Machine Learning using Neural Network And Evolutionary Algorithm

Mr. Tushar Ghude.
Dept. of Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India.

Prof. Avinash Shrivas.
Dept. of Computer Engineering
Vidyalankar Institute of Technology
Mumbai, India.

Abstract—**Multi-layer networks use a variety of learning techniques, the most popular being back-propagation. Though multilayered feedforward neural networks possess a number of properties which make them particularly suited to solve complex pattern classification problems it faces difficulties in solving some real world problem due to lack of a training algorithms which reliably finds a nearly globally optimal set of weights in a relatively short time. Also the choice of the basic parameter (network topology, learning rate, initial weights) often already determines the success of the training process. The selection of these parameters follows in practical use rule of thumb, but their value is at most arguable. Genetic algorithms (GAs) usually avoid local minima by searching in several regions simultaneously. This paper presents a modified Epnet algorithm to evolve network architecture and weights simultaneously. This paper also introduces a new method of encoding different neural networks and weights.**

Keywords— ***Evolution, evolutionary algorithms, evolution of network architecture, generalization, learning, neuralnetwork design, complexification, Neuroph, genetic algorithms, artificial neural network***

I. INTRODUCTION

An Artificial neural network (ANN) is interconnected network of artificial neurons. By the use of training, generally a gradient descent algorithm such as back-propagation is able to learn map input patterns to output patterns. A neural network trained for classification is designed to take input samples and classify them into groups. These groups may be fuzzy, without clearly defined boundaries. These groups may also have quite rigid boundaries. The ability of an ANN to learn is considered to be a property of its structure as well as the value of its weights, however the most appropriate ANN structure is still generally heuristically chosen for an application.

Backpropagation (BP) training algorithm [10] has been known to be very useful in solving a wide variety of real world problems (such as Pattern Classification, Clustering, Function Approximation, Forecasting, Optimization, Pattern Association and Control) but despite its popularity in the training of multilayer perceptron (MLP), BP has some drawbacks. neural networks can get stuck in local minima depending on the shape of the error surface, the values of the randomly initialized weights and some other parameters, that is, BP very much depends on good, problem specific parameter settings. There also might be other factors leading to this problem. For example descending very fast on a steep valley, if network is using first order gradient descent, it might get to the opposite slope and bounce back and forth all the time.

II. PRINCIPLE STRUCTURE OF A GENETIC ALGORITHM AND NEURAL NETWORK (GANN) SYSTEM

The idea of combining GA and ANN came up first in the late 80s, and it has generated an intense field of research in the 1980s. By combining genetic algorithms with neural networks (GANN), genetic algorithm is used to find network parameters. The inspiration for this idea comes from nature: In real life, the success of an individual is not only determined by his knowledge and skills, which he gained through, experience (the neural network training), it also depends on his genetic heritage (set by the genetic algorithm).

A genetic algorithm tries to simulate the natural evolution process. Its purpose is to optimize a set of parameters. In the original idea, proposed by John Holland [Holland, 1975], the genetic information is encoded in a bit string of fixed length, called the parameter string or individual. A possible value of a bit is called an allele. Each parameter string represents a possible solution to the examined problem. For the GANN problem, it contains information about the construction of a neural network. The quality of the solution is stored in the fitness value [8]. The basic GA operators are crossover [6], selection and mutation [9]. The selection of individuals for cross-over and mutation is biased towards good individuals. The chance of an individual to be selected is based on its relative fitness in the population. Crossover is generating offspring from two parent networks. This is performed by taking parts of the bit-string of one of the parents and the other parts from the other parent and combining both in the child. The probability of mutation is a set percentage of the number of active connections in the offspring that will undergo weight mutation.

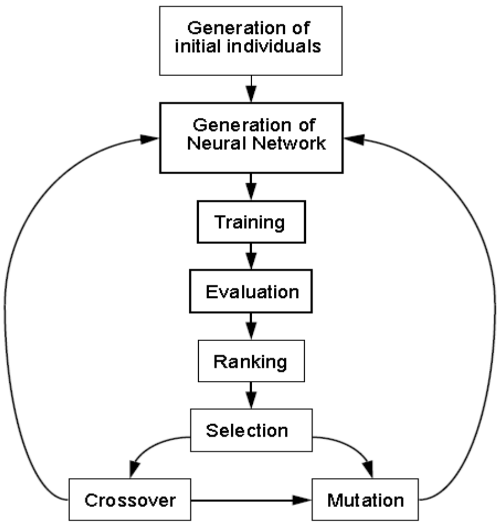
Generation of initial individuals
Generation of Neural Network
Training
Evaluation
Ranking
Selection
Crossover
Mutation

Fig. 1. *Structure of GANN*

III. DESCRIPTION OF THE PROPOSED ALGORITHM

The emphasis of the algorithm is on maintaining the behavioral link between the parent network and offspring and is achieved by the use of evolutionary programming, partial weight training after each architectural mutation and node splitting in order to give a chance for newly created network to stabilized (protecting new innovation).

