Playing Tic-Tac-Toe Using Genetic Neural Network with Double Transfer functions

[1]S.H. Ling, and [2]H.K. Lam
[1]Faculty of Engineering and Information Engineering, University of Technology Sydney, NSW, Australia
[2]Division of Engineering, The King's College London, Strand, London, WC2R 2LS, United Kingdom

Abstract: Computational intelligence is a good tool for game development. In this paper, an algorithm of playing game tic-tac-toe with computational intelligence is developed. This algorithm is learned by a Neural Network with Double Transfer functions (NNDTF), which is trained by genetic algorithm (GA). In the NNDTF, the neuron has two transfer functions and exhibits a node-to-node relationship in the hidden layer that enhances the learning ability of the network. A Tic-Tac-Toe game is used to show the NNDTF provide a better performance than traditional neural network.

I. Introduction

Games such as Backgammon, Chess, Checkers, Go, Othello and Tic-tac-toe are widely used platforms for studying the learning ability of and developing learning algorithms for machines. By playing games, the machine intelligence can be revealed. Some techniques of artificial intelligence, such as the brute-force methods and knowledge-based methods [1], have been reported. Brute-force methods, e.g. retrograde analysis [2] and enhanced transposition-table methods, solve the game problems by constructing databases for the games. For instance, the database is formed by a terminal position [2]. The best move is then determined by working backward on the constructed database. For knowledge-based methods, the best move is determined by searching a game tree. For games such as Checkers, the tree spanning is very large. Tree searching will be time consuming even for a few plies. Hence, an efficient searching algorithm is an important issue. Some searching algorithms, which are classified as knowledge-based methods, are threat-space search and λ-search, proof-number search [3], depth-first proof-number search and pattern search.

It can be seen that the above game-solving methods depend mainly on the database construction and searching. The problems are solved by forming a possible set of solutions based on the endgame condition, or searching for the set of solutions based on the current game condition. The machine cannot learn to play the games by itself. Unlike an evolutionary approach in [1], neural network (NN) was employed to evolve and to learn for playing Tic-tac-toe without the need of a database. Evolutionary programming was used to design the NN and link weights. A similar idea has been applied in a Checkers game [6-9]. Other games such as Backgammon [4], Othello [10] and Checkers [11] applying NNs or computational intelligence techniques can also be found.

In this paper, a neural network with double transfer functions (NNDTF) is proposed to learn the rules of Tic-tac-toe. Each possible move is evaluated by a proposed algorithm with a score. By maximizing the total scores (evaluated values), the rules of Tic-tac-toe can be extracted by the NNDTF. Different from the traditional feed-forward multiple-perception NN, proposed transfer functions and a node-to-node relationship are introduced to the proposed NN. The modified transfer functions are allowed to change the shapes during operation. Hence, the working domain is larger than that of the traditional one. By introducing the node-to-node relationship between hidden nodes, information can be exchanged between hidden layers. As a result, the learning ability is enhanced. A genetic algorithm (GA) [12] is investigated to train the NNDTF. The trained NNDTF will then be employed to play Tic-tac-toe with a human player as an example.

This paper is organized as follows. An algorithm to evaluate each move on playing the Tic-tac-toe will be proposed in section II. NNDTF will be presented in section III. Genetic Algorithm will be presented in section IV. Training of the NNDTF using GA to learn the rules of Tic-tac-toe will be presented in section V. An example on playing Tic-tac-toe with a human player will be given in section VI. A conclusion will be drawn in section VII.

II. Algorithm for Playing Tic-tac-toe

The game Tic-tac-toe, also known as naughts and crosses, is a two-player game. Each player will place a marker, “X” for the first player and “O” for the second player, in turn in a three-by-three grid area. The first player takes the first move. The goal is to place three markers in a line of any direction on the grid area.

An algorithm is proposed in this section to evaluate the move on each grid. An “X” and an “O” on a grid are denoted by 1 and –1 respectively. An empty grid is denoted by 0.5. The following procedure is used to evaluate each possible move.

1) Place an “X” on an empty grid.
2) Corresponding to step 1, sum up all the grid values for each line in any direction, e.g., for a grid in the corner, we have three evaluated values because there are three lines to win or lose.
3) Remove the “X” placed in step 1 and place an “X” on another empty grid. Evaluate this grid using the algorithm in step 2. Repeat this process till all empty grids are evaluated.
4) After evaluation, each grid will have been assigned at least 2 evaluated values for all possible lines, e.g. the center grid will have 4 evaluated values, corner grids will have 3 evaluated values and other grids will have 2 evaluated values. There are totally 6 possible evaluated values: 3 (1+1+1), 2.5 (1+1+0.5), 2 (1+0.5+0.5), 1 (–1+1+1), 0.5 (–1+1+0.5) and –1 (–1–1+1). The most important evaluated value of a grid is 3, which indicts a winning of the game (3 “X”s in a line) if you put an “X” on that grid. The priority of taking that move is the highest. The second important evaluated value of a grid is –1 (2 “O”s and 1 “X” in a line), which indicts that the opponent should be prevented from winning the game. The priority of taking that move is the second highest. Using this rationale, the list of priority in a descending order is: 3, –1, 2.5, 2, 1, 0.5. Based on these assigned evaluated values, a score will be assigned to each possible move. First, each evaluated value is assigned a score: $3 \rightarrow \gamma_6 = 7^7$, $-1 \rightarrow \gamma_5 = 6^6$, $2.5 \rightarrow \gamma_4 = 5^5$, $2 \rightarrow \gamma_3 = 4^4$, $1 \rightarrow \gamma_2 = 3^3$ and $0.5 \rightarrow \gamma_1 = 2^2$. The chosen scores have the following properties,

