Distributed Artificial Neural Network Training Using an Intelligent Agent

G.R.C De Silva & Asoka S. Karunananda

Informatics Institute of Technology,
In collaboration with Manchester Metropolitan University, UK

Faculty of Information Technology, University of Moratuwa,
Sri Lanka

Abstract

Obtaining the best configuration for an Artificial Neural Network (ANN) has always been problematic due to the non availability of a uniform method of finding the best configuration. The trial-and-error approach is used even today for this purpose regardless of its limitations. This project exploits intelligent Agent technology and distributed computing techniques to automate and streamline the task of ANN training. A static, centralized Agent autonomously generates and trains multiple ANN's in varying configurations for a given dataset. By analyzing the progress of each training session the Agent discovers which ANN configuration is best suited for the training data. The distributed system on which the Agent operates allows multiple ANN's to train concurrently which helps identify the best configuration in a much shorter timeframe. Knowledge on past ANN trainings is gathered by the Agent from which it learns to identify most suitable ANN configurations for subsequent training sessions. The end result is a fully automated system that can identify the most suitable ANN configuration for a given dataset quickly, efficiently and autonomously.

1. Introduction

ANN's have become an attractive Artificial Intelligence (AI) technique for building modern intelligent applications. Image recognition, sound recognition, financial forecasting etc. are popular application areas for ANN's. However training a Neural Network to understand the problem correctly takes considerable manual effort because various configuration parameters of an ANN need to be assigned proper values (i.e. network topology, activation function, learning rate etc.). Since these values differ depending on the problem, it requires a certain degree of experimentation before identifying which values are most suitable for these parameters. This is currently done manually by the ANN developer in an ad-hoc manner. This, together with the most widely used single machine based ANN training method makes it a time consuming process to identify and train a suitable ANN configuration for the problem.

Several techniques have been devised to overcome these limitations in both identifying the proper network parameters and also speeding up the ANN training process. Use of Genetic Algorithms (GA) and Evolutionary Computing to devise network parameters instead of manually identifying them is explained in [17] [18] [19] [20]. Research has also been carried out on developing specialized hardware to train neural networks which is much faster than the software based approach that is widely used [6].

The above mentioned papers propose techniques for addressing the limitations of ANN training. However their main focus is based either on identifying the proper configuration values for the ANN more efficiently or improving the speed of ANN training. This paper focuses on identifying a technique that can address both these issues where the manual nature of ANN training can be reduced to a minimum at the same time reducing the overall time taken to obtain a suitable ANN configuration for any given problem.

1.1. ANN Training: Practices and Issues

ANN training can be categorized into two modes as supervised and unsupervised training. The supervised method is most widely used, where the network is given the inputs as well as the desired outputs for each input. The network tries to produce the desired output for each input during training, where the difference of the desired and actual output

which is also referred to as the 'error' is propagated backwards through the network which modifies its connection weights according to a learning algorithm. Back-propagation learning rule is mostly used for this purpose because it's a general purpose algorithm which is applicable for many real world problems such as image recognition [1]. However it has its limitations such as the tendency for network paralysis and the long periods taken for ANN training [2].

Unsupervised mode is the other alternative, where the network tries to learn patterns in the data on its own without any given desired outputs. Application areas such as data classification, data mining and data compression use unsupervised training. However this mode of training is still not as popular as supervised training. Kohonen self organizing maps are commonly used for this purpose [3].

Identifying the best ANN configuration for the training data is the most challenging aspect when training ANN's in supervised mode. There are many parameters that need consideration such as the learning rule, activation function, network topology, learning rate, momentum, bias etc. Proper values for most of these parameters cannot be determined in advance, because they depend heavily on the nature of the training data. Therefore it requires a certain degree of experimentation to identify which values are most suitable for each parameter. For example, identifying the number of layers and hidden neurons for the network can only be done in an ad-hoc manner by trying out different architectures and observing the training progress of each [2]. This is perhaps the most time consuming task when training ANN's. The ANN developer needs to create the ANN configuration manually by means of a development tool or by directly coding it in a programming language. The created ANN then requires training, where it should be continuously monitored by the developer to decide whether the network is training successfully or not. Mostly, the initial configuration tested will not suffice and alternate configurations needs to be tested which results in repeating the same procedure all over again. Apart from that, each configuration that is tested needs to be kept track of by the developer to avoid re-training any configuration that has already been tested.

