$

The other step is the function operation of net value and the output $y$ of neural cell is produced

$$y = f(net) \quad (2)$$

$f(net)$ is called activating function or transform function, which often takes the form of step function, linear function or Sigmoid function, of which the Sigmoid function is more commonly used. The Sigmoid function is as follows

$$f(x) = \frac{1}{1+e^{-\lambda x}} \quad (3)$$

To calculate the error for each output neuron using the squared error function between the actual output $o_i$ and the target output $t_i$ and sum them to get the total error (Bao, et al. 2011).

$$E = \sum \frac{1}{2}(o_i-t_i)^2 \quad (4)$$

1.2.2 Backward phase

The goal with backward phase is to update each of the weights in the network, so that they cause the actual output to be closer to the target output, thereby minimizing the error for each output neuron and the network as a whole. It is two steps; error is propagated from output layer to hidden layer, then from hidden layer to input layer. The network weights are updated in the negative gradient direction. A learning rate $\gamma$ defines the step length of the correction. The corrections for the weights are given by

$$\Delta w_{ji} = -\gamma o_i \delta_j \quad (5)$$

$$\Delta b = \gamma \delta_j \quad (6)$$

where $\delta_j = o_j(1-o_j)(o_j-t_j)$, if node $j$ is an output node.

and $\delta_k = o_k(1-o_k) \sum \delta_j w_{kj}$, if the node $k$ is a hidden node.

Finally adjust the weights and biases of all links as follows:

$$w_{(\text{new})} = w_{(\text{old})} + \Delta w \quad (7)$$
$$b_{(\text{new})} = b_{(\text{old})} + \Delta b$$  \hspace{1cm} (8)

After updating weights and biases, check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output by using the following formula (Rojas, 1996).

$$E_{AV} = \frac{1}{m} \sum_{n=1}^{m} E(n)$$  \hspace{1cm} (9)

Where \(m\) is the number of training samples and \(E(n)\) is the error for each output neuron. From equation “(9)” it can be seen that \(E_{AV}\) is the function of all weights, threshold values as well as input signals of the ANN. To minimize the function value, it is necessary to let the ANN keep learning (Bao, et al. 2011).

1.2.3 Problems of BP algorithm

Although the BP algorithm has been shown to be effective in training feedforward neural networks in many applications, it contains some drawbacks; the major one being that it is easy to get stuck in local minima, it becomes unable to find the optimal global solution of a problem and the generalization ability of trained ANN becomes poor (Islam, et al. 2011) (Xiao, et al. 2007). The cause of this problem is that the BP algorithm adopts the gradient descent algorithm to train the network which requires careful selection of parameters such as network topology, initial weights and biases, learning rate value, activation function, and value for the gain in the activation function. An improper choice of these parameters can lead to slow network convergence, network error or failure (Rehman & Nawi, 2011) (Leng, et al. 2013). Referring to Figure 1.2, if the starting point for gradient descent was chosen inappropriately, more iterations of the algorithm will only make it approach a local minimum, never reaching the global one (Teng, 2014).
Evolutionary algorithms can help prevent the problem of convergence to local minima and can exploit global optima. Most of these algorithms draw their inspiration from biological processes like particle swarm optimization (PSO) algorithm, ant colony optimization (ACO) algorithm and genetic algorithm (GA) (Camargo, et al. 2012) (Kattan, et al. 2010).

1.3 Genetic Algorithm

A genetic algorithm is a kind of artificial intelligence and global optimization methods (Xin-hui, et al. 2010). It is based on the concepts of natural selection and genetics, which was proposed in 1975 by Holland, from the University of Michigan, and his students (Zhang, et al. 2009) (Zang & Yu, 2011). It is especially useful for complex optimization problems where the parameters’ numbers are large and the analytical solutions are difficult to obtain, and it overcomes the fault of easily falling into local minima point using traditional optimization methods. This algorithm has been applied in different areas such as fuzzy control, path planning, modeling, classification, etc. (Mishra & Patra, 2008) (Xin-hui, et al. 2010). Its basic operations are coding, choice of fitness function and genetic operators which are selection, crossover and mutation (Xin-hui, et al. 2010) (Gill, et al. 2010).

The operations of GA are as follows (Goldberg, 1989):

a) **Encoding:** The genetic algorithm cannot directly deal with a problem’s data, so the chromosomes are converted in a sequence string using a coding mechanism. The study of biological inheritance initiate from the chromosome, the chromosomes are arranged in a string from the gene. In general, genetic algorithms depend on binary encoding.
b) **Generate initial population:** The initial population is randomly generated containing solutions, which are called individuals or chromosomes.

c) **Determine the parameters:** Include the size of selected population, crossover probability $P_c$, mutation probability $P_m$ and other supplementary parameters (Zhang, et al. 2009).

d) **Define the fitness function.** This function is used to evaluate the chromosomes in order to transfer the best to the next generation. Fitness function is a measure of solution quality and an important basis for selection through evolution any changes. For different problems, the definition ways of fitness function are different (Li, et al. 2011).

e) **Genetic operators (selection, crossover and mutation):** The selection operator in GA achieves the purpose of the survival of the fittest individuals. The individual with high fitness has a high probability to the next group and vice versa. Crossover operation plays a key role in the genetic algorithm and in the main method of generating new individuals. For each generation, two selected parents' chromosomes will go through a crossover operation process to produce offspring on the basis of crossover probability. In this operation, the first step is to generate a random crossover location. Then, all genes after that location are swapped between the parents to produce two offspring. Assume that two chromosomes were $a = (1 \ 1 \ 1 \ 1)$ and $b = (0 \ 0 \ 0 \ 0)$ keep the first bit and exchange the others between parents, producing two new chromosomes which are $a’ = (1 \ 0 \ 0 \ 0)$ and $b’ = (0 \ 1 \ 1 \ 1)$. The two offspring will then replace the worst two chromosomes. Subsequently, all the chromosomes will go through mutation operation. In mutation operation, a random chromosome is selected, based on the probability, a random position in this chromosome is chosen and the bit value is swapped. For instance, if mutation is applied to the third bit of string $a’$, the transformed string becomes $(1 \ 0 \ 1 \ 0)$. The mutation operation is an auxiliary method in the process in which GA determines the ability of GA’s local search, while maintaining the population diversity (Wan-Yu & Kai, 2007) (Ahmad, et al. 2010).

f) **Repeat (d), (e) steps** until reaching the max-generation or find an acceptable solution (Miao, et al. 2010).
To simplify the operations of GA, Figure 1.3 expresses these operations as a flowchart.

![Figure 1.3: GA structure (Zhang, et al. 2009)](image)

The GA is applied to solve complex design optimization problems because it can handle both discrete and continuous variables and nonlinear objective and constrain functions without requiring gradient information (Katiyar, 2010). It is a non-linear optimization method that has strong global searching capacity (Zhang & Wang, 2008). However, GA in the fitness function under the condition of improper selection may have converging local optimal and could not achieve optimum. For dynamic data, there are some difficulties in calculating the optimal solution by using a genetic algorithm, because chromosome population is likely to cause premature convergence, and will not produce any new changes to altered data (Xiaofeng, 2011).

### 1.4 Particle Swarm Optimization Algorithm

PSO is one of the evolutionary algorithms proposed by James Kennedy and Russell Eberhart in 1995 and its idea is inspired by the social behavior of a bird flocking or a fish schooling in its search for food (Hu, et al. 2013) (Hongbiao & Genwang, 2012). The theory centers around the idea that flocks of birds search randomly for food in a specific region, and though initially none of the birds know the location of food, they are able to gain knowledge about its location after each iteration. Each bird shares information with the other birds in a swarm, so a bird searching for food at the next iteration depends on self-experience and the experiences of other birds. In the same way, the PSO algorithm works to get the best solution for optimization problems. The algorithm depends on a population that represents a swarm
containing lots of particles where each of them refers to a solution of an optimization problem. The algorithm searches for the optimal value through iterations, while evaluating the quality of a solution through the fitness function (Dan, 2013) (Chang & Xu, 2008).

Each particle has a position denoted as \( x \) and a velocity denoted as \( v \) and adjusts its position according to the pbest, which refers to the current best position of a particle that was reached and the gbest, which refers to the best position of all particles in a swarm respectively. And using the following rules which are:

\[
v_{i+1} = wv_i + c_1 r_1 (pbest_i - x_i) + c_2 r_2 (gbest_i - x_i) \\
x_{i+1} = x_i + v_{i+1}
\]  

(10)  

(11)

Where \( w \) is inertia weight coefficient which can be a value between 0 and 1 of computed based on a given formula, \( c_1 \) and \( c_2 \) are acceleration constants mostly in interval \([0, 2]\), \( r_1 \) and \( r_2 \) are random numbers in interval \([0, 1]\). In addition, the position and velocity of the particle are limited to the range \([X_{min}, X_{max}]\) and \([V_{min}, V_{max}]\) (Chang & Xu, 2008) (Hongbiao & Genwang, 2012).

The procedures of the PSO can be summarized as follows (Chen, et al. 2008):

1) Initialize the population of particles and the position and velocity for each particle randomly.

2) Evaluate the fitness value of all particles depending on the fitness function based on the goal of the optimization problem.