$$\gamma_6 > 4\gamma_5 \quad (1)$$

$$\gamma_5 > 4\gamma_4 \quad (2)$$

$$\gamma_4 > 4\gamma_3 \quad (3)$$

$$\gamma_3 > 4\gamma_2 \quad (4)$$

$$\gamma_2 > 4\gamma_1 \quad (5)$$

The sum of the scores of a grid is the final score. The final scores will be used to determine the priorities of the possible move. A higher final score of a grid indicates a higher priority of that move. The reasons for choosing the scores in this way with the properties of (1) to (5) are as follows. As the evaluated value of 3 indicates a winning of the game (3 "X"s in a line), the score of γ_6 must be the highest. There are at most four evaluated values for a grid. Hence, γ_6 must be greater than 4 times the second largest evaluated values, i.e. $4\gamma_5$. Consequently, the priority of a grid having an evaluated score with a higher priority will not be affected by other lower evaluated scores. For instance, consider a grid having evaluated values of 3 and 0.5, and another grid having evaluated values of -1, 2.5, 2 and 0.5. The final score of the former grid ($7^7 + 2^2$) is bigger than and latter grid ($7^7 + 2^2$ and $6^6 + 5^5 + 4^4 + 2^2$). Thus, the "X" should be place at the grid having an evaluated value of 3 to win the game.

Take the game as shown in Fig. 1 as an example, we have 3 "X"s and 3 "O"s. The next move will be to place an "X". After assigning an empty grid to be 0.5, an "X" to be 1, an "O" to be –1, Fig. 1(b) is obtained. Following Step 1 to Step 3, we obtain the evaluated values as shown in Fig. 1(c). Based on Step 4, Fig. 1(d) shows the final scores for the empty grids. As the highest score is 873324, the most appropriate move is to put an "X" on the bottom right corner. This move not only lines up 3 "X"s to win a game, but also prevents the opponent to line up 3 "O"s. The second appropriate move, indicted by the final score of 52906, can gain a chance to win by lining up 2 "X"s, and prevent the opponent to win.

III. Neural Network with Double Transfer functions (NNDTF)

NN was proved to be a universal approximator [13]. A 3-layer feed-forward NN can approximate any nonlinear continuous function to an arbitrary accuracy. NNs are widely applied in areas such as prediction, system modeling and control [13]. Owing to its particular structure, a NN is good in learning [2] using some learning algorithms such as GA [1] and back propagation [2]. In general, the processing of a traditional feed-forward NN is done in a layer-by-layer manner. In this paper, by introducing a node-to-node relationship in the hidden layer of the NN, a better performance can be obtained.

Fig. 2 shows the proposed neuron. It has two activation transfer functions to govern the input-output relationships of the neuron: static transfer function (STF) and dynamic transfer function (DTF). For the STF, the parameters are fixed and its output depends on the inputs of the neuron. For the DTF, the parameters of the activation transfer function depend on the outputs of other neurons and its STF. With this proposed neuron, the connection of the proposed NN is shown in Fig. 3, which is a three-layer NN. A node-to-node relationship is introduced in the hidden layer. Comparing with the traditional feed-forward NN [13], it was reported in [14] that the proposed NN can offer a better performance and need fewer hidden nodes. The details of the NNDTF are presented as follows.

A. The neuron model

We consider the STF first. Let v_{ij} be the synaptic connection weight from the i-th input component x_i to the j-th neuron. The output κ_j of the j-th neuron's STF is defined as,

$$\kappa_j = net_s^j\left(\sum_{i=1}^{n_{in}} x_i v_{ij}\right),\ i = 1, 2, \ldots, n_{in},\ j = 1, 2, \ldots, n_h \tag{6}$$

where n_{in} denotes the number of input and $net_s^j(\cdot)$ is a static activation transfer function. The activation transfer function is defined as,

$$net_s^j\left(\sum_{i=1}^{n_{in}} x_i v_{ij}\right) = \begin{cases} e^{\frac{-\left(\sum_{i=1}^{n_{in}} x_i v_{ij} - m_s^j\right)^2}{2\sigma_s^{j^2}}} - 1 & \text{if } \sum_{i=1}^{n_{in}} x_i v_{ij} \le m_s \\ 1 - e^{\frac{-\left(\sum_{i=1}^{n_{in}} x_i v_{ij} - m_s^j\right)^2}{2\sigma_s^{j^2}}} & \text{otherwise} \end{cases},$$

$$j = 1, 2, \ldots, n_h \tag{7}$$

where m_s^j and σ_s^j are the static mean and static standard deviation for the j-th STF respectively. The parameters (m_s^j and σ_s^j) are fixed after the training processing. Thus, the activation transfer function is static. The output of the STF depends on the inputs of the neuron only. From (7), the output value is ranged from –1 to 1. The shape of the proposed activation transfer function is shown in Fig. 4 and Fig. 5. It can be observed from these 2 figures that $net(f) \to 1$ as $f \to \infty$ and $net(f) \to -1$ as $f \to -\infty$.