Algorithm combines the architectural evolution of a neural network with its weight learning. This step-wise process involves the five mutation operators: hybrid training (using a back-propagation algorithm and simulated annealing), node deletion, node addition, altering learning rate and momentum, connection deletion. Connection deletion is performed at the end as it is observed during the study deleting and adding random connections during the evolution process hampers network learning when mutation factor is large (at the beginning of the process), instead adding or deleting nodes with connections in a specified way helps in maintaining behavioral link. In order to encourage evaluation of smaller network penalty can be added. Connection deletion can be done safely at the end of evaluation process when network is relatively steady by deleting connections with the smallest absolute values.

Basic steps in algorithm

Step 1: Initial Population Generation and Initialization

In this step, algorithm randomly generates and initializes a population of feed-forward artificial neural networks with randomized node density and weight values within a specified range. The initial population is generated based on the four user specified parameters Minimum Initial Node Density, Initial Node Connection Density, Minimum Hidden Nodes Kept and Minimum Connections Kept. In order to find out minimum node and connection density necessary, run NEAT algorithm [2] for the given problem for time t_0. (NEAT stands for Neuroevolution of augmenting topologies. It is a method for evolving artificial neural networks with a genetic algorithm. NEAT implements the idea that it is most effective to start evolution with small, simple networks consisting of only input layer neurons and output layer neurons and allow them to become increasingly complex over generations. Process of complexification [2] is effective for continual elaboration to find out highly sophisticated networks for continuously changing environment but convergence is rather slow and it takes very long time to build a near optimum solution [8]. In classification problems once network is generalized it does not undergoes major changes in its hierarchy). Idea behind using NEAT is only to find out approximate minimum network architecture and therefore time required is comparatively small. NEAT also helps in order to decide ranges for initial weight distribution. Node and connection density is achieved by generating fully interconnected networks with all the hidden nodes active. A random percentage between the Minimum Initial Node Density is set and then hidden nodes are removed at random to achieve the required node density level. The number of hidden nodes in the network cannot be less than the Minimum Hidden Nodes Kept.

Step 2: Partially train each network in the population on the training set for a certain number of epochs using Back Propagation with adaptive learning rates. The number of epochs is specified by the user. The error value E of each network on the validation set is checked after partial training. If E has not been significantly reduced, then the assumption is that the network is trapped in a local minimum.

Step 3: Rank the networks in the population according to their error values, from the best to the worst. If the best network found is acceptable or the maximum number of generations has been reached, stop the evolutionary process and go to Step 9.

Step 4: Use rank-based selection to choose one parent network from the population. Copy a parent net to generate sub-population. The sub-population inherits all the parameter settings of the main population. Only diversity that is admitted to the sub-population is Standard Deviation for connection and bias weight mutation.

Step 5: obtaining offspring network

5a. Select two parent networks from the sub-population. Clone the second parent to produce an offspring.

5b. Crossover: The probability of crossover is a set percentage of the number of active connections in the offspring that will undergo weight crossover. These numbers of connections are selected at random from the first parent and the associated weights are copied to the offspring. The parent and the offspring have identical structures as they are derived from one main network, hence a connection that exists in the parent will also exist in the offspring. If the offspring is better than the parent it was cloned from, it replaces that parent in the sub-population, otherwise if the offspring is better than the first parent it replaces that parent in the sub-population. If the offspring is inferior, then it is discarded and no replacement will occur.

Step 6: Mutation: Generate a random number between 50 and 100. This will be the percentage of connection weights that will be mutated. Flip a computational coin where the result may be 0 or 1 with 50 percent probability of either result. The coin toss is used to decide whether to increase or decrease the standard deviation of the distribution used for sampling values for weight mutation. When the standard deviation is increased (coin toss results in a 1), bigger numbers are added to connection and bias weights. When the standard deviation is decreased (coin toss results in a 0), smaller numbers are used. When network comes close to convergence, the coin toss is additionally biased (90 percent) in favor of decreasing the standard deviation for weight mutation. If the new solution has a higher fitness than the parent, then terminate simulated annealing. The new solution will replace the parent in the main population.

Step 7: Node deletion: First decide the number of hidden nodes N_{hidden} to be deleted by generating a uniformly distributed random number between one and a user-specified maximum number. N_{hidden} is normally very small in the experiments, no more than three in most cases. Then delete N_{hidden} hidden nodes from the parent network uniformly at random. Partially train the pruned network by the Back Propagation to obtain an offspring network. If the offspring is better than the worst network in the current population, replace the worst by the offspring and go to Step 5a). Replacement in the main population concludes a generation. Otherwise discard the offspring continue.

Step 8: Node splitting: The maximum number of new nodes that will be added in this step is a random number between a user set minimum and maximum. A new hidden node is added by splitting an existing hidden node.

The weights of incoming connections to the new node and to the split node will be the same.

The weights of the outgoing connections from the new node have the value: (-1 * a) * OriginalWeight.

The weights of the outgoing connection from the split node have the value: (+1 * a) * OriginalWeight.