3) Make a comparison between each particle’s fitness value and its pbest. If the current value is better than the pbest, then assign the current fitness value to its pbest. Otherwise the pbest will remain as it is.

4) Determine the current best fitness value among all the particles’ fitness values in a population. Then, compare this value with the gbest and assign a current value to gbest if the current value is better than gbest, otherwise leave it as it is.

5) Update the position and velocity of each particle using the functions (10) and (11).

6) Stop if the best solution that match the predefined minimum error is found or reached the maximum number of iteration, otherwise repeat the process from step (2) to (5).
To simplify the operations of PSO, Figure 1.4 expresses these operations as a flowchart.

![PSO algorithm flowchart](image)

**Figure 1.4: PSO algorithm flowchart (Ülker & Ülker, 2014)**

PSO is successfully and widely applied in many areas such as function optimization, artificial neural network training, task assignment, pattern recognition, fuzzy system control and other fields (Chang & Xu, 2008). It has many advantages, including the fact that it is very simple, easy to understand and implement, strong, robust, has great potential for global exploration and does not need to be adjusted by many parameters (Ren, et al. 2012) (Hu, et al. 2013). However, PSO has the phenomenon of premature convergence (Dan, 2013).

1.5 **PSO vs. GA**

PSO is similar to GA in many ways. Both algorithms start with a randomly generated population, using fitness function depending on the problem requiring a solution, to evaluate and update the population and search for the optimal solution, with neither of them guaranteeing success. In contrast, PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. They also have memory, which is important for the algorithm; whereas in GA, previous knowledge of the problem is destroyed once the population changes (Juang, 2004). PSO uses real numbers to represent the particles unlike GA, which needs to be changed to binary encoding or special genetic operators that have to be used. The information sharing mechanism in PSO is significantly different. In GAs, chromosomes...
share information with each other. So the whole population moves like a single group towards an optimal area. PSO, only gbest gives information to others (Mishra & Patra, 2008) (Srivastava, et al. 2009) (Azzam-ul-Asar, et al. 2007). It should be noted that PSO algorithm does not require sorting of fitness values of solutions in any process. This might be a significant computational advantage over GA, especially when the population size is large. The updates of velocity and position in PSO also only require a simple arithmetic operation of real numbers (Kachitvichyanukul, 2012).

GA is an efficient and parallel method that can perform global searching. It can acquire and accumulate the knowledge of search space automatically in the search process, and control the search process self-adaptively to achieve optimization. However, the traditional genetic algorithm exists "pseudo-random" and "precocious" phenomenon, get precise solution is very inefficient. PSO algorithm is essentially a type of random search algorithm, with advantages that include individual number less, simple calculation, high precision, strong local search capability, and faster convergence speed, but the global search ability, PSO than GA is poor. GA and PSO, can complement each other, that integrates global searching ability with high convergence speed make the optimization effect is more ideal (Xin-qiu & Yan-sheng, 2011).

1.6 Statement of the Problem

Even though the ANN proved successful in many areas, in cases where mathematical approaches were unable to find a solution, it suffers from some shortcomings, including falling into local minima, which causes oscillation and slow convergence rates for some problems. To solve this problem, various research attempts have been tried in order to improve the classification accuracy of ANN for certain domains. This was done by incorporating various techniques, including GA and PSO, or a combination of them. The goal of this research is to propose an approach based on PSO and GA to overcome the problem of local minima and to increase the classification accuracy of ANN, regardless of the domain.

1.7 Objectives
1.7.1 Main Objective

The main objective of this research is to design a new optimization approach that depends on a combination of GA and PSO to increase the classification accuracy of ANN regardless of the domain.
1.7.2 Specific Objectives

- Using GA for ANN.
- Using PSO for ANN.
- Combination of GA and PSO to achieve the main objectives.
- Incorporation and modification of the used tool (WEKA) in according with our needs.
- Applying the experiments in various domains.
- Analyzing the results.

1.8 Significance of the Thesis

1. Improve the classification accuracy of ANN.
2. Promote the research in ANN classification.
3. Develop an approach to increase the classification accuracy for any domain.
4. Utilize a hybrid approach of PSO and GA in a way that might be useful in other areas.

1.9 Scope and Limitations

1. The used tool for ANN classification is WEKA.
2. The proposed approach to be applied on many datasets from various domains.
3. The used datasets are collected from UCI (Asuncion & Newman, 2007).
4. The neural network structure contains one hidden layer with multiple nodes that is determined by WEKA program, not multiple hidden layers.

1.10 Research Methodology

Our approach is based on the following steps to achieve our objectives:

1. **Collecting data:** Datasets from various domains are collected from UCI.
2. **Algorithm development:** Developing a new approach based on a combination of GA and PSO to improve the classification of ANN. Chapter 3 describes our proposed approach.
3. **Implementation:** This step includes the coding of data to be compatible with the neural network needs, using WEKA program to choose the best number of hidden nodes for the neural network structure and using Microsoft Visual studio IDE and C# programming language to implement our algorithm.
4. Evaluation:

Each dataset is going to be divided into training set and testing set. The proposed approach will be evaluated based on the following process:

a. Each dataset will be applied on ANN without any optimization algorithm.

b. Each dataset will be used with some methods presented in previous researches which are ANN with GA, ANN with PSO, GA followed by PSO, and GA and PSO each alone.

c. Each dataset will be used with our proposed approach.

d. Compare the results obtained by all the above approaches in a, b and c and analyze them based on the thesis objectives.

1.11 Thesis Outline

This thesis is organized as follows:

**Chapter (2):** Includes works, approaches and methods related to improve the classification accuracy of ANN. More concentration will be on combining both genetic algorithm and particle swarm optimization to enhance the performance of artificial neural network. This chapter clarifies the difference between these works and our proposed approach.

**Chapter (3):** Describes the proposed approach and how it can be applied.

**Chapter (4):** Presents the used datasets, tools and the conducted experiments including the analysis and evaluation of the results.

**Chapter (5):** Presents the conclusion of the thesis and the future directions.
Chapter 2: Related Works

Few studies combined the PSO and GA to enhance the performance of the ANNs, so this chapter presents these studies and at the end clarifies the differences between the proposed approach and these studies.

Juang (Juang, 2004) proposed a new evolutionary learning algorithm called HGAPSO. The concept of this algorithm was that in each generation, after the calculation of the fitness values of all individuals in the same population, the top-half best performing ones were chosen and regarded as elites. Subsequently, these elites were enhanced by using PSO. The group of elites is regarded as a swarm and each elite corresponds to a particle in it. At the end, these enhanced elites constitute half of the population in the new generation, whereas the other half is generated by performing a crossover and mutation operation on these enhanced elites. This algorithm is applied in temporal sequence production by fully connected recurrent neural network and dynamic plant control problems by Takagi-Sugeno-Kang-type recurrent fuzzy network and compared to both GA and PSO and prove its efficiency.

Chen, et al., (Chen, et al. 2008) proposed the method that integrates the GA with PSO by adding PSO as an operator inside GA, meaning that the operators of GA, which are selection, crossover and mutation, must be run, and then the PSO should be run in each iteration. It is applied to the temperature neural network prediction in transverse flux induction heating. The results showed that the performance of this algorithm is better than GA or PSO and can be used in electromagnetic engineering domain.

Kuo, et al., (Kuo, et al. 2009) proposed a new algorithm called HPSGO to improve the learning performance of radial-basis function neural network (RBFNN). The idea of this algorithm was that in each generation in GA and before performing the GA operators, the PSO learning in only one iteration must be performed first. After that, the produced chromosomes were duplicated to current chromosomes. After implementing the GA operators, a new population from two groups of chromosomes that were produced from GA with PSO and PSO alone was selected. This algorithm is applied to the problem of forecasting the daily sales of papaya milk in the retail industry and is compared with other algorithms, namely GA, PSO and the Box-
Jenkins model. The Experimental results showed that the proposed algorithm is better in accuracy than other algorithms in sales forecasting problem.

Caputo, et al., (Caputo, et al. 2010) proposed an evolutionary optimization algorithm, called GSO. The idea of this algorithm is that in every iteration of the population is randomly divided into two parts which are evolved with GA and PSO techniques respectively. At this point, the fitness of the newly generated individuals is evaluated and they are recombined in the updated population which is again divided into two parts in the next iteration for the next run of genetic or particle swarm operators. The simulation results showed its ability to be used in neural network applications for a wide spectrum of engineering problems.

Xin-qian and Yan-sheng (Xin-qiu & Yan-sheng, 2011) combined the PSO and GA by adding the GA crossover and mutation operators inside the PSO which is called GPSA. The researcher established a backpropagation (BP) neural network tandem cold rolling force prediction model, and optimized it using this approach. The highest prediction accuracy of the GPSA approach proved that this method can be applied to practical production.

Yu-liang, et al., (Yu-liang, et al. 2012) used the same approach presented in (Xin-qiu & Yan-sheng, 2011) but with another type of neural network and domain. The researcher combined the GA and PSO to optimize the radial-basis function neural network (RBFNN) which is called GA-PSO-RBF, and then applied it in fault diagnosis for generator unit. This idea was created to avoid the drawbacks of PSO-RBF. The simulation results showed that GA-PSO-RBF is better than PSO-RBF in training speed, convergence accuracy, and diagnosis accuracy; thus, it was a new efficient diagnosis approach.