Clearly, this approach is inefficient and there is too much manual work involved. Also, since only a single computer is used for training ANN's, each configuration needs to be trained sequentially. As a result, it takes a considerable amount of time to identify the configuration that is most suitable for the network.

The rest of this paper discusses the current approaches and solutions used by ANN developers to overcome the limitations identified above. The focus will then move onto describing how Agent Technology and Distributed Computing can be used to devise a novel solution to address these issues. The remainder of the paper will discuss how this solution has been implemented into a functional prototype followed by the evaluation results of the proposed solution which will confirm the applicability of using Agent Technology and Distributed Computing for ANN training.

2. Current approaches & solutions

Several approaches can be seen as techniques used by ANN developers to improve the task of ANN training. Hierarchical and Modular ANN's [3] are used to break down the training into smaller components and aggregate the final results into a single network. This allows you to obtain the final network quicker than training a very large and complex network which may take longer to train.

Evolutionary computing techniques are used to devise the optimum architecture for a network. Neuro Evolution of Augmenting Topologies (NEAT) is one such method which uses an evolutionary process of obtaining the best ANN configuration [4]. Genetic Algorithms are also used during ANN training to evolve various network parameters such as connection weights [5].

Hardware based approaches are used to improve the speed of ANN training. Digital and analog hardware chips such as NeuraLogix NLX-620 and Micro Devices MD-1220 are some examples [6]. Hardware based techniques are capable of training ANN's much faster than software based implementations.

Among the many software toolkits available for ANN development, NeuroSolutions [7], Statistica Neural Networks [8], Neural Network Toolbox for MATLAB [9], JOONE Distributed Training Environment [10] are popular choices. The graphical user interfaces provided by most of these solutions make ANN development easier and less time consuming than the manual coding method.

Although the approaches discussed contribute to ANN development, they do not fully address the problems related to ANN training. The first problem is that creating different ANN configurations requires manual effort. Although these solutions provide feature rich user interfaces for this purpose, the creation of ANN's is still a manual process that needs to be performed by the user.

The remaining problem is the time factor. Almost all solutions discussed above use a single computer for training ANN's in a sequential manner. This becomes a problem as the number of configurations that require training becomes very large.

Yet another issue is the inability to keep track of ANN configurations that have already been tested. Current approaches tend to re-start the development of ANN's every time a new dataset is trained. There is very little support for re-using the knowledge gained from previous training sessions in current approaches which could contribute immensely to identifying suitable ANN configurations much sooner.

One final issue is the cost factor. Most of the solutions mentioned above are commercial products which come at a cost, and it is difficult to obtain such software for the typical academic student.

3. Approach: Agent Technology & Distributed computing

3.1 Suitability of Agent Technology

Looking at the tasks involved in developing and training ANN's, the concepts of Agent technology were identified as very promising for this purpose. The autonomous, reactive, proactive behavior of Agents [11] together with learning capability blends in well to address the problems faced while training ANN's.

During ANN training, the developer needs to experiment with various ANN configurations by creating, training them and analyzing their outcome. The autonomous features of Agents are ideal for this type of task, where the Agent can perform these operations on its own without user intervention. During training, the reactive nature of Agents becomes useful where the training may need to be halted and re-started (i.e the network becomes paralyzed during training). The learning abilities of Agents can be used to store vast amounts of knowledge gathered during ANN training such as which configurations are best suited for given datasets by classification. This information can be re-used later to identify which configurations are most suited for a particular dataset given.

We have taken a single Agent based approach rather than a multi Agent one, due to the fact that a centrally located Agent can control the entire distributed system at the same time keeping track of the status on remote machines from a single location. Also the need to aggregate results of all trainings for evaluation purposes needs to be done centrally, therefore a single central Agent was chosen for this purpose. Since the system can be controlled from a single location, the Agent need not move across different machines. Therefore a static Agent has been identified as more suitable rather than a mobile Agent.