Considering the DTF, the neuron output z_j of the j-th neuron is defined as,

$$z_j = net_d^j(\kappa_j, m_d^j, \sigma_d^j),\ j = 1, 2, \ldots, n_h \tag{8}$$

where $net_d^j(\cdot)$ is the DTF defined as follows,

$$net_d^j(\kappa_j, m_d^j, \sigma_d^j) = \begin{cases} e^{\frac{-\left(\kappa_j - m_d^j\right)^2}{2\sigma_d^{j^2}}} - 1 & \text{if } \kappa_j \le m_d^j \\ 1 - e^{\frac{-\left(\kappa_j - m_d^j\right)^2}{2\sigma_d^{j^2}}} & \text{otherwise} \end{cases},\ j = 1, 2, \ldots, n_h \tag{9}$$

where

$$m_d^j = p_{j+1,j} \times z_{j+1} \tag{10}$$

$$\sigma_d^j = p_{j-1,j} \times z_{j-1} \tag{11}$$

m_d^j and σ_d^j are the dynamic mean and dynamic standard deviation for the j-th DTF. z_{j-1} and z_{j+1} represent the output of the $j-1$-th and j+1-th neurons respectively. $p_{j+1,j}$ denotes the weight of the link between the j+1-th node and the j-th node and $p_{j-1,j}$ denotes the weight of the link between the $j-1$-th node and the j-th node. It should be noted that if $j = 1$, $p_{j-1,j}$ is equal to $p_{n_h,j}$ and if $j = n_h$, $p_{j+1,j}$ is equal to $p_{1,j}$. In this DTF, unlike the STF, the activation transfer function is dynamic as the parameters of its activation transfer function depend on the outputs of the $j-1$-th and j+1-th neurons. Referring to (1-4), the input-output relationship of the proposed neuron is as follows,

$$z_j = net_d^j(net_s^j(\sum_{i=1}^{n_{in}} x_i v_{ij}), m_d^j, \sigma_d^j),\ j = 1, 2, \ldots, n_h \tag{12}$$

B. Connection of the NNDTF

As shown in Fig. 3, the NNDTF has three layers with n_{in} nodes in the input layer, n_h nodes in the hidden layer, and n_{out} nodes in the output layer. In the hidden layer, the neuron model presented in the previous section is employed. The output value of the hidden node depends on the neighboring nodes and input nodes. In the output layer, a static activation transfer function is employed. Considering an input-output pair $(\mathbf{x}, \mathbf{y})$, the output of the *j*-th node of the hidden layer is given by

$$z_j = net_d^j(net_s^j(\sum_{i=1}^{n_{in}} x_i v_{ij})),\ j = 1, 2, \ldots, n_h \tag{13}$$

The output of the NNDTF is defined as,

$$y_l = net_o^l(\sum_{l=1}^{n_{out}} z_j w_{jl}),\ l = 1, 2, \ldots, n_{out} \tag{14}$$

$$= net_o^l(\sum_{l=1}^{n_{out}} net_d^j(net_s^j(\sum_{i=1}^{n_{in}} x_i v_{ij}))w_{jl}) \tag{15}$$

where w_{jl} denotes the weight of the link between the *j*-th hidden and the *l*-th output nodes; $net_o^l(\cdot)$ denotes the activation transfer function of the output neuron. The transfer function of the output node is defined as follows,

$$net_o^l(z_j) = \begin{cases} e^{\frac{-(z_k - m_o^l)^2}{2\sigma_o^{l\,2}}} - 1 & \text{if } z_k \le m_o^l \\ 1 - e^{\frac{-(z_k - m_o^l)^2}{2\sigma_o^{l\,2}}} & \text{otherwise} \end{cases} \tag{17}$$

where m_o^l and σ_o^l are the mean and the standard deviation of the output node activation transfer function respectively. The parameters of the NNDTF can be trained by GA [12].

IV. Genetic Algorithm

Genetic algorithms (GAs) are powerful searching algorithms. The traditional GA process [15-17] is shown in Fig. 6. First, a population of chromosomes is created. Second, the chromosomes are evaluated by a defined fitness function. Third, some of the chromosomes are selected for performing genetic operations. Forth, genetic operations of crossover and mutation are performed. The produced offspring replace their parents in the initial population. This GA process repeats until a user-defined criterion is reached. In this paper, the traditional GA is modified and new genetic operators [12] are introduced to improve its performance. The modified GA process is shown in Fig. 7. Its details will be given as follows.