From previous studies, researchers focused their approaches on optimizing a specific neural network that is used to solve a particular problem. In this research, the proposed approach is domain independent. In addition, our proposed approach is different from all the previous methods, where in all the previous methods a sort of internal mixture and combination is performed between GA and PSO whereas in our proposed approach we keep applying the optimization alternatively between GA and PSO until no improvement can be done. The idea behind this is to continuously exploit each of the optimization algorithms capabilities at certain stage. The concept behind our approach is to obtain the highest possible classification accuracy
by utilizing the advantages of each of the optimization approaches and overcome the disadvantages of each approach by the other approach.
Chapter 3: Methodology and Implementation

This chapter presents and explains the structure of the proposed approach and the steps which have been followed to implement this approach.

3.1 Overall Methodology

The proposed methodology is depicted in Figure 3.1. The steps of the proposed approach can be summarized as follows:

1. Choose a real dataset as a problem domain.
2. Create a population of size n of ANNs. This population contains ANNs of the same structure with randomly varied weights.
3. Set the initial value of the best set of ANNs m as n (m = n).
4. Apply backpropagation algorithm on ANNs until an acceptable solution is reached (reach the requested error) or no improvement. If an acceptable solution has been reached, then the algorithm is quitted and the solution is found. If the solution is not obtained, do the following:
   4.1 Calculate the fitness value for each ANN (The number of correct classifications)
   4.2 Repeat the following steps until an acceptable solution is reached or no further improvement is possible.
      4.2.1 Form the obtained ANNs as a population for the GA. Use GA to update the weights of the ANNs by using crossover and mutation. GA will stop when no more improvements can be achieved to reach an acceptable solution (in this last case, the algorithm will quit in case the required solution is obtained).
      4.2.2 If no more improvements with the usage of GA;
         Select the best m (the m top ANNs) ANNs from the n ANNs population. The m can be computed as m = n/2.
         Apply PSO to improve the classification of m ANNs.
      4.2.3 If no more improvement with usage of PSO;
         The new n ANNs population becomes the m top ones (this means n = m).
      4.2.4 Otherwise, if an acceptable solution is reached, quit the algorithm with the obtained solution.
5. Select the best ANN as the best classifier.

Figure 3.1: Proposed methodology flowchart
The proposed approach is based on a practical idea that says: PSO is better than GA when the population is small. The idea is taken from the travelling sales-man problem where PSO performs better than GA when the number of cities is less (Rani & Kumar, 2014) (Malik & Tayal, 2014). The proposed methodology starts with GA and then applies PSO for a smaller list of ANNs. Furthermore, the proposed approach keeps minimizing the optimum set until the best one is reached. After applying backpropagation for the first time on the initial population, GA and PSO keep turning the population size repeatedly by half until the optimum solution is obtained. The other interesting part of the algorithm is that it might quit at any stage if an acceptable solution is obtained. The repeating process of PSO and GA ensures that the proposed approach obtains the optimum solution regardless of the domain. The code for some functions of the proposed approach is shown in Appendix A.

3.2 ANN representation as a chromosome

The total number of ANN weights depends on the number of neurons in all layers which are input layer, hidden layer and output layer and is calculated by using the following formula.

Total number of weights = (number of input neurons * number of hidden neurons) +
(number of hidden neurons * number of output neurons) + number of hidden neurons +
number of output neurons

Where the number of hidden neurons is to represent the bias weights of the hidden neurons and the number of output neurons is to represent the bias weights of the output neurons. Initial weights are chosen randomly within some small interval that is [-1, 1], and each weight is a real number corresponds to the weighted link from one neuron to another. Suppose we have a 4-7-3 neural network, so the total number of weights is (4*7) + (7*3) + 7 + 3 = 59. Since a chromosome is a collection of genes, a set of weights can be represented by n-gene chromosome, where each gene corresponds to a single weighted link in the network. This means each ANN is represented by one chromosome, and the genes of the chromosome are the weights of ANN. Figure 3.2 illustrates a sample of GA population, where this population is represented by six chromosomes to represent six ANN, and each chromosome is represented by the weights of its corresponding ANN. As shown in the Figure 3.2, the population is a matrix of n-chromosomes and each row contains a group of genes equals to the network’s weights that represent a chromosome.
3.2.1 Crossover operator

Crossover needs chromosomes that are selected from the population to be parents. The problem is how to select these chromosomes. According to Darwin's evolution theory, the best ones should survive and create new offspring. There are many methods how to select the best chromosomes; roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others (Sharma & Mehta, 2013). In our proposed algorithm, we used the tournament selection; n individuals are selected randomly from the GA population. The n value is determined by tau-percent from the population size; suppose that the tau value=0.4 and the population size=10, then the n value=10*0.4=4 and the selected individuals are sorted from smallest error to largest error (in our case we use Mean Square Error (MSE) as shown at the end of this section), and the top two of those individuals are returned.

We get two parents from the tournament selection and by applying the crossover operator; we will get two new offspring. In our case, we use single point crossover where a single point on both parents string is selected, then all data beyond that point in either parent is swapped between the parent organisms. Figure 3.3 illustrates the single point crossover.
Figure 3.3: Single point crossover technique

Figure 3.4 and Figure 3.5 illustrate the chromosomes before and after the crossover of the Glass dataset.
We used MSE to calculate the error. For example, if one target output is (0, 0, 1, 0) for Car dataset, and the computed output is (0.2, 0.1, 0.7, 0.4), then the squared error is 

\[
\frac{(0.2 - 0)^2 + (0.1 - 0)^2 + (0.7 - 1)^2 + (0.4 - 0)^2}{2} = \frac{0.04 + 0.01 + 0.09 + 0.16}{2} = 0.3/2 = 0.15.
\]

The mean value of squared errors is the average of all errors on the training data.

### 3.2.2 Mutation operator

The mutateRate parameter is essentially the probability that a particular gene will be mutated. If a gene is targeted for mutation, its value is adjusted up or down by a random amount proportional to the value of the mutateChange and maxGene parameters. For example, suppose maxGene = 10.0 and mutateChange is 0.01. The mutation delta will be a random value between -1.0 and +1.0. Figure 3.6 shows the chromosome values for the Glass dataset before and after mutation.
Figure 3.6: Chromosome values before and after mutation

Figure 3.7 shows a sample of the ANN weights in the proposed algorithm and also represent a chromosome in the GA population. These weights of the 21-6-4 neural network of the Car dataset and the total number of weights is \((21 \times 6) + (6 \times 4) + 6 + 4 = 160\).
3.3 ANN representation as a particle

Like GA, PSO population is a swarm of particles and each particle has a position and a velocity. The ANN weights are represented as a particle’s position, and the particle’s velocity will be initialized randomly.

PSO uses a collection of particles and it is an iterative process. In each iteration, every particle moves to a new position which, hopefully, represents a better problem solution. A particle's movement is based on the particle's current speed and velocity.

The key to PSO is the computation of a particle's new velocity. Expressed in math terms, the velocity and position update equations are:

\[ v(t+1) = (w * v(t)) + (c1 * r1 * (p(t) - x(t)) + (c2 * r2 * (g(t) - x(t)) \]

\[ x(t+1) = x(t) + v(t+1) \]