3.2 Suitability of a Distributed System

In order to speed up the process of identifying the best ANN configuration, we have made use of a distributed system. Conventionally, ANN training is carried out in a sequential manner using a single computer for each configuration tested. Performing this operation on a distributed system enables training of varying ANN configurations concurrently on different machines, thus reducing the overall time taken to identify the best configuration for the training data. The system is scalable, which means new machines can be added dynamically during operation thus increasing the number of ANN configurations that can be concurrently trained. This type of configuration is ideal for a Local Area Network, where machines on the network are mostly sitting idle. The processing power of these machines can be harnessed for the task of ANN training.

3.3 Inputs, process and outputs of the system

The inputs to the system are the initial ANN configuration which includes parameters for Emax, network topology, learning algorithm, learning rate, cycles to train and other learning algorithm related parameters along with the training dataset will be in the form of a text file.

The process of the system will be creating and training various ANN configurations for the

training data and identifying which configuration performs best out of the entire collection of configurations tested.

The output of the system will be a fully trained ANN created to the most suitable configuration which can be readily deployed in a Java application using Java object serialization.

3.4 Features of the system

A host of features will be incorporated onto the system to facilitate the successful training and monitoring of ANN's.

The system will be capable of developing, training and evaluating various ANN configurations autonomously.

The system will be scalable, where client machines can be connected to the centralized Agent server during runtime increasing the processing capabilities of the system.

ANN training on each machine will be remotely administered by the intelligent Agent, and their progress will be displayed on the central server in a graphical format.

Resources on the distributed system will be efficiently utilized. No clients will be sitting idle when there is pending ANN training jobs at the same time, no client will be overloaded with ANN training jobs.

The Agent will be customizable, where the user can change various parameters to alter the behavior of the Agent during operation.

4. Design

The system consists of 3 main modules. Based on the client/server architecture [12], the Agent runs on a centralized server and other machines connect to this server as clients to acquire ANN training jobs.
The concept of ANN training using Agent Technology and Distributed Computing is depicted in Figure 1. The main modules of the system are shown and the communication direction between these modules is depicted using the arrows.

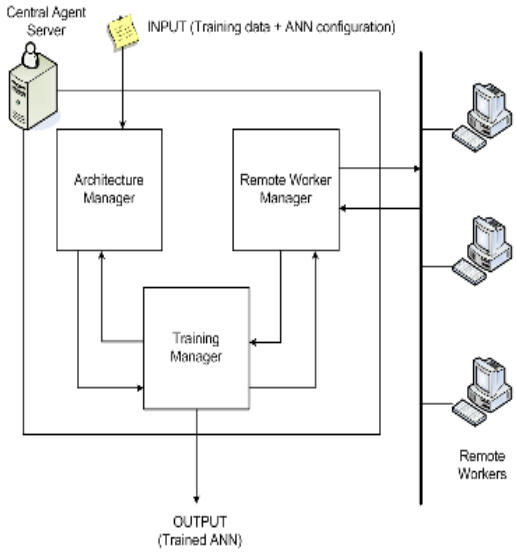

Figure 1: High-level System Design

4.1 Architecture Manager Module

An *ANN Architecture Manager Module* is developed to facilitate the creation of various ANN configurations. The parameters for the ANN such as number of hidden layers, neurons per hidden layer; activation function etc. can be configured by this component to create varying ANN configurations. These ANN's are created in memory as objects which can be transported across the network to remote machines for training.

4.2 Worker Manager Module

A *Worker Manager Module* is developed to manage the resources on the distributed system. A mechanism to keep track of which client machines are available to take ANN training jobs is required, which will be the main purpose of this module. Tasks such as accepting client registrations, de-registering clients from the server, keeping track of client details such as their hostname, IP address, machine type etc. will be maintained by this module.