A. Initial Population

The initial population is a potential solution set *P*. The first set of population is usually generated randomly.

$$P = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{pop\_size}\} \tag{18}$$

$$\mathbf{p}_i = \begin{bmatrix} p_{i_1} & p_{i_2} & \cdots & p_{i_j} & \cdots & p_{i_{no\_vars}} \end{bmatrix},\ i = 1, 2, \ldots, pop\_size;\ j = 1, 2, \ldots, no\_vars \quad (19)$$

$$para^j_{\min} \le p_{i_j} \le para^j_{\max} \quad (20)$$

where *pop_size* denotes the population size; *no_vars* denotes the number of variables to be tuned; p_{i_j}, $i = 1, 2, \ldots, pop\_size$; $j = 1, 2, \ldots, no\_vars$, are the parameters to be tuned; $para^j_{\min}$ and $para^j_{\max}$ are the minimum and maximum values of the parameter p_{i_j} for all *i*. It can be seen from (18) to (20) that the potential solution set *P* contains some candidate solutions $\mathbf{p}_i$ (chromosomes). The chromosome $\mathbf{p}_i$ contains some variables p_{i_j} (genes).

B. Evaluation

Each chromosome in the population will be evaluated by a defined fitness function. The better chromosomes will return higher values in this process. The fitness function to evaluate a chromosome in the population can be written as,

$$fitness = f(\mathbf{p}_i) \quad (21)$$

The form of the fitness function depends on the application.

C. Selection

Two chromosomes in the population will be selected to undergo genetic operations for reproduction by the method of spinning the roulette wheel [1]. It is believed that high potential parents will produce better offspring (survival of the best ones). The chromosome having a higher fitness value should therefore have a higher chance to be selected. The selection can be done by assigning a probability q_i to the chromosome $\mathbf{p}_i$:

$$q_i = \frac{f(\mathbf{p}_i)}{\sum_{j=1}^{pop\_size} f(\mathbf{p}_j)},\ i = 1, 2, \ldots, pop\_size \quad (22)$$

The cumulative probability $\hat{q}_i$ for the chromosome $\mathbf{p}_i$ is defined as,

$$\hat{q}_i = \sum_{j=1}^{i} q_j,\ i = 1, 2, \ldots, pop\_size \quad (23)$$

The selection process starts by randomly generating a nonzero floating-point number, $d \in \begin{bmatrix} 0 & 1 \end{bmatrix}$. Then, the chromosome $\mathbf{p}_i$ is chosen if $\hat{q}_{i-1} < d \le \hat{q}_i$, $i = 1, 2, \ldots, pop\_size$, and $\hat{q}_0 = 0$. It can be observed from this selection process that a chromosome having a larger $f(\mathbf{p}_i)$ will have a higher chance to be selected. Consequently, the best chromosomes will get more offspring, the average will stay and the worst will die off. In the selection process, only two chromosomes will be selected to undergo the genetic operations.

D. Genetic Operations

The genetic operations are to generate some new chromosomes (offspring) from their parents after the selection process. They include the crossover and the mutation operations.

1. Crossover

The crossover operation is mainly for exchanging information from the two parents, chromosomes $\mathbf{p}_1$ and $\mathbf{p}_2$, obtained in the selection process. The two parents will produce one offspring. First, four chromosomes will be generated according to the following mechanisms,

$$\mathbf{os}_c^1 = \begin{bmatrix} os_1^1 & os_2^1 & \cdots & os_{no\_vars}^1 \end{bmatrix} = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2} \tag{24}$$

$$\mathbf{os}_c^2 = \begin{bmatrix} os_1^2 & os_2^2 & \cdots & os_{no\_vars}^2 \end{bmatrix} = \mathbf{p}_{\max}(1-w) + \max(\mathbf{p}_1, \mathbf{p}_2)w \tag{25}$$

$$\mathbf{os}_c^3 = \begin{bmatrix} os_1^3 & os_2^3 & \cdots & os_{no\_vars}^3 \end{bmatrix} = \mathbf{p}_{\min}(1-w) + \min(\mathbf{p}_1, \mathbf{p}_2)w \tag{26}$$

$$\mathbf{os}_c^4 = \begin{bmatrix} os_1^4 & os_2^4 & \cdots & os_{no\_vars}^4 \end{bmatrix} = \frac{(\mathbf{p}_{\max} + \mathbf{p}_{\min})(1-w) + (\mathbf{p}_1 + \mathbf{p}_2)w}{2} \tag{27}$$

$$\mathbf{p}_{\max} = \begin{bmatrix} para_{\max}^1 & para_{\max}^2 & \cdots & para_{\max}^{no\_vars} \end{bmatrix} \tag{28}$$

$$\mathbf{p}_{\min} = \begin{bmatrix} para_{\min}^1 & para_{\min}^2 & \cdots & para_{\min}^{no\_vars} \end{bmatrix} \tag{29}$$

