Evolving Artificial Neural Networks for Prediction in Robocup Soccer

M. Moghimi Najaf Abadi, C. Lucas
ECE Department
University of Tehran
m.moghimi@ece.ut.ac.ir

Abstract: *The prediction of the future states in Multi Agent Systems such as Robocup Soccer has been a challenging problem since the beginning of the MAS. Robocup 3D Soccer is selected because of its global view of the agents. An Artificial Neural Network is used for prediction. The goal is to concentrate on the design of an optimal ANN. In order to handle it, a genetic algorithm is used for optimization of the design, which shows a great improvement over the manual methods.*

Keywords: Genetic Algorithm, Evolutionary Strategy, Artificial Neural Network, Opponent Modelling, Prediction, Robocup Soccer.

1 Introduction

Considering the growth of Robocup Teams in recent years, constructing a hardwired team which has a constant behavior against the opponent is not a good choice anymore. The agent must have more flexible behavior while playing against a new team. Therefore it can be concluded that the understanding of how the opponent plays is an important fact. This makes Opponent Modelling becomes an active branch in soccer simulation communities' efforts.

An Artificial Neural Network may learn pattern between its inputs and outputs, thus can be used to model a soccer team. A new problem arises here: Which network architecture has the best quality? Because of the limited time in a match, this problem can have a great effect in efficiency and accuracy of the predicted values. Neural networks domain is complex, so evolving neural networks can be the right method to find the optimal network.

Evolutionary Algorithms are used to perform various tasks, such as connection weight training, architecture design and learning rule adaptation [2]. Excessive works in architecture design with evolutionary algorithms prove its power in architecture design of an ANN.

The purpose of the Robocup Simulated Soccer League is to provide a standardized problem domain for Artificial Intelligence research based on a soccer simulation called the Robocup Soccer Server [5].

Two teams of eleven agents which have been programmed by human must play a soccer game and they compete to win the match. The agents are autonomous to perform well-defined actions known as *drive* and *kick*. The former moves the agent and the latter kicks the ball if the agent is near the ball. Agents cannot jump, so the goal is not high. Agents get a message from the server each cycle and they have limited time to decide which action is the best, and then they send their decided actions to the server. In soccer 3D a message contains information on the agent, its team mates, the opponent positions and the location of the ball.

Agent's global view of the world makes the understanding the world state become easier. Using this information a neural network can be trained.

The prediction system will be described in the next section. In section 3, evolution of neural network will be explained and finally the results of the system will be discussed in section 4.

2 The Prediction System

Artificial Neural Networks are popular machine learning methods which are widely used in Robocup [6]. For prediction purposes, a neural network is used, which gets the state of the match and the outputs opponents' movements. A multi layer feed-forward neural net is used for this task and Back Propagation Algorithm is used for training the network. Section 3 describes the network architecture. Yet a problem remains, which is the input and output of the network.

The prediction is more useful for defender agents than the attackers because defenders' play is more passive and depends on what opponent attackers do. Thus the movement of opponent attackers is a good choice for the output. It's better to predict the opponent attackers' positions in the next k cycles in the future. The integer k should not be so large, because it will decrease the accuracy of the prediction system. There are two parameters x and y for each player, so the output size is twice the number of the attackers which the system wants to predict.

The input is more evident because the world state is well defined. It contains forty seven real valued parameters, two for each player and three for the ball as it has z parameter in addition to x and y. Other values may be added to the input set, such as the preceded state of the world or the objects velocities, but adding these values results in long training time.

Prediction system output is used by defenders, so some input and output pairs should be ignored, because opponent attack states are required, in which the last of the ball owners is the opponent and the ball is as near to the goal as a threshold and it is moving towards the goal.

It should be noted that the agent cannot find out the real ball owner, but it can assume that the player controlling the ball executes a kick if the ball velocity increases [3].

3 Evolving Neural Networks

One interesting result in the evolving of the neural networks is the digit recognition problem which consists of identifying fixed representation of digits 1-9 correctly. The result shows that no hidden neurons is needed that makes experimenters surprised as they were not aware of the fact that the problem was linearly separable [4].