4.3 Training Manager Module

Finally, a *Training Manager Module* manages all aspects of training the ANN. This is the Agent component. It sends ANN's to train on remote machines, monitors the training conditions, takes decisions based on the conditions and finally brings together the trained ANN's back to the central server. The module acts as the front end to the user and

integrates with the other 2 modules to create the final system.

5. Implementation

Bringing together the 3 main modules, the system has been implemented using Java as the core development language. Java qualifies as the language of choice for this system due to its platform independent nature which is ideal for the distributed system. Together with pure Java, several Java based open-source frameworks have also been used to develop the modules which are discussed below.

5.1 Architecture Manager Implementation

The *Architecture Manager* module is capable of creating ANN objects in varying configurations (i.e topology, activation function etc.) To facilitate this task, JOONE (Java Object Oriented Neural Engine) [10] which is an open-source framework written in Java has been used. It provides the base classes for creating ANN's through Java code. The customization components of the Architecture Manager module use this framework as the base for generating varying ANN configurations.

5.2 Worker Manager Implementation

The infrastructure required by the *Worker Manager* module for communication between the centralized Agent and remote PCs on the distributed system is implemented using Java RMI (Remote Method Invocation) technology. [13][14][15] Remote interfaces at both client and server ends enable communication, transportation and remote method invocation of ANN objects between server and clients. The object serialization capability in Java RMI contributes to passing ANN objects to remote machines and retrieving them as return values.

5.3 Training Manager Implementation

The *Training Manager* module has been developed using JADE (Java Agent Development Environment) [16] which is an open-source Agent platform written in Java. The various behaviors of the Agent are implemented using this framework, and it has been integrated with Worker Manager and Architecture Manager Modules to bring the system together. The Training Manager module also connects to a MS Access database, where it periodically queries and updates records of past ANN training information.

Using the tools and technologies mentioned above, a fully functional prototype has been developed to demonstrate the concept of distributed ANN training using Agent technology.

6. Evaluation

Following the successful implementation of the system, a formal evaluation was carried out to asses the outcome of the proposed system.

6.1 Evaluation criteria

The criteria for the evaluation were mainly the suitability of the approach taken by the authors to address the problems identified during ANN training. The accuracy of the proposed system, features it provides to the users, usability of the system and also the limitations of the system are among the evaluation criteria.

To evaluate the accuracy of ANN training of the prototype, 4 training datasets were used to test the system. 2 datasets were obtained from projects which involved image identification. 2 more datasets were obtained from projects that involved financial forecasting.

The system was tested on an operational Local Area Network (LAN) with a single machine running as the Agent server and 5 other machines on the LAN connected to the Agent server as worker machines to train the ANN's that are generated.

6.2 Evaluation audience and method

Several experts in the field of Artificial Neural Networks were chosen as expert evaluators. They are experienced lecturers on the subject of ANN's for many years at leading educational institutions in Sri Lanka both government and private. 3 subjects were chosen from the field of ANN as evaluators for this purpose. Apart from them, 2 subjects were chosen to evaluate the Distributed Component of the proposed solution. One of them has over 2 years industrial experience in Distributed Computing and the other over 8 years of experience in lecturing on Distributed Systems for undergraduate students.

Apart from the expert group, a selected group of 15 individuals were also selected as standard evaluators. These included both undergraduate and graduate students who have experienced developing and training Neural Networks during their final year research projects.

The evaluation process was carried out using formal interviews, questionnaires and a demonstration of the prototype.

6.3 Evaluation results

Using the training datasets explained above as inputs to the system, the system was tested on an operational LAN using 5 machines connected to the Agent server as workers to train the generated ANN's.

The ANN training results for the 1st dataset from the image recognition problem are given below. The dataset consists of 17 training patterns with 3 input elements and 3 desired output elements in each pattern. 10 different ANN configurations were created and trained using the conventional manual method and the proposed system. Each ANN configuration was trained for 10000 cycles. The dataset used for this training is depicted in Figure 2.