where $w \in \begin{bmatrix} 0 & 1 \end{bmatrix}$ denotes a weight to be determined by users, $\max(\mathbf{p}_1, \mathbf{p}_2)$ denotes a vector with each element obtained by taking the maximum among the corresponding element of $\mathbf{p}_1$ and $\mathbf{p}_2$. For instance, $\max(\begin{bmatrix} 1 & -2 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 3 & 1 \end{bmatrix}) = \begin{bmatrix} 2 & 3 & 3 \end{bmatrix}$. Similarly, $\min(\mathbf{p}_1, \mathbf{p}_2)$ gives a vector by taking the minimum value. For instance, $\min(\begin{bmatrix} 1 & -2 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 3 & 1 \end{bmatrix}) = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$. Among $\mathbf{os}_c^1$ to $\mathbf{os}_c^4$, the one with the largest fitness value is used as the offspring of the crossover operation. The offspring is defined as,

$$\mathbf{os} \equiv \begin{bmatrix} os_1 & os_2 & \cdots & os_{no\_vars} \end{bmatrix} = \mathbf{os}_c^{i_{os}} \tag{30}$$

where i_{os} denotes the index i which gives a maximum value of $f(\mathbf{os}_c^i)$, i = 1, 2, ,3 ,4.

If the crossover operation can provide a good offspring, a higher fitness value can be reached in less iteration. As seen from (24) to (27), the offspring spreads over the domain: (24) and (27) will move the offspring near centre region of the concerned domain (as w in (27) approaches 1, $\mathbf{os}_c^4$ approaches $\frac{\mathbf{p}_1 + \mathbf{p}_2}{2}$), and (25) and (26) will move the offspring near the domain boundary (as w in (25) and (26) approaches 1, $\mathbf{os}_c^2$ and $\mathbf{os}_c^3$ approaches $\mathbf{p}_{\max}$ and $\mathbf{p}_{\min}$ respectively). The chance of getting a good offspring is thus enhanced.

2. Mutation

The offspring (30) will then undergo the mutation operation. The mutation operation is to change the genes of the chromosomes. Consequently, the features of the chromosomes inherited from their parents can be changed. Three new offspring will be generated by the mutation operation:

$$\mathbf{nos}_j = \begin{bmatrix} os_1 & os_2 & \cdots & os_{no\_vars} \end{bmatrix} + \begin{bmatrix} b_1 \Delta nos_1 & b_2 \Delta nos_2 & \cdots & b_{no\_vars} \Delta nos_{no\_vars} \end{bmatrix}, \; j = 1, 2, 3 \tag{31}$$

where b_i, i = 1, 2, …, no_vars, can only take the value of 0 or 1; Δnos_i, i = 1, 2, …, no_vars, are randomly generated numbers such that $para_{\min}^i \le os_i^j + \Delta nos_i \le para_{\max}^i$. The first new offspring (j = 1) is obtained according to (31) with that only one b_i (i being randomly generated within the range) is allowed to be 1 and all the others are zeros. The second new offspring is obtained according to (31) with that some randomly chosen b_i are set to be 1 and others are zero. The third new offspring is obtained according to (31) with all $b_i = 1$. These three new offspring will then be evaluated using the fitness function of (21). A real number will be generated randomly and compared with a user-defined number $p_a \in [0 \quad 1]$. If the real number is smaller than p_a, the one with the largest fitness value f_l among the three new offspring will replace the chromosome with the smallest fitness f_s in the population (even when $f_l < f_s$.) If the real number is larger than p_a, the first offspring will replace the chromosome with the smallest fitness value f_s in the population if $f_l > f_s$; the second and the third offspring will do the same. p_a is effectively the probability of accepting a bad offspring in order to reduce the chance of converging to a local optimum. Hence, the possibility of reaching the global optimum is kept.

We have three offspring generated in the mutation process. From (31), the first mutation ($j = 1$) is in fact a uniform mutation. The second mutation allows some randomly selected genes to change simultaneously. The third mutation changes all genes simultaneously. The second and the third mutations allow multiple genes to be changed. Hence, the domain to be searched is larger as compared with a domain characterized by changing a single gene. As three offspring are produced in each generation, the genes will have a larger space for improving the fitness value when the fitness value is small. When the fitness values are large and nearly steady, changing the value of a single gene (the first mutation) may be enough as some genes may have reached the optimal values.

After the operation of selection, crossover, and mutation, a new population is generated. This new population will repeat the same process. Such an iterative process can be terminated when the result reaches a defined condition, e.g. a defined number of iterations have been reached.

V. Training of the NNDTF

In this section, the GA will be employed to train the parameters of the NNDTF to play Tic-tac-toe based on the gaming algorithm in Section II. The NNDTF with 9 inputs and 1 output is employed. The grids are numbered from 1 to 9 from right to left and from top to bottom. An "X" on the grid is denoted by 1, an "O" is denoted by –1, and an empty grid is denoted by 0.5. The grid pattern represented by numerical values will be used as the input of the NNDTF. The output of the NNDTF ($y(t)$ which a floating point number ranged from 1 to 9) represents the position of the marker that should be placed on. In order to have a legal move (place a marker on an empty grid), the marker is placed on an empty grid that has its grid number closest to the output of the network.