Xin Yao divided the evolving neural networks researches into two categories, direct encoding scheme and indirect scheme [2]. Direct encoding scheme is powerful, but its search space, neural nets space, in this problem is very large because of the numerous input neurons and probably the hidden neurons. Indirect methods such as parameterized, developmental rule or fractal representation work with small search space. Thus, faster convergence with indirect methods is expected.

In this research, combination of a special parameterized representation and nodes transfer function evolution is presented. GA's are capable of many non traditional modifications and can be changed in the following form to evolve neural networks.

Table 1: Pseudo Code for Genetic Algorithm

1	Initialize an original population of the individuals (chromo+somes) randomly.
2	Decode each individual to an ANN.
3	Evaluate the fitness function, or similarly the error function, for each individual by letting the ANN train and computing its error function and its training time.
4	Select parents for crossover.
5	Apply crossover and mutation operators, which form the next population.
6	Goto 2.

The following two subsections explain the solutions for these representation and fitness problems.

3.1 Representation Scheme

At the machine level, an individual in the population is a string of bits or numbers. To the GA, an individual is identical to its chromosome [1]. How to encode an individual to its string is the representation problem.

There is a multi layer neural network which we want to encode to a string of numbers. The representation depends on what information we want to store. The most important parameters in the design of a network are number of its hidden layers, the size of the layers and neurons activation function. The last fact is not directly related to the architecture. These are major parameters an ANN needs to be set.

One simple solution to this problem is to store one number for each hidden layer which represents its

size. But how can GA test different layers number? One solution is to run the GA for different layers number to find the best solution similar to Iterative Deepening Search Algorithm. Another solution that is implemented in this research is to let the GA change the number of the layers. Again, the simple approach is to encode the number of the layers as a new parameter and let the chromosome contain the information of the layers which is less than an integer. It represents the maximum number of layers. A better solution is to use variable length chromosomes which means the higher the layers number, the lengthier the chromosome.

Because activation functions space is very huge, which equals to one dimensional functions space, the selective approach is used. The GA must select among Sigmoid, Gaussian, Linear and Tan-Sigmoid which are the most applied functions used for neurons activation function.

To put it more precisely, for each layer one number for activation function and one for the layer size is stored, which construct the layer information, and the layers themselves construct the whole chromosome. To handle its variable length, as figure 1 shows a modified version of the crossover operator is used which selects one point in each parent, breaks the two parents' chromosomes and then connect one piece of each parent to one piece of another. In this way, chromosomes can have different lengths.

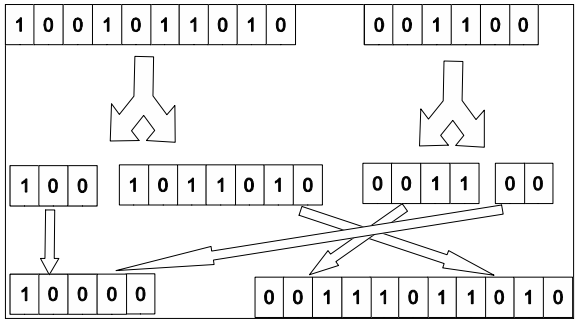

Figure 1: Crossover schema

3.2 Computing Fitness

Using the fitness function, GA tries to find the best solution which maximizes the fitness function. This function is used for giving a rank to each individual in the population in order to give it more chance to live or to have a child. The error function can similarly be used for this purpose, and in some problems it is easier to compute the error rather than the fitness, especially in evolving neural networks [1]. Given a non-trained neural network which is constructed from an individual chromosome, the system must be capable of computing its fitness. First of all the network should be trained train network with a special dataset. Since long time is needed for convergence of an ANN, it is impossible to let the network train until it has an error value less than a threshold. The fitness is only needed for comparison, so if we have set the same conditions for networks, we can compare them together. For instance, letting the networks be trained within some constant epochs can be done in a short period of time.