Using the conventional technique (manually creating and training ANN through code)

Total ANN training time (seconds)	1380
Lowest training error achieved	0.1372
Final ANN topology generated	3-8-4-3

Using the proposed system

Total ANN training time (seconds)	270
Lowest training error achieved	0.1301
Final ANN topology generated	3-7-4-3

```
0.02629959514;0.00851724137931035;0.0411666667;0;0;1
0.02354826254;0.00835483870967742;0.037;0;0;1
0.02254406130;0.0100384615384615;0.0435;0;0;1
0.02614574898;0.00851724137931035;0.0352857143;0;0;1
0.02036470588;0.00879310344827586;0.0425;0;0;1
0.02614574898;0.00851724137931035;0.0354257143;0;0;1
0.01963745502;0.00896428571428571;0.0352871429;0;0;1
0.02024014336;0.00962068965517241;0.06975;0;1;0
0.02468339768;0.00925;0.0863333333333333;0;1;0
0.02122180451;0.0102307692307692;0.0665;0;1;0
0.02434732824;0.0100769230769231;0.0655;0;1;0
0.01967669133;0.0095;0.0665;0;1;0
0.02155248387;0.00953846153846154;0.0826666667;0;1;0
0.03531666667;0.0141176470588235;0.024;1;0;0
0.03238618487;0.0132222222222222;0.0238;1;0;0
0.03685331746;0.01576;0.0315;1;0;0
0.0358;0.0153125;0.0222727272727273;1;0;0
```

Figure 2: Training Dataset

The significance when comparing the above results is the time period taken to train 10 different ANN configurations for the training data. As the results show, using an Agent to create the ANN's virtually eliminates the requirement for the user to create each ANN configuration. Because of that, generation of a new ANN configuration is almost instantaneous. This is in contrast to the manual method where it takes at least 20 seconds for the user to modify the previous configuration and create a newer ANN configuration to start training. This also needs to be done keeping in mind the configurations that were tested earlier to avoid re-training a previous configuration. Such tasks are all handled autonomously by the Agent.

The use of a Distributed System has clearly reduced the total training time for all ANN's drastically. As the results show, 5 different ANN configurations could be concurrently trained. This number can be dynamically increased by adding more machines as workers to the system at any point during operation to increase the capacity of training more ANN configurations concurrently.

Apart from the speed in obtaining the final ANN configuration and the autonomy of the system, the graphical reporting capability of the system should also be pointed out where the training error graph for each ANN configuration can be easily viewed using the user interface. Besides from this, the user has the option of saving a chosen number of ANN configurations that produces the lowest training error. Most of these facilities are not available in the conventional training method used.

The data classification ability of the system

should also be noted. The final configuration for the ANN was saved by the Agent to the database. When the system was introduced with the same dataset as a new training session, it was capable of classifying the dataset given using the Kohonen Self Organizing Map and identifying the previously successful ANN configuration of 3-7-4-3 as the first configuration to be trained. This sort of functionality is not available in the conventional ANN training methods discussed.

The overall results of the evaluation revealed that the system scored an eighty percent rating as 'Good', and around a twelve percent rating of 'Excellent' by the evaluators. The remainder fell under the 'Average' rating. The system was mostly appreciated due to the simplicity and ease of use. The scalability of the system and capability to run the client application on any platform was also seen very attractive by the evaluators. The graphical reporting of training progress was also very much appreciated.

Although the evaluators did not find any major drawbacks of the system, some useful additions to the system were suggested which included dynamic worker discovering capability and unsupervised ANN training support which are features that are not currently available in the proposed system.

From the authors' point of view, the system was originally developed to automate and make more efficient the task of ANN training. We feel that this objective has been met where the system is capable of identifying a suitable ANN configuration on its own without the user's intervention.

However it should be pointed out that the solution provided by the system may not be the 'optimum' solution, because there is no method of measuring if a given ANN configuration is the best configuration for the training data. However, considering the vast numbers of ANN configurations that the system is capable of testing, there is much greater capacity to identify a suitable ANN configuration for the training data because testing such large numbers of ANN configurations is not practical in the conventional manual approach of ANN training.