**Figure 3.7: Weights of ANN in the car domain**

<table>
<thead>
<tr>
<th>-0.069</th>
<th>-0.003</th>
<th>-0.500</th>
<th>-0.085</th>
<th>0.984</th>
<th>-0.692</th>
<th>0.425</th>
<th>0.595</th>
<th>-0.419</th>
<th>-0.304</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.927</td>
<td>-0.382</td>
<td>0.568</td>
<td>-0.522</td>
<td>0.524</td>
<td>-0.499</td>
<td>-0.257</td>
<td>-0.091</td>
<td>0.478</td>
<td>0.612</td>
</tr>
<tr>
<td>-0.932</td>
<td>-0.128</td>
<td>-0.615</td>
<td>-0.031</td>
<td>-0.814</td>
<td>-0.217</td>
<td>0.974</td>
<td>0.676</td>
<td>-0.703</td>
<td>-0.878</td>
</tr>
<tr>
<td>-0.905</td>
<td>-0.107</td>
<td>0.039</td>
<td>0.767</td>
<td>-0.466</td>
<td>-0.909</td>
<td>-0.199</td>
<td>-0.288</td>
<td>0.989</td>
<td>0.873</td>
</tr>
<tr>
<td>-0.848</td>
<td>0.794</td>
<td>-0.087</td>
<td>0.851</td>
<td>-0.680</td>
<td>-0.347</td>
<td>0.914</td>
<td>0.840</td>
<td>-0.650</td>
<td>0.797</td>
</tr>
<tr>
<td>-0.137</td>
<td>-0.038</td>
<td>-0.543</td>
<td>0.142</td>
<td>-0.610</td>
<td>-0.941</td>
<td>-0.387</td>
<td>0.531</td>
<td>-0.272</td>
<td>0.201</td>
</tr>
<tr>
<td>-0.666</td>
<td>0.794</td>
<td>0.298</td>
<td>-0.542</td>
<td>-0.399</td>
<td>0.180</td>
<td>0.579</td>
<td>0.792</td>
<td>0.944</td>
<td>0.433</td>
</tr>
<tr>
<td>-0.699</td>
<td>0.969</td>
<td>-0.880</td>
<td>-0.605</td>
<td>0.460</td>
<td>0.227</td>
<td>0.959</td>
<td>-0.466</td>
<td>-0.351</td>
<td>0.533</td>
</tr>
<tr>
<td>-0.131</td>
<td>-0.866</td>
<td>0.326</td>
<td>-0.500</td>
<td>0.260</td>
<td>0.133</td>
<td>-0.564</td>
<td>0.897</td>
<td>0.377</td>
<td>-0.525</td>
</tr>
<tr>
<td>0.479</td>
<td>0.270</td>
<td>0.984</td>
<td>-0.212</td>
<td>0.539</td>
<td>-0.597</td>
<td>-0.557</td>
<td>-0.545</td>
<td>0.250</td>
<td>0.140</td>
</tr>
<tr>
<td>0.074</td>
<td>-0.879</td>
<td>0.896</td>
<td>-0.083</td>
<td>-0.902</td>
<td>-0.168</td>
<td>-0.958</td>
<td>-0.148</td>
<td>0.682</td>
<td>0.163</td>
</tr>
<tr>
<td>-0.900</td>
<td>-0.921</td>
<td>-0.117</td>
<td>0.195</td>
<td>-0.668</td>
<td>-0.800</td>
<td>-0.577</td>
<td>-0.203</td>
<td>0.199</td>
<td>0.467</td>
</tr>
<tr>
<td>-0.256</td>
<td>0.681</td>
<td>-0.585</td>
<td>0.468</td>
<td>0.954</td>
<td>-0.571</td>
<td>-0.953</td>
<td>-0.867</td>
<td>-0.934</td>
<td>0.057</td>
</tr>
<tr>
<td>-0.216</td>
<td>0.504</td>
<td>0.284</td>
<td>0.509</td>
<td>-0.541</td>
<td>-0.252</td>
<td>-0.762</td>
<td>-0.591</td>
<td>-0.597</td>
<td>-0.572</td>
</tr>
<tr>
<td>0.091</td>
<td>0.584</td>
<td>-0.785</td>
<td>-0.786</td>
<td>-0.625</td>
<td>0.406</td>
<td>-0.612</td>
<td>-0.211</td>
<td>-0.544</td>
<td>0.339</td>
</tr>
<tr>
<td>0.980</td>
<td>0.646</td>
<td>0.256</td>
<td>0.782</td>
<td>-0.605</td>
<td>0.393</td>
<td>0.706</td>
<td>-0.573</td>
<td>-0.037</td>
<td>0.669</td>
</tr>
<tr>
<td>-0.215</td>
<td>0.909</td>
<td>-0.214</td>
<td>-0.375</td>
<td>-0.621</td>
<td>-0.404</td>
<td>-0.206</td>
<td>0.373</td>
<td>-0.264</td>
<td>0.584</td>
</tr>
<tr>
<td>0.962</td>
<td>-0.986</td>
<td>-0.605</td>
<td>-0.229</td>
<td>-0.624</td>
<td>0.160</td>
<td>0.466</td>
<td>-0.799</td>
<td>0.093</td>
<td>-0.446</td>
</tr>
<tr>
<td>-0.959</td>
<td>0.258</td>
<td>0.677</td>
<td>-0.273</td>
<td>0.077</td>
<td>0.138</td>
<td>-0.753</td>
<td>0.502</td>
<td>0.114</td>
<td>0.066</td>
</tr>
<tr>
<td>0.042</td>
<td>0.811</td>
<td>0.496</td>
<td>-0.266</td>
<td>0.643</td>
<td>0.182</td>
<td>-0.202</td>
<td>0.591</td>
<td>0.835</td>
<td>0.779</td>
</tr>
<tr>
<td>-0.394</td>
<td>0.014</td>
<td>-0.947</td>
<td>-0.128</td>
<td>0.377</td>
<td>0.966</td>
<td>0.251</td>
<td>-0.515</td>
<td>0.406</td>
<td>0.235</td>
</tr>
<tr>
<td>0.927</td>
<td>0.506</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let's say that a particle's current position, \(x(t)\), is \(\{x_0, x_1\} = \{2.0, 3.0\}\), and that the particle's current velocity, \(v(t)\), is \(-1.0, -1.5\). Additionally, assume that constant \(w = 0.7\), constant \(c_1 = 1.4\), constant \(c_2 = 1.4\), and that random numbers \(r_1\) and \(r_2\) are 0.5 and 0.6, respectively. Finally, suppose that the particle's best known position is \(p(t) = \{2.5, 3.6\}\) and the global best known position by any particle in the swarm is \(g(t) = \{1.3, 2.4\}\). Then, the new velocity and position values are:

\[
v(t+1) = (0.7 \times \{-1.0, -1.5\}) + 
(1.4 \times 0.5 \times (\{2.5, 3.6\} - \{2.0, 3.0\})) + 
(1.4 \times 0.6 \times (\{1.3, 2.4\} - \{2.0, 3.0\})) \\
= \{-0.70, -1.05\} + \{0.35, 0.42\} + \{-0.59, -0.50\} \\
= \{-0.94, -1.13\}.
\]

\[
x(t+1) = \{2.0, 3.0\} + \{-0.94, -1.13\} \\
= \{1.06, 1.87\}.
\]

Figure 3.8 and Figure 3.9 illustrate an example to calculate new velocity and new position of a particle in a Glass dataset.

---

**Figure 3.8:** Example of calculating new position and new velocity
3.4 Choosing problem domains

The proposed approach is applied on nine datasets obtained from UCI (Asuncion & Newman, 2007). The datasets are selected to have various domains and characteristics including number of instances, number of attributes and type of attributes. Before using the datasets, they have been processed by deleting the records with missing values to get good results. The following table illustrates the datasets which were used and their characteristics.

![Table showing datasets characteristics](image)
<table>
<thead>
<tr>
<th>Dataset name</th>
<th>no. of instances</th>
<th>no. of attributes without the class</th>
<th>Attributes type</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass Identification</td>
<td>214</td>
<td>9</td>
<td>Real</td>
<td>Physical</td>
</tr>
<tr>
<td>Nursery</td>
<td>12960</td>
<td>8</td>
<td>Categorical</td>
<td>Social</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>625</td>
<td>4</td>
<td>Integer</td>
<td>Social</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>1728</td>
<td>6</td>
<td>Categorical</td>
<td>Business</td>
</tr>
<tr>
<td>User Knowledge Modeling</td>
<td>403</td>
<td>5</td>
<td>Real</td>
<td>Education</td>
</tr>
<tr>
<td>Contraceptive Method Choice</td>
<td>1473</td>
<td>7</td>
<td>Categorical, Integer</td>
<td>Life</td>
</tr>
<tr>
<td>Ecoli</td>
<td>336</td>
<td>7</td>
<td>Real</td>
<td>Life</td>
</tr>
<tr>
<td>Seeds</td>
<td>210</td>
<td>7</td>
<td>Real</td>
<td>Life</td>
</tr>
<tr>
<td>Draw</td>
<td>531</td>
<td>400</td>
<td>Integer</td>
<td>images</td>
</tr>
</tbody>
</table>

Table 3.1: Characteristics of datasets

Figure 3.10 illustrates an example of dataset file before encoding.

![Car dataset file before encoding](image)

Figure 3.10: Car dataset file before encoding
3.5 Data encoding

Neural networks natively process numerical data, so the categorical data must be encoded to numeric values. Each categorical attribute is encoded depending on the number of categories, so if we take the last attribute that is the class. This class has four categories which are unacc, acc, good, and vgood and after encoding will be (0 0 0 1), (0 0 1 0), (1 0 0 0) and (0 1 0 0). Figure 3.11 displays the encoding values of the class categories.

![Figure 3.11: Encoding data](image)

3.6 Network structure

This step determines the structure of the ANNs depending on the used dataset. The structure of the ANNs contains the number of inputs in input layer reflecting the number of attributes after encoding, the number of hidden layers and the number of outputs after encoding. Since there is no mature theory of neural network to determine the number of hidden layers, the present more effective method is test method (Hong-tao & Xiu-lan, 2011), so we used the WEKA program to find out the best number of hidden nodes for each dataset using this method. To be able to deal with the WEKA program, we first converted the dataset file format from txt
to csv or arff. The dataset has been split into two parts; 80% for training and 20% for testing. Figure 3.12 illustrates the results of the best number of hidden nodes of Car dataset (6 is the best number of hidden nodes for the Car dataset and 100% of classification accuracy was obtained).