Two main factors are available. They are output quality and training time or the network complexity. The best value that represents the network complexity is its training time for our purpose. We can compare the network output with the desired outputs for a dataset to compute the output error. We can add these factors with different coefficients. Consider the output of network rmse to be *oe* and the training time to be *tt*. The error function is computed by the following formula:

$$error = \alpha \times oe + \beta \times tt \qquad (1)$$

Since function is only used for comparison as mentioned, only α / β is important. Considering the importance of time and the network error, we have different error functions, and results show that each has its own optimal network. The α / β factor is very effective on the result. Decreasing the value of α / β results in small networks with less complexity and vice versa. The next section shows the system output for different α's and β's.

4 Results

The following tables show the result of GA with specified parameters. The training time has important effect on the optimal neural network. These results reveal the fact that ``the lower the α / β, the bigger the network". The best value for α / β for prediction task in a real play is 1e-5. The mean error of the optimal network founded by the GA is about 5 meters for *k* equals to 5, if it was trained during a reasonable time in a match.

This means that it can predict the opponent attacker's positions in 5 next cycles with 5 meters accuracy. The best manual designed network is less accurate. Its error is at least 8 meters in predicting their positions.

5 Conclusion and Future Works

These results document the quality of the solution that GA's afford when applied to prediction task.

The saving in time and labor are among their most important advantages. Equally important is the higher level confidence in the solution. The genetic algorithm can cover considerably larger search spaces than be covered manually [1].
This system can be used when the dataset is available in offline mode. The long time needed for running the genetic algorithm makes it non practical for online uses.
Future work is to design a system to be used in online mode which needs faster algorithms.

References

[1] R. Hochman and T. Khoshgoftaar and E. Allen and J. Hudepohl, ``Using the genetic algorithm to build optimal neural networks for fault-prone module detection", Proceeding of the Seventh Int. Symposium on Software Reliability Engineering, pp 152-162. Oct. 30 - Nov 2, 1996, White Plians, N.Y.

[2] X. Yao, ``Evolving Artificial Neural Networks", Proceeding of the IEEE, VOL. 87, NO. 9, Sep 1999.

[3] T.Steffens, ``Adapting Similarity Measures to Agent Types in Opponent Modelling", Workshop on Modeling Other Agents from Observasions at AAMAS 2004, (2004), pp 125-128.

[4] A. J. Jones, ``Genetic algorithms and their applications to the design of neural networks.", Journac of Neural Computing & Applications, Feb 1993, pp 32-45.

[5] M. Chen, K. Dorer, E. Foroughi, et al, ``Users Manual, Robocup Soccer Server, for Soccer Server Version 7.07 and later" February 11, 2003.

[6] M. H. Dezfoulian, N. Kaviani, A. Nikanjam, et al, ``Training a Simulated Soccer Agent how to Shoot using Artificial Neural Networks", Proceeding of the Iranian Researchers Conference in Europe, 2005.

Table 2: The result for $\alpha = 1$ and $\beta = 1e-10$

Generation	Fitness	Average Fitness	Best Network Architecture
10	7e+11	7e+12	2 tansig 24 sigmoid
20	6e+11	6e+12	3 tansig 20 identity
30	4e+11	7e+12	5 tansig 20 identity
40	4e+11	5e+12	5 tansig 20 identity
50	4e+11	5e+12	5 tansig 20 identity

Table 3: The result for $\alpha = 1$ and $\beta = 0$

Generation	Fitness	Average Fitness	Best Network Architecture
10	1.68e-02	3.62e-02	5 tansig 24 sigmoid
20	1.66e-02	3.80e-02	5 tansig
30	1.60e-02	3.55e-02	5 tansig 24 gaussian 2 identity
40	1.60e-02	4.51e-02	5 tansig 24 gaussian 2 identity
50	1.40e-02	1.14e-01	5 tansig 24 gaussian 2 identity

Table 4: The result for $\alpha = 1$ and $\beta = 1e-5$

Generation	Fitness	Average Fitness	Best Network Architecture
10	70040.6	100040.6	2 tansig 24 gaussian
20	66040.6	70020.6	2 tansig
30	30040.6	50043.4	1 tansig
40	30040.6	61020.2	1 tansig
50	30040.6	30130.2	1 tansig