7. Discussion & Conclusion

We have made use of Agent technology and a distributed system as an approach to improve and streamline the process of ANN training. The various features of Agents and how they are incorporated into the ANN training process were discussed throughout this paper. We also pointed out how a distributed system could improve the time taken to identify a suitable ANN configuration. The design and implementation of the proposed system was later discussed pointing out the different modules of the system and the technologies used by each module.

Based on the capabilities of the system we believe that this is a good solution to train ANN's autonomously with minimum user involvement. The main strength of this system lies in its ability to generate ANN's, train and monitor their training and also make decisions during the training period all without the user's intervention. The amount of manual work that is saved by using this system is therefore very high. Also, the distributed system helps identify a suitable ANN configuration much faster than the sequential training methods discussed. As new machines can be dynamically connected to the Agent server to train ANN's, the overall time taken to obtain the final result can be further reduced. The data classification abilities of the system and learning capability allow it to intelligently identify possible ANN configurations that may be best suited for the given dataset. The final output of the system can be readily used as a production Neural Network in any Java application together with the JOONE framework.

Overall we feel that the implemented system is capable of identifying a suitable ANN configuration autonomously in a fast and efficient manner. We also feel that the system has potential to make the task of ANN training much more user friendly than the traditional methods discussed and could be used as a helper tool to aid ANN development.

8. References

[1] Artificial Neural Networks Technology (1992). [Online]. Dave Anderson and George McNeil. http://www.dacs.dtic.mil/techs/neural/neural.title.html

[2] AI-FAQ (2002). [Online]. Warren S. Sarle. ftp://ftp.sas.com/pub/neural/FAQ3.html

[3] generation5 (2004). [Online]. Wan Hussain and Wan Ishak. http://www.generation5.org/articles.asp?Action=List&Topic=Neural%20Networks

[4] Evolutionary Neural Networks: Design Methodologies (2001-2003). [Online]. Rinku Dewri. http://ai-depot.com/Articles/47/EANN.html

[5] Using Genetic Algorithms with Neural Networks (2000). [Online]. James Matthews. http://www.generation5.org/content/2000/nn_ga.asp

[6] REVIEW OF HARDWARE NEURAL NETWORKS: A USER'S PERSPECTIVE (1998). [Online]. Clark S. Lindsey and Thomas Lindblad. http://www.particle.kth.se/~lindsey/elba2html/ba2html.html

[7] NeuroSolutions (2005). [Online]. NeuroDimension, Inc. http://www.neurosolutions.com/

[8] Statistica (2005). [Online]. StatSoft, Inc. http://www.statsoft.com/products/stat_nn.html

[9] The MathWorks: Accelerating the pace of engineering and science (2006). [Online]. The MathWorks, Inc. http://www.mathworks.com/products/neuralnet

[10] Java Object Oriented Neural Engine (2006). [Online]. Paula Marrone. http://www.jooneworld.com/docs/dte.html

[11] Software Agent (2006). In Wikipedia: The free encyclopedia. http://en.wikipedia.org/wiki/Software_agent

[12] Andrew S. Tanenbaum and Maarten van Steen (2002) Distributed Systems: Principles and Paradigm, USA: Prentice-Hall.

[13] Getting started using RMI (1999) [Online]. Sun Microsystems, Inc. http://java.sun.com/j2se/1.4.2/docs/guide/rmi/getstart.doc.html

[14] Kenneth Baclawski (1998) Java RMI Tutorial [Online]. http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

[15] Ann Wollrath and Jim Waldo (2005) Trail:RMI[Online]. http://java.sun.com/docs/books/tutorial/rmi/index.html

[16] Jean Vaucher and Ambroise Ncho (2003). JADE primer and tutorial. [Online]. http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html

[17] Zvi Boger and René Weber. Finding an Optimal Artificial Neural Network Topology in Real-Life Modeling - Two Approaches

[18] Bernhard ¨Omer (1995). Genetic Algorithms for Neural Network Training on Transputers

[19] *Michael Watts and Nikola Kasabov.* GENETIC ALGORITHMS FOR THE DESIGN OF FUZZY NEURAL NETWORKS

[20] Kenneth O. Stanley and Risto Miikkulainen. Efficient Evolution of Neural Network Topologies