![Weka Explorer](image)

*Figure 3.12: Best number of hidden nodes of Car dataset*
Table 3.2 shows the number of hidden nodes in the hidden layer and the network structure for each dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>no. of hidden nodes</th>
<th>Network structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass Identification</td>
<td>3</td>
<td>9-3-6</td>
</tr>
<tr>
<td>Nursery</td>
<td>7</td>
<td>26-7-5</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>5</td>
<td>4-5-3</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>6</td>
<td>21-6-4</td>
</tr>
<tr>
<td>User Knowledge Modeling</td>
<td>2</td>
<td>5-2-4</td>
</tr>
<tr>
<td>Contraceptive Method Choice</td>
<td>10</td>
<td>7-10-3</td>
</tr>
<tr>
<td>Ecoli</td>
<td>8</td>
<td>7-8-8</td>
</tr>
<tr>
<td>Seeds</td>
<td>2</td>
<td>7-2-3</td>
</tr>
<tr>
<td>Draw</td>
<td>9</td>
<td>400-9-3</td>
</tr>
</tbody>
</table>

*Table 3.2: Number of hidden layers & Network structure for datasets*

In each dataset, we create population of ten ANNs with the same structure and different weights. Backpropagation algorithm is applied for each ANN. The obtained ANN after backpropagation algorithm is used as chromosomes in the GA population if the solution has not been reached by backpropagation. Then, the half of the population which give best fitness values will be part of the PSO population if the solution has not been reached by GA. GA and PSO algorithms will be applied repeatedly with the half size of the previous population until the optimum solution is reached.
3.7 Parameters

The parameters used for GA and PSO are shown in Table 3.3.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP, GA, PSO</td>
<td>Maximum number of iterations = 2000</td>
<td>Controls how much each weight and bias value can change in each update step. Larger values increase the speed of training at the risk of overshooting optimal weight values.</td>
</tr>
<tr>
<td>BP, GA, PSO</td>
<td>minimum error = 0.0001</td>
<td></td>
</tr>
<tr>
<td>GA, PSO</td>
<td>Population size=10</td>
<td></td>
</tr>
<tr>
<td>BP</td>
<td>Learning rate = 0.05</td>
<td>Helps prevent training from getting stuck with local, non-optimal weight values and also prevents oscillation where training never converges to stable values.</td>
</tr>
<tr>
<td>BP</td>
<td>Momentum = 0.01</td>
<td>Helps prevent training from getting stuck with local, non-optimal weight values and also prevents oscillation where training never converges to stable values.</td>
</tr>
<tr>
<td>GA</td>
<td>mutateRate = 0.20</td>
<td>Controls how many genes in a newly-generated child's chromosome will be mutated.</td>
</tr>
<tr>
<td>GA</td>
<td>mutateChange = 0.01</td>
<td>Controls the magnitude of the change of mutated genes.</td>
</tr>
<tr>
<td>GA</td>
<td>tau = 0.40</td>
<td>This variable is the &quot;selection pressure&quot; and controls the likelihood that the two best individuals in the population will be selected as parents for reproduction.</td>
</tr>
<tr>
<td>PSO</td>
<td>probDeath = 0.005</td>
<td>This variable is set to a small value and is used to randomly kill and then regenerate a particle. The death-birth process is optional in PSO.</td>
</tr>
<tr>
<td>PSO</td>
<td>inertia weight = 0.729</td>
<td>Determines the influence of the current velocity.</td>
</tr>
<tr>
<td>PSO</td>
<td>c1=1.49445</td>
<td>This variable is called cognitive weights and determine the influence of the particle's best position.</td>
</tr>
<tr>
<td>PSO</td>
<td>c2=1.49445</td>
<td>This variable is called social weights and determine the influence of the global best-known position. The values used here have been suggested by PSO research.</td>
</tr>
<tr>
<td>PSO</td>
<td>r1 and r2 are random values in the range [0, 1)</td>
<td>Values that add a randomization effect to a particle's movement, which helps to prevent the particle from becoming stuck in a dead end.</td>
</tr>
</tbody>
</table>

*Table 3.3: Proposed algorithm's parameters*
3.8 Tools, Equipment and Methods

➢ Tools:
  - WEKA software is used for choosing best hidden nodes of neural network structure
  - Microsoft Word for document typing.
  - Microsoft Excel for spreadsheets.
  - Microsoft Visio for draw methodology flowchart.
  - Microsoft visual studio 2013 IDE for implementation of the proposed algorithm.
  - C# programming language.

➢ Equipment:
  - Lenovo IdeaPad U310 Laptop with Intel Core i5 (3rd Gen) 3317U / 1.7 GHz and 4 GB RAM.
Chapter 4: Experimental Results and Evaluation

This chapter discusses the results of experiments with a comparison between our proposed approach and other approaches.

To evaluate our approach, we present the obtained results based on three categories. First, ANN without any optimization algorithm, second, ANN supported with the proposed approach and finally ANN supported with GA alone, PSO alone, or GA followed by PSO.

4.1 Learning Based ANN

In this experiment, we use datasets that have been mentioned in the methodology’s chapter with ANN without the support of any optimization algorithm. The obtained results are shown in Table 4.1.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Accuracy Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass Identification</td>
<td>34.89</td>
</tr>
<tr>
<td>Nursery</td>
<td>91.89</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>84.8</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>70.81</td>
</tr>
<tr>
<td>User Knowledge Modeling</td>
<td>72.84</td>
</tr>
<tr>
<td>Contraceptive Method Choice</td>
<td>15.25</td>
</tr>
<tr>
<td>Ecoli</td>
<td>57.35</td>
</tr>
<tr>
<td>Seeds</td>
<td>73.81</td>
</tr>
<tr>
<td>Draw</td>
<td>60.75</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>62.49</strong></td>
</tr>
</tbody>
</table>

As noticed from Table 4.1, ANN alone gave mostly good results. In some cases, we obtained very bad accuracy such as in Contraceptive Method Choice dataset. For Glass Identification, we get also low classification accuracy. For the remaining datasets, we still get good accuracy.
Low accuracy might occur due to the variations in the instances of the dataset and relatively the small size of the datasets considering the number of attributes.

### 4.2 The proposed approach

In this case, we applied our proposed approach on the datasets given in Table 4.1, and the results are shown in Table 4.2.

**Table 4.2: Comparison between ANN results and our approach results**

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>ANN accuracy percentage</th>
<th>Our approach percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass Identification</td>
<td>34.89</td>
<td>62.79</td>
</tr>
<tr>
<td>Nursery</td>
<td>91.89</td>
<td>100</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>84.8</td>
<td>98.4</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>70.81</td>
<td>95.84</td>
</tr>
<tr>
<td>User Knowledge Modeling</td>
<td>72.84</td>
<td>96.3</td>
</tr>
<tr>
<td>Contraceptive Method Choice</td>
<td>15.25</td>
<td>21.02</td>
</tr>
<tr>
<td>Ecoli</td>
<td>57.35</td>
<td>86.76</td>
</tr>
<tr>
<td>Seeds</td>
<td>73.81</td>
<td>97.63</td>
</tr>
<tr>
<td>Draw</td>
<td>60.75</td>
<td>87.85</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>62.49</strong></td>
<td><strong>82.95</strong></td>
</tr>
</tbody>
</table>

As shown in Table 4.2, the proposed approach could improve the classification accuracy of ANN alone. This can be even noticed in cases where the ANN gets high accuracy such as in Nursery and Balance Scale datasets. Also for low accuracy with ANN, the proposed approach could improve the accuracy, for example with Contraceptive Method Choice and Glass Identification. The accuracy improvement is not much but this could be of two reasons, in case of very low accuracy, the number of instances in the dataset is not enough and there is a variation between the instances, which will make the generalization a difficult task. For the case of high accuracy, only marginal possible improvement can be obtained because this means also that ANN alone is performing well.
4.3 Other Approaches

In this part, we use the same datasets for various methods to compare their results with our proposed approach. The considered methods are GA alone, ANN with GA, ANN with PSO and GA followed by PSO. In GA alone, a chromosome represents weights of ANN generated randomly, and by only using genetic operators, GA will find out the best chromosome that solve the problem. In ANN with GA, backpropagation algorithm is applied first and then followed by GA to improve the obtained weight. Using PSO alone, every particle is an ANN weights generated randomly and the best position is tried to be obtained within all particles in a swarm. ANN with PSO is applying ANN with backpropagation at the beginning then PSO is applied to improve the classification accuracy. GA followed by PSO is performed by applying GA and then PSO without applying backpropagation algorithm. The last obtained population by GA is used by PSO as its initial population. The obtained results are shown in Table 4.3.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>GA</th>
<th>ANN+GA</th>
<th>PSO</th>
<th>ANN+PSO</th>
<th>GA+PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass Identification</td>
<td>37.21</td>
<td>51.16</td>
<td>41.86</td>
<td>53.49</td>
<td>58.14</td>
</tr>
<tr>
<td>Nursery</td>
<td>69.56</td>
<td>92.17</td>
<td>80.9</td>
<td>93.48</td>
<td>97.49</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>83.2</td>
<td>86.4</td>
<td>86.4</td>
<td>87.2</td>
<td>96</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>61.85</td>
<td>78.32</td>
<td>69.65</td>
<td>92.49</td>
<td>95.95</td>
</tr>
<tr>
<td>User Knowledge Modeling</td>
<td>66.67</td>
<td>76.54</td>
<td>82.72</td>
<td>83.83</td>
<td>95.06</td>
</tr>
<tr>
<td>Contraceptive Method Choice</td>
<td>0.34</td>
<td>17.65</td>
<td>0.68</td>
<td>18.32</td>
<td>18.98</td>
</tr>
<tr>
<td>Ecoli</td>
<td>63.24</td>
<td>76.47</td>
<td>67.65</td>
<td>80.82</td>
<td>82.35</td>
</tr>
<tr>
<td>Seeds</td>
<td>88.1</td>
<td>92.86</td>
<td>90.48</td>
<td>94.62</td>
<td>95.24</td>
</tr>
<tr>
<td>Draw</td>
<td>61.59</td>
<td>68.88</td>
<td>63.55</td>
<td>70.09</td>
<td>73.83</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>59.08</strong></td>
<td><strong>71.16</strong></td>
<td><strong>64.88</strong></td>
<td><strong>74.93</strong></td>
<td><strong>79.23</strong></td>
</tr>
</tbody>
</table>

It is noticed from the results shown in Table 4.3 that GA followed by PSO gives the best results among other approaches. It is to be noticed that our proposed approach outperforms noticeably all other methods including the GA followed by PSO.
The charts from 4.1 to 4.9 give a comparison between the results of every dataset for all the methods including our approach.