To perform the training, we have to determine the parameters to be trained and the fitness function describing the problem's objective. The parameters of the modified network to be turned is $[v_{ij} \quad m_s^j \quad \sigma_s^j \quad p_{j+1,j} \quad p_{j-1,j} \quad w_{jl} \quad m_o^l \quad \sigma_o^l]$ for all i, j, l, which will be chosen as the chromosome for the GA. 100 different training patterns (obtained based on the proposed gaming algorithm stated in Section II) are used to feed into the NNDTF for training. The fitness functions is designed as follows,

$$fitness = \frac{\sum_{i=1}^{100} S_m(y(t), \mathbf{x}(t))}{\sum_{i=1}^{100} S_{\max}(\mathbf{x}(t))} \tag{32}$$

where $y(t)$ denotes the output of the NNDTF with the t-th training pattern $\mathbf{x}(t)$ as the input, $S_m(y(t), \mathbf{x}(t))$ denotes the final score for grid $y(t)$ and the t-th training pattern $\mathbf{x}(t)$ based on the gaming algorithm. $S_{\max}(\mathbf{x}(t))$ denotes the maximum final score value among all the empty grids for the t-th training pattern $\mathbf{x}(t)$. The GA is to maximize the fitness value (ranged from 0 to 1) so as to force the output of the NNDTF to the grid number having the largest final score to ensure the best move.

VI. Example

In this session, a 9-input-1-output NNDTF will be used for training. The number of hidden nodes is chosen to be 8. 100 training patterns are used for training with 50000 iterations. The population size, probability of acceptance, and w are chosen to be 10, 0.5 and 0.1 respectively. After training, the fitness value obtained is 0.9605. The upper and lower bounds of each parameter are 1 and –1 respectively. The initial values of the parameters are generated randomly.

For comparison purpose, a traditional 3-layer feed-forward NN [18] trained by GA with arithmetic crossover and non-uniform mutation [18] is also applied under the same conditions to learn the gaming algorithm in Section II. The probabilities of crossover and mutation are selected to be 0.8 and 0.1 respectively. The shape parameter of the traditional GA for non-uniform mutation [18] is selected to be 5. These parameters are selected by trial and error for best performance. After training for 50000 iterations, the fitness value obtained is 0.9456.

To test the performance of our proposed method, our trained NN plays tic-tac-toe with the trained traditional NN for 50 games is used for comparison. The first 25 grid patterns, which are generated randomly with 2 “O”s and 2 “X”s, are the same as the next 25 grid patterns. For the first 25 games, the proposed approach moves first. For the second 25 games, the traditional approach moves first. The results are tabulated in Table I. It can be seen that the proposed approach performs better. The number of win is 18 by using NNDTF while the number of win is 13 only by using tradition NN.

VII. Conclusion

A neural network with double transfer functions and trained with genetic algorithm has been proposed. An algorithm of playing Tic-tac-toe has bee presented. A new transfer function of the neuron is proposed. The proposed neural; network is trained by genetic algorithm to learn the algorithm of playing Tic-tac-toe. As a comparison, the trained NN has played against a traditional NN trained by traditional GA. The result shows that the proposed approach performs better.

References:

[1] H.J.V.D. Herik, J.W.H.M. Uiterwijk and J.V. Rijswijck, “Games Solved: Now and in the Future,” *Artificial intelligence*, vol. 134, pp. 277-311, 2002.

[2] H.J.V.D. Herik, I.S. Herschberg, “The Construction of an Omniscient Endgame Data Base,” *ICCA J.*, vol. 8, no. 2, pp. 66-87, 1985.

[3] L.V. Allis, M.V.D. Meulen and.V.D. Herik, “Proof-Number Search,” *Artificial Intelligence*, vol. 66, no.1, pp. 91-124, 1994.

[4] G. Tesauro, “Programming Backgammon Using Self-Teaching Neural Nets,” *Artificial intelligence*, vol. 134, pp. 181-199, 2002.

[5] D.B. Fogel, “Using Evolutionary Progamming to Create Neural Networks that are capable of Playing Tic-Tac-Toe,” in *Proc. 1995 IEEE Int. Conf. Neural Networks*, San Francisco, CA, 1993, pp. 875-880.

[6] D.B. Fogel and K. Chellapilla, “Verifying Anaconda’s Expert Rating by Competing against Chinook: Experiments in Co-evolving a neural Checkers Player,” *Neurcomputing*, vol. 42, pp. 69-86, 2002.

[7] K. Chellapilla and D.B. Fogel, “Evolving Neural networks to Play Checkers Without Relying on Expert Knowledge,” *IEEE Trans. Neural Networks*, vol. 10, no. 6, pp. 1382-1391, 1999.

[8] K. Chellapilla and D.B. Fogel, “Evolving an Expert Checkers Playing Program Without Using Human Expertise,” *IEEE Trans. Evolutionary Computation*, vol. 5, no. 4, pp. 422-428, 2001.