Where, ‘a’ is a random number taken from a Gaussian distribution with a mean of 0 and a set standard deviation.

Step 9: Repeat above procedure for every parent in the main population. After the evolutionary process, train the best network further on the combined training and validation set using Back Propagation until it converges. (there is no further significant improvement in the accuracy after a certain number of epochs/cycles).

IV. NETWORK ENCODING

Evolutionary algorithms [8] operate on a population of genotypes (also referred to as genomes). In neuroevolution [2], a genotype is mapped to a neural network phenotype that is evaluated on some task to derive its fitness. In direct encoding schemes the genotype directly maps to the phenotype. That is, every neuron and connection in the neural network is specified directly and explicitly in the genotype. In contrast, in indirect encoding schemes the genotype specifies indirectly how that network should be generated. Indirect encodings are often used to achieve several aims:

- Allow recurring structures or features in the network (modularity and other regularities);
- Compression of phenotype to a smaller genotype

Indirect encodings have been shown to produce highly regular solutions to problems, but their bias toward regularity makes it difficult for them to properly handle irregularities in problems.

Implemented algorithm uses a different type of encoding technique to represent the genotype of the ANN. Neural Network is implemented using 'Neuroph'. Neuroph is an object-oriented neural network framework written in Java. It can be used to create and train neural networks in Java programs. Neuroph provides Java class library with basic support for creating and training Neural Networks (current version of Neuroph does not include inbuilt support for genetic algorithms.)

To encode a neural network we use two types of data structures to represent the genotype of the ANN. A two-dimensional String array used to store source neuron and destination neuron for a connection and one-dimensional array of real numbers used to store weight associated with that connection. To encode a Network label every neuron with a unique ID - layer index followed by index of the neuron in that layer. Every row in the two-dimensional array represents an input connection for destination neuron; connection weight is stored separately in second array. Bias will be encoded at the end of the layer. For example

TABLE I. ENCODING STUCTURE

Source	Destination
l0-n0	l1-n0
l0-n1	l1-n0
l0-n2	l1-n0
l0-n3	l1-n0
l0-n4	l1-n0
l0-n5	l1-n0
l0-n6	l1-n0

0.906345956
0.695224389
0.365454554
0.654654556
1.848646846
0.654686545
0.846545454

Table above represents a neural network with seven neurons in first layer (i.e. layer 0) connected to a neuron in second layer (i.e. layer 1). Connection l0-n6 (layer-index 0, neuron-index 6) to l1-n0 (layer-index 1, neuron-index 0) is a bias. This type of encoding is very flexible can support both types of crossover - bit level in which algorithm considers every connection separately and node level crossover. [11]

V. CONCLUSION

This paper describes an algorithm for evolving neural network using genetic algorithm and possible methods for solving related problems like network encoding and maintaining behavioral link while altering network topology. Given network encoding scheme is useful for implementing both bit level and node level crossover. Also presents a way to improve the time factor and avoid frequent alteration of network in Epnet using two methodologies at the same time. Its expected error factor is less than pure gradient descend using heuristic based fix network topology. Calculation of the error fitness function needs further research. In particular calculating network complexity penalty for large networks since initially algorithm tends to favor larger networks in training process.

REFERENCES

[1] A New Evolutionary System for Evolving Artificial Neural Networks Xin Yao, Senior Member, IEEE, and Yong Liu 3, MAY 1997 IEEE TRANSACTIONS ON NEURAL NETWORKS

[2] Efficient Reinforcement Learning through Evolving Neural Network Topologies Kenneth O. Stanley Department of Computer Sciences University of Texas at Austin Austin, TX 78712.

[3] Artificial Neural Networks and Genetic Algorithms Instructor: Shu-Heng Chen Department of Economics. National Chengchi University

[4] Fusion of neural networks with fuzzy logic and genetic algorithm Sung-Bae Cho Department of Computer Science, Yonsei University, 134 Shinchon-dong, Sudaemoon-ku, Seoul 120-749, South Korea.

[5] Genetic Algorithm And Neural Network for OPTICAL CHARACTER RECOGNIZATIO Hendy Yeremia, Niko Adrianus Yuwono, Pius Raymond and Widodo Budiharto.

[6] A Study of Crossover Operators in Genetic Programming William M. Spears.

[7] Training Feedforward Neural Networks Using Genetic Algorithms David J. Montana and Lawrence Davis BBN Systems and Technologies

[8] D. B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. New York: IEEE Press, 1995.

[9] Y. Liu and X. Yao, “Evolutionary design of artificial neural networks with different nodes,” in Proc. 1996 IEEE Int. Conf. Evolutionary.

[10] Comparing the Performance of Backpropagation Algorithm and Genetic Algorithms in Pattern Recognition Problems Chukwuchekwa Ulumma Joy. Department of Mathematics Federal University of Technology,Owerri.

[11] Evolving crossover, mutation and training rate in a population of Neural Networks. Dara Curran. Dept of information technology, University Of Ireland.