**Figure 4.1:** A comparison between the results of the Glass Identification dataset

**Figure 4.2:** A comparison between the results of the Nursery dataset
Figure 4.3: A comparison between the results of the Balance Scale dataset

Figure 4.4: A comparison between the results of the Car dataset
Figure 4.5: A comparison between the results of the User Knowledge Modeling dataset

Figure 4.6: A comparison between the results of the Contraceptive Method Choice dataset
Figure 4.7: A comparison between the results of the Ecoli dataset

Figure 4.8: A comparison between the results of the Seeds dataset
4.4 Discussion

We can summarize the obtained results as follows:

1- The overall results show that the proposed approach is better than all other approaches.
2- In step one of the testing, we have got lower accuracy percentage like in Glass and Ecoli datasets. The reason for that is the nature of data. To justify this, we tried to decrease number of attributes and increase number of instances of the Glass dataset and we got accuracy percentage equal 97%.
3- GA followed by PSO performance is better than GA alone and PSO alone as discussed in (Chen, et al. 2008), (Kuo, et al. 2009) and (Caputo, et al. 2010).
4- ANN with GA performance is better than GA as discussed in (Golmakani, et al. 2007).
5- ANN with PSO performance is better than PSO as discussed in (Shayeghi, et al. 2010).
6- ANN with PSO performance is better than ANN with GA as discussed in (Han, et al. 2009) (N, et al. 2013).
7- GA followed by PSO performance is better than both GA with ANN and PSO with ANN as discussed in (Yu-liang, et al. 2012).
8- PSO performance is better than GA as discussed in (E.A., et al. 2004).
Chapter 5 : Conclusion and Future Directions

5.1 Conclusion

In this thesis we proposed an approach based on both PSO and GA to increase the classification accuracy in ANN and to overcome the problem of local minima regardless of the given domain.

The proposed approach starts with finding out the best ANN using backpropagation algorithm through various attempts. The best obtained ANN will be used as one of the population for the GA algorithm. If the solution is still not reached, PSO algorithm will start working with the half population with the best fitness values. The process of keeping repeatedly applying GA followed by PSO with every time half of the last population with the best fitness values will be applied until the optimum solution is obtained.

To test and evaluate the proposed approach, nine datasets obtained from UCI with various domains and characteristics have been used. The testing was divided in three steps which are applying the datasets on ANN without any optimization algorithms, applying datasets on ANN using the proposed approach and finally applying datasets on some methods presented in previous researches; GA alone, ANN followed by GA, PSO alone, ANN followed by PSO and GA followed by PSO.

The results show the superiority and the capability of our proposed approach for all the datasets to increase the classification accuracy. It has been noticed that, in cases when other algorithms perform so bad, our proposed approach could clearly improve the results, examples for this are Glass Identification, Ecoli, Seeds, Car Evaluation, Contraceptive Method Choice, Draw and User Knowledge Modeling datasets. Also in cases when other approaches perform well, our approach was still able to improve the classification accuracy, examples for this are Nursery and Balance Scale datasets. It is also worth to notice that the best approach after our approach was applying GA followed by PSO, but none of its results is better than the one obtained by our approach. On contrary, our proposed approach obtained higher accuracy than the GA followed by PSO approach, for example the accuracy percentage obtained by GA followed by PSO are 58.14, 97.49, 96, 95.95, 95.06, 18.98, 82.35, 95.24 and 73.83 for Glass Identification, Nursery, Balance Scale, Car Evaluation, User Knowledge Modeling, Contraceptive Method Choice, Ecoli, Seeds and Draw datasets respectively, whereas for the
same datasets in the same order using our proposed approach, we got accuracy of percentage 62.79, 100, 98.40, 95.84, 96.30, 21.02, 86.76, 97.63 and 87.85.

5.2 Future Directions
Based on our current research, some of the future directions could be:

1. Applying the proposed approach with a different neural network approach like Radial-Basis Function (RBF).
2. Applying more datasets on the proposed approach.
3. Try other swarm intelligence approaches like Ant Colony Optimization (ACO) and Artificial Bees Colony (ABC) instead of using PSO, and investigate the obtained results.
References


• Han, M., Fan, J., & Han, B. (2009). An Adaptive Dynamic Evolution Feedforward Neural Network on Modified Particle Swarm Optimization. *International Joint Conference on Neural Networks*, (pp. 1083-1089).


Appendix A

Data Coding:
The following class encodes the non-numeric data to numeric values, because the neural network deals only with numeric data. In addition to the necessity of encoding non-numeric data, it also normalizes numeric data.

class NormalizeData
{
    public static double[][] Normalize(string[][] rawData)
    {
        //Nursery dataset
        //this array contains the types of each attribute inside the dataset
        string[] colTypes = new string[9] { "categorical", "categorical", "categorical", "categorical", "categorical", "categorical", "binary", "categorical", "categorical" };

        //Transform method that transform all types of data to numeric values.
        double[][] nnData = Transform(rawData, colTypes);

        return nnData; //return data after transformation.
    }

    static double[][] Transform(string[][] data, string[] colTypes)
    {
        //Scanning tokenized data to extract distinct values
        //ex. The class attribute has five distinct values which are (recommend, priority, not_recom, very_recom and spec_prior)
        string[][] distinctValues = GetValues(data, colTypes);
        Console.WriteLine("\nDistinct values:\n");
        ShowDistinctValues(distinctValues); //show the distinct values in each column

        //Computing number of columns for result matrix, because categorical encoding needs additional columns
        //ex: the class attribute in Nursary dataset contains five distinct values that means each value transform from one column to five columns
        //recommend will be transformed to (0.0 0.0 0.0 0.0 0.1)
        int extraCols = NumNewCols(distinctValues, colTypes);

        double[][] result = new double[data.Length][];
        for (int i = 0; i < result.Length; ++i)
            result[i] = new double[data[0].Length + extraCols];
    }
}
// Computing means and standard deviations of numeric data
double[] means = GetMeans(data, colTypes);
double[] stdDevs = GetStdDevs(data, colTypes, means);
for (int row = 0; row < data.Length; ++row)
{
    int k = 0; // walk across result cols
    for (int col = 0; col < data[row].Length; ++col)
    {
        string val = data[row][col];
        bool isBinary = (colTypes[col] == "binary");
        bool isCategorical = (colTypes[col] == "categorical");
        bool isNumeric = (colTypes[col] == "numeric");
        bool isIndependent = (col < data[0].Length - 1);
        bool isDependent = (col == data[0].Length - 1);

        if (isBinary && isIndependent) // binary x value -> -1.0 or +1.0
        {
            result[row][k++] = BinaryIndepenToValue(val, col, distinctValues);
        }
        else if (isBinary && isDependent) // binary y value -> 0.0 or 1.0
        {
            result[row][k] = BinaryDepenToValue(val, col, distinctValues); // no k++
        }
        else if (isCategorical && isIndependent) // cat x value -> [0.0, 1.0, 1.0] or [-1.0, -1.0, -1.0]
        {
            double[] vals = CatIndepenToValues(val, col, distinctValues);
            for (int j = 0; j < vals.Length; ++j)
            {
                result[row][k++] = vals[j];
            }
        }
        else if (isCategorical && isDependent) // cat y value -> [1.0, 0.0, 0.0]
        {
            double[] vals = CatDepenToValues(val, col, distinctValues);
            for (int j = 0; j < vals.Length; ++j)
            {
                result[row][k++] = vals[j];
            }
        }
        else if (isNumeric && isIndependent)
        {
            result[row][k++] = NumIndepenToValue(val, col, means, stdDevs);
        }
        else if (isNumeric && isDependent)
        {
            result[row][k] = double.Parse(val); // no k++
        }
    } // each col
} // each row
return result;
// binary x value -> -1 or +1
static double BinaryIndepenToValue(string val, int col, string[][] distinctValues)
{
    if (distinctValues[col].Length != 2)
        throw new Exception("Binary x data only 2 values allowed");
    if (distinctValues[col][0] == val)
        return -1.0;
    else
        return +1.0;
}