[9] K. Chellapilla and D.B. Fogel, “Evolution, Neural Networks, Games, and Intelligence,” *Proceedings of The IEEE*, vol. 87, no. 9, pp. 1471-1496, 1999.

[10] S.Y. Chong, M.K. Tan, J.D. White, “Observing the evolution of neural networks learning to play the game of Othello,” *IEEE Trans. Evolutionary Computation*, vol. 9, no. 3, pp. 422-428, 2005.

[11] D.E. Beal and M.C. Smith, “Random Evolutions in Chess,” *ICCA J.*, vol. 17, no. 1, pp. 3-9, 1994.

[12] F.H.F. Leung, H.K. Lam, S.H. Ling**,** and P.K.S. Tam, “Tuning of the structure and parameters of neural network using an improved genetic algorithm,” *IEEE Trans. Neural Networks*, vol.14, no. 1, pp.79–88, Jan. 2003.

[13] M. Brown and C. Harris, *Neuralfuzzy adaptive modeling and control*. Prentice Hall, 1994.

[14] S.H. Ling, F.H.F. Leung, H.K. Lam, Y.S. Lee, and P.K.S. Tam, “A novel GA-based neural network for short-term load forecasting,” *IEEE Trans. Industrial Electronics*, vol. 50, no. 4, pp.793–799, Aug. 2003.

[15] J.H. Holland, *Adaptation in natural and artificial systems*, Ann Arbor, MI: University of Michigan Press, 1975.

[16] D.T. Pham and D. Karaboga, *Intelligent optimization techniques, genetic algorithms, tabu search, simulated annealing and neural networks*. Springer, 2000.

[17] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution Programs*, (2n ed.). Springer-Verlag, 1994.

[18] S. Haykin, *Neural networks: A comprehensive foundation*, (2n ed.). Upper Saddle River, N.J.: Prentice Hall, 1999.

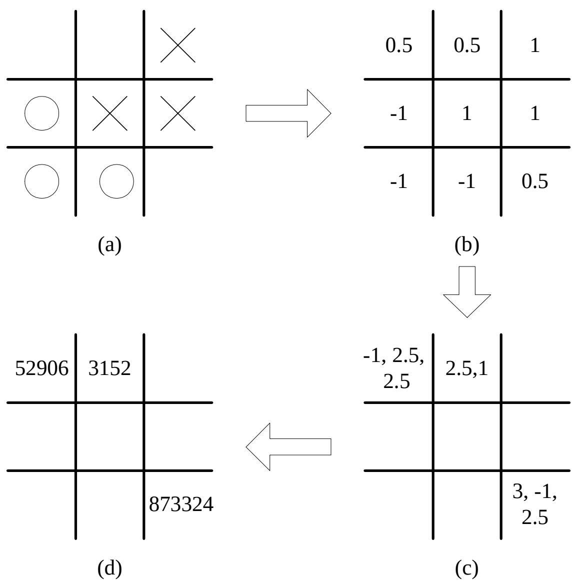

Fig. 1. Evaluation process.

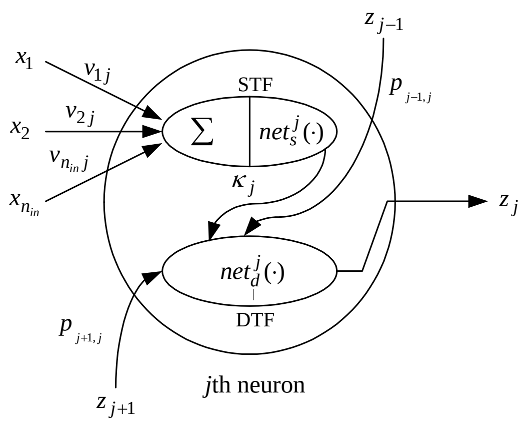

Fig. 2. Model of the proposed neuron.

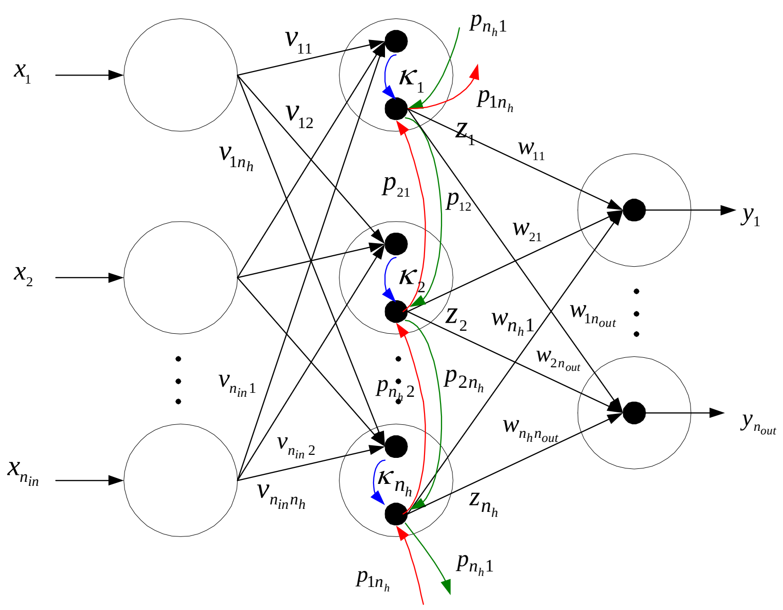

Fig. 3. Connection of the NNDTF.