// y value -> 0 or 1
static double BinaryDepenToValue(string val, int col, string[][] distinctValues)
{
    if (distinctValues[col].Length != 2)
        throw new Exception("Binary y data only 2 values allowed");
    if (distinctValues[col][0] == val)
        return 0.0;
    else
        return 1.0;
}

// categorical x value -> 1-of-(C-1) effects encoding
static double[] CatIndepenToValues(string val, int col, string[][] distinctValues)
{
    if (distinctValues[col].Length == 2)
        throw new Exception("Categorical x data only 1, 3+ values allowed");
    int size = distinctValues[col].Length;
    double[] result = new double[size];

    int idx = 0;
    for (int i = 0; i < size; ++i)
    {
        if (distinctValues[col][i] == val)
        {
            idx = i; break;
        }
    }

    if (idx == size - 1) // the value is the last one so use effects encoding
    {
        for (int i = 0; i < size; ++i) // ex: [-1.0, -1.0, -1.0]
        {
            result[i] = -1.0;
        }
    }
    else // value is not last
    {
result[result.Length - 1 - idx] = +1.0; // ex: [0.0, 1.0, 0.0]

return result;

// categorical y value -> 1-of-C dummy encoding
static double[] CatDepenToValues(string val, int col, string[][] distinctValues)
{
    if (distinctValues[col].Length == 2)
        throw new ArgumentException("Categorical x data only 1, 3+ values allowed");
    int size = distinctValues[col].Length;
    double[] result = new double[size];

    int idx = 0;
    for (int i = 0; i < size; ++i)
    {
        if (distinctValues[col][i] == val)
        {
            idx = i; break;
        }
    }
    result[result.Length - 1 - idx] = 1.0; // ex: [0.0, 1.0, 0.0]
    return result;
}

// numeric x value -> (x - m) / s, this technique is sometimes called Gaussian normalization.
static double NumIndepenToValue(string val, int col, double[] means, double[] stdDevs)
{
    double x = double.Parse(val);
    double m = means[col];
    double sd = stdDevs[col];
    return (x - m) / sd;
}

// number of additional columns needed due to categorical encoding
static int NumNewCols(string[][] distinctValues, string[] colTypes)
{
    int result = 0;
    for (int i = 0; i < colTypes.Length; ++i)
    {
        if (colTypes[i] == "categorical")
        {
            int numCatValues = distinctValues[i].Length;
            result += (numCatValues - 1);
        }
    }
    return result;
}
// examine tokenized data to get distinct values for cat and binary columns
static string[][] GetValues(string[][] data, string[] colTypes)
{
    int numCols = data[0].Length;
    string[][] result = new string[numCols][];
    for (int col = 0; col < numCols; ++col)
    {
        if (colTypes[col] == "numeric")
        {
            result[col] = new string[] { "(numeric)" ];
        }
        else
        {
            Dictionary<string, bool> d = new Dictionary<string, bool>();
            for (int row = 0; row < data.Length; ++row)
            {
                string currVal = data[row][col];
                if (d.ContainsKey(currVal) == false)
                    d.Add(currVal, true);
            }
            result[col] = new string[d.Count];
            int k = 0;
            foreach (string val in d.Keys)
            result[col][k++] = val;
        }
    }
    return result;
}

// calculate the mean of each column
// mean = (summation of all values in a column)/no of values in a column
static double[] GetMeans(string[][] data, string[] colTypes)
{
    double[] result = new double[data.Length];
    for (int col = 0; col < data[0].Length; ++col) // each column
    {
        if (colTypes[col] != "numeric") continue; // curr col is not numeric
        Console.WriteLine("col:" + col);
        double sum = 0.0;
        for (int row = 0; row < data.Length; ++row)
        {
            double val = double.Parse(data[row][col]);
            sum += val;
        }
        result[col] = sum / data.Length;
    }
}
// calculate the standard deviation of each column
// Standard deviation = \sqrt{\frac{\sum (each\ value\ in\ a\ column - mean)^2}{no\ of\ values\ in\ a\ column}}
static double[] GetStdDevs(string[][] data, string[] colTypes, double[] means)
{
    double[] result = new double[data.Length];
    for (int col = 0; col < data[0].Length; ++col) // each column
    {
        if (colTypes[col] != "numeric") continue; // curr col is not numeric
        double sum = 0.0;
        for (int row = 0; row < data.Length; ++row)
        {
            double val = double.Parse(data[row][col]);
            sum += (val - means[col]) * (val - means[col]);
        }
        result[col] = Math.Sqrt(sum / data.Length);
    }
    return result;
}

// Method to print the distinct values in each attribute.
static void ShowDistinctValues(string[][] distinctValues)
{
    for (int i = 0; i < distinctValues.Length; ++i)
    {
        Console.Write("[" + i + "] ");
        for (int j = 0; j < distinctValues[i].Length; ++j)
        {
            Console.Write(distinctValues[i][j] + " ");
        }
        Console.WriteLine("\n");
    }
}

The algorithm:

This class is main part of the algorithm.

class Program
{
    static void Main(string[] args)
    {
}
Console.WriteLine("nBegin neural network training with back-propagation demo\n");

int n = 10; // no of ANN
double[][] lastWeights = new double[n][];
int counter = 0;

string line;

// Files

// nursery DataSet file
string file_name = Environment.CurrentDirectory + @"\DataSets\nursery.txt";

// Read the file
int lineCount = File.ReadLines(file_name).Count();
string[][] rawData = new string[lineCount][];

System.IO.StreamReader file = new System.IO.StreamReader(file_name);
int first_line = File.ReadLines(file_name).Take(1).First().ToString().Split(',').Length;
while ((line = file.ReadLine()) != null)
{
    // display data line by line.
    rawData[counter] = new string[first_line];
    rawData[counter] = line.Split(',');
    counter++;
}

file.Close();

// normalize the data to used it with the neural network
double[][] allData = NormalizeData.Normalize(rawData);
System.Console.WriteLine("There were {0} lines.", counter);

Console.WriteLine("nCreating 80% training and 20% test data matrices");

double[][] trainData = null;
double[][] testData = null;

// Creating 80% training and 20% test data matrices
MakeTrainTest(allData, out trainData, out testData);

// network structure for the nursery Dataset
const int numInput = 26;
const int numHidden = 7;
const int numOutput = 5;

// training parameters specific to GA
int popSize = n;
double exitError = 0.0001;
double mutateRate = 0.20;
double mutateChange = 0.01;
```csharp
double tau = 0.40;
double best_error = 0;

// training parameters specific to BP
int maxEpochs = 2000;
double learnRate = 0.05;
double momentum = 0.01;

// training parameters specific to PSO
double probDeath = 0.005;

// array of the weights
double[,] weights = new double[NeuralNetworkTraining.numWeights];

Console.WriteLine("nCreating a " + numInput + "," + numHidden + "," + numOutput + ",output neural network");

NeuralNetworkTraining nn = new NeuralNetworkTraining(numInput, numHidden, numOutput);

Console.WriteLine("nInitializing weights and bias to small random values");

// This function to initialize the values of weights.
double[[,] intializeweights = nn.InitializeWeights(n);

Console.WriteLine("Setting maxEpochs = 2000, learnRate = 0.05, momentum = 0.01");

bool find_sol = false;
List<double> list1 = new List<double>();

// This array save the values of minimum error of each ANN
double[,] mse = new double[n];

// This array save the values of fitness value of each ANN
double[,] fitness = new double[n];

for (int g = 0; g < n; ++g)
{
    nn.SetWeights(intializeweights[g]);
    // Apply a Backpropagation algorithm in each ANN
    find_sol = nn.TrainBP(intializeweights[g], trainData, maxEpochs, learnRate, momentum, exitError);
    // if the solution is found, stop
    if (find_sol == true)
    {
        Console.WriteLine("nTraining complete and the solution has been found by BP algorithm");
        // get the final neural network weights.
        weights = nn.GetWeights();
        Console.WriteLine("Final neural network weights and bias values.");
    }

    mse[g] = exitError;
    fitness[g] = nn.GetFitness();
}
```

56
ShowVector(weights, 10, 3, true); // print the final neural network weights.

Console.WriteLine("\nThe test data length is " + testData.Length);
double testAcc = nn.Accuracy(testData); // calculate the classification accuracy of ANN

// printing the classification accuracy of ANN
Console.WriteLine("\nClassification accuracy = " + testAcc.ToString("F4");
Console.WriteLine("\nEnd neural network training with back-propagation demo\n");
break;
} else {
    // save the min error in the mse array
    mse[g] = NeuralNetworkTraining.mse_nosol;
    // calculate the fitness value of each ANN and save it in the fitness array
    fitness[g] = nn.Accuracy(testData);
}

int count = 0;
if (find_sol == false) {
    // get the min error
    double min_mse = mse[0];
    for (int i = 1; i <= mse.Length - 1; i++){
        if (mse[i] < min_mse)
            min_mse = mse[i];
    best_error = min_mse;
    Console.WriteLine("the min error is: " + min_mse);
    // the index of min error
    int index = Array.IndexOf(mse, min_mse);
    // get the best fitness value
    double best_fitness = fitness[0];
    for (int i = 1; i <= fitness.Length - 1; i++){
        if (fitness[i] > best_fitness)
            best_fitness = fitness[i];
    Console.WriteLine("the best fitness value is: " + fitness[index]);
    // the index of best fitness value
    // save the weights that gives the minimum error among the ANNs
    Array.Copy(initializeWeights[index], 0, NeuralNetworkTraining.weights_minerr, 0,
    NeuralNetworkTraining.weights_minerr.Length);