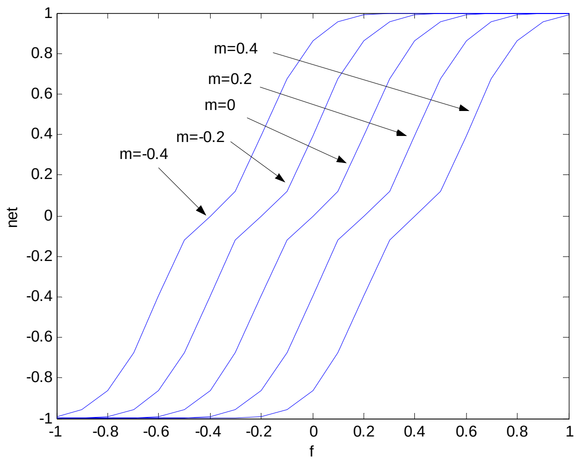

Fig. 4. Sample transfer functions of the proposed neuron (σ =0.2).

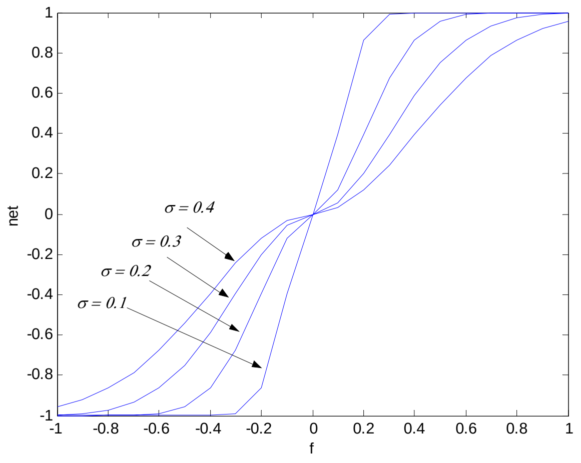

Fig. 5. Sample transfer functions of the proposed neuron (m=0).

```
Procedure of the simple GA
begin
        $\tau \to 0$      // $\tau$: iteration generation
initialize $\mathbf{P}(\tau)$            //$\mathbf{P}(\tau)$: population for iteration $t$
        evaluate $f(\mathbf{P}(\tau))$          // $f(\mathbf{P}(\tau))$:fitness function
while (not termination condition) do
      begin
          $\tau \to \tau+1$
          select 2 parents $\mathbf{p}_1$ and $\mathbf{p}_2$ from $\mathbf{P}(\tau-1)$
          perform genetic operations (crossover and mutation)
          reproduce a new $\mathbf{P}(\tau)$
          evaluate $f(\mathbf{P}(\tau))$
      end
end
end
```

Fig. 6. Procedure of simple GA.

```
Procedure of the improved GA
begin
        $\tau \to 0$      // $\tau$: iteration
        initialize $\mathbf{P}(\tau)$              //$\mathbf{P}(\tau)$: population for iteration $t$
        evaluate $f(\mathbf{P}(\tau))$ // $f(\mathbf{P}(\tau))$:fitness function
while (not termination condition) do
        begin
        $\tau \to \tau+1$
        select 2 parents $\mathbf{p}_1$ and $\mathbf{p}_2$ from $\mathbf{P}(\tau-1)$
        perform crossover operation according to equations (7) to (13)
          perform mutation operation according to equation (14) to three
          offspring $\mathbf{nos}_1$, $\mathbf{nos}_2$ and $\mathbf{nos}_3$
        // reproduce a new $\mathbf{P}(\tau)$
                if random number < $p_a$   // $p_a$: probability of acceptance
                    The one among $\mathbf{nos}_1$, $\mathbf{nos}_2$ and $\mathbf{nos}_3$ with the largest fitness
                    value replaces the chromosome with the smallest fitness
                    value in the population
                else
                    if $f(\mathbf{nos}_1)$ > smallest fitness value in the $\mathbf{P}(\tau-1)$
                        $\mathbf{nos}_1$ replaces the chromosome with the smallest fitness
                        value
                    end
                    if $f(\mathbf{nos}_2)$ > smallest fitness value in the updated $\mathbf{P}(\tau-1)$
                        $\mathbf{nos}_2$ replaces the chromosome with the smallest fitness
                        value
                    end
                    if $f(\mathbf{nos}_3)$ > smallest fitness value in the updated $\mathbf{P}(\tau-1)$
                        $\mathbf{nos}_3$ replaces the chromosome with the smallest fitness
                        value
                    end
                end
```

Fig. 7. Procedure of the modified GA.

	Proposed NN Wins:	Draws:	Loses
Proposed approach moves first	18:	3:	4
Traditional approach moves first	13:	4:	8

Table I: Results of the proposed NN playing Tic-tac-toe against with the traditional NN for 50 games.