    // starting with GA
    GA:
    Console.WriteLine("Beginning training using GA because there is no solution with BP");
    // adding the best weights as a chromosome in the GA population.
}
find_sol = nn.TrainGA(trainData, popSize, maxEpochs, exitError, mutateRate, mutateChange, tau, false);

if (find_sol == true)
{
    //if the min error has been reached, then here the algorithm is finished.
    Console.WriteLine("Training complete");
    Console.WriteLine("The best classifier:");

    //print the final weights and calculate the classification accuracy.
    ShowVector(NeuralNetworkTraining.GA_bestSolution, 10, 3, true);
    double testAcc = nn.Accuracy(testData);
    Console.WriteLine("Accuracy on test data = ");
    Console.WriteLine(testAcc.ToString("F4"));
}
else
{
    double testAcc = nn.Accuracy(testData);
    list1.Add(testAcc);
    Console.WriteLine(testAcc);
    if (testAcc > best_fitness)
    {
        best_fitness = testAcc;
        best_error = NeuralNetworkTraining.GA_bestError;
        Console.WriteLine("The best classifier:");
        ShowVector(NeuralNetworkTraining.GA_bestSolution, 10, 3, true);
        Console.WriteLine("Error = " + best_error);
        Console.WriteLine("Accuracy on test data = ");
        Console.WriteLine(testAcc.ToString("F4"));
        Array.Copy(NeuralNetworkTraining.weights_minerr, 0, NeuralNetworkTraining.weights_minerr.Length);
    }
    else
    {
        //starting with PSO if we don’t reach the desired error.
        Console.WriteLine("The training by using GA does not give a solution, start with PSO");
        Population[] pop = new Population[n];

        Outer:
        //get the best half weights from GA
        getHalfWeights(fitness, pop, nn, testData);

        PSO:
        //apply PSO
        Console.WriteLine("Beginning training using PSO");
        find_sol = nn.TrainPSO(trainData, popSize, maxEpochs, exitError, probDeath);
        if (find_sol == true)
{ Console.WriteLine("Training complete");
    Console.WriteLine("The best classifier:");
    ShowVector(NeuralNetworkTraining.PSO_bestSolution, 10, 3, true);
    testAcc = nn.Accuracy(testData);
    Console.WriteLine("Accuracy on test data = " + testAcc.ToString("F4");
    Console.WriteLine("End neural network training");
}
else //there is no result from PSO
{
    testAcc = nn.Accuracy(testData);
    list1.Add(testAcc);
    Console.WriteLine(testAcc);
    if (testAcc > best_fitness)
    {
        best_fitness = testAcc;
        best_error = NeuralNetworkTraining.PSO_bestError;
        Console.WriteLine("The best classifier:");
        ShowVector(NeuralNetworkTraining.PSO_bestSolution, 10, 3, true);
        testAcc = nn.Accuracy(testData);
        Console.WriteLine("Error= " + best_error);
        testAcc = nn.Accuracy(testData);
        Console.WriteLine("Accuracy on test data = ");
        Console.WriteLine(testAcc.ToString("F4");
        getHalfWeights(fitness, pop, nn, testData);
        //PSO will be executed many times until we can’t get better than the best solution that have already reached.
        goto PSO;
    }
}

if (find_sol == true)
{
    Console.WriteLine("Training complete");
    Console.WriteLine("The best classifier:");
    ShowVector(NeuralNetworkTraining.GA_bestSolution, 10, 3, true);
    testAcc = nn.Accuracy(testData);
    Console.WriteLine("Accuracy on test data = ");
    Console.WriteLine(testAcc.ToString("F4");
}
else
{
}
testAcc = nn.Accuracy(testData);
list1.Add(testAcc);
Console.WriteLine(testAcc);
if (testAcc > best_fitness)
{
    best_fitness = testAcc;
    best_error = NeuralNetworkTraining.GA_bestError;
    Console.WriteLine("\nThe best classifier:");
    ShowVector(NeuralNetworkTraining.GA_bestSolution, 10, 3, true);
    Console.WriteLine("Error= " + best_error);
    testAcc = nn.Accuracy(testData);
    Console.Write("\nAccuracy on test data = ");
    Console.WriteLine(testAcc.ToString("F4"));
    //GA and PSO keep turning the population size repeatedly by half until the optimum solution is obtained.
    goto GA2;
}
else
{
    while (count<=30)
    {
        count++;
        goto Outer;
    }
    Console.WriteLine(list1.Max());
}
//end else
}
Console.ReadLine();
}// Main

// This function get the best half weights
static void getHalfWeights(double[] fitness, Population[] pop, NeuralNetworkTraining nn, double[][] testData)
{
    //calculate the fitness value of the GA weights
    for (int i = 0; i <= fitness.Length - 1; i++)
    {
        pop[i] = new Population(NeuralNetworkTraining.updated_weights[i].Length);
        Array.Copy(NeuralNetworkTraining.updated_weights[i], 0, pop[i].Weights, 0, NeuralNetworkTraining.numWeights);
nn.SetWeights(NeuralNetworkTraining.updated_weights[i]);
fitness[i] = nn.Accuracy(testData);
pop[i].Fitness = fitness[i];
}

// give the best half size of the weights
double[] arr_temp = new double[NeuralNetworkTraining.numWeights];
int x, j;
double tempValue;
for (x = 0; x <= pop.Length - 1; x++)
{
    for (j = x + 1; j <= pop.Length - 1; j++)
    {
        if (pop[j].Fitness > pop[x].Fitness)
        {
            tempValue = pop[j].Fitness;
            pop[j].Fitness = pop[x].Fitness;
            pop[x].Fitness = tempValue;
            Array.Copy(pop[x].Weights, 0, arr_temp, 0, NeuralNetworkTraining.numWeights);
            Array.Copy(pop[j].Weights, 0, pop[x].Weights, 0, NeuralNetworkTraining.numWeights);
            Array.Copy(arr_temp, 0, pop[j].Weights, 0, NeuralNetworkTraining.numWeights);
        }
    }
}

// save the best 5 weights that has best fitness value
for (int h = 0; h <= pop.Length - 6; h++)
{
    NeuralNetworkTraining.half_weights[h] = new double[NeuralNetworkTraining.numWeights];
    Array.Copy(pop[h].Weights, 0, NeuralNetworkTraining.half_weights[h], 0, NeuralNetworkTraining.numWeights);
}

// Making training and testing data by splitting them and initialize them
static void MakeTrainTest(double[]][] allData, out double[]][] trainData, out double[]][] testData)
{
    // split allData into 80% trainData and 20% testData
    Random rnd = new Random(0);
    int totRows = allData.Length;
    int numCols = allData[0].Length;

    int trainRows = (int)(totRows * 0.80); // hard-coded 80-20 split
    int testRows = totRows - trainRows;

    trainData = new double[trainRows][];
testData = new double[testRows][];

    int[] sequence = new int[totRows]; // create a random sequence of indexes
for (int i = 0; i < sequence.Length; ++i)
    sequence[i] = i;

for (int i = 0; i < sequence.Length; ++i)
{
    int r = rnd.Next(i, sequence.Length);
    int tmp = sequence[r];
    sequence[r] = sequence[i];
    sequence[i] = tmp;
}

int si = 0; // index into sequence[]
int j = 0; // index into trainData or testData

for (; si < trainRows; ++si) // first rows to train data
{
    trainData[j] = new double[numCols];
    int idx = sequence[si];
    Array.Copy(allData[idx], trainData[j], numCols);
    ++j;
}

j = 0; // reset to start of test data
for (; si < totRows; ++si) // remainder to test data
{
    testData[j] = new double[numCols];
    int idx = sequence[si];
    Array.Copy(allData[idx], testData[j], numCols);
    ++j;
}

// MakeTrainTest

//print final neural network weights and bias values
static void ShowVector(double[] vector, int valsPerRow, int decimals, bool newline)
{
    for (int i = 0; i < vector.Length; ++i)
    {
        if (i % valsPerRow == 0) Console.WriteLine();
        Console.Write(vector[i].ToString("F" + decimals).PadLeft(decimals + 4) + " ");
    }
    if (newline == true) Console.WriteLine();
}

} // class Program
**Initialize weights:**

This function initializes random values for the ANN weights

```csharp
public double[][] InitializeWeights(int n)
{
    // initialize weights and biases to small random values
    double[][] initialWeights = new double[n][];
    for (int x = 0; x < n; ++x)
    {
        initialWeights[x] = new double[numWeights];
        for (int i = 0; i < numWeights; ++i)
        {
            initialWeights[x][i] = 1 * (rnd.NextDouble() * 2 - 1); // values range [-1,1]
            return initialWeights;
        }
    }
}
```