Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 46 (2003) 1493–1509
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Genetic Neural Networks to Approximate Feedback Nash Equilibria in Dynamic Games

S. SIRAKAYA*
Departments of Economics and Statistics and Center for Statistics and Social Sciences
University of Washington, Seattle, WA 98195-4320, U.S.A.
sirakaya@stat.washington.edu

N. M. ALEMDAR
Department of Economics, Bilkent University, 06800 Bilkent, Ankara, Turkey
alemdar@bilkent.edu.tr

(Received October 2001; revised and accepted June 2003)

Abstract—This paper develops a general purpose numerical method to compute the feedback Nash equilibria in dynamic games. Players' feedback strategies are first approximated by neural networks which are then trained online by parallel genetic algorithms to search over all time-invariant equilibrium strategies synchronously. To eliminate the dependence of training on the initial conditions of the game, the players use the same stationary feedback policies (the same networks), to repeatedly play the game from a number of initial states at any generation. The fitness of a given feedback strategy is then computed as the sum of payoffs over all initial states. The evolutionary equilibrium of the game between the genetic algorithms is the feedback Nash equilibrium of the dynamic game. An oligopoly model with investment is approximated as a numerical example.

Keywords—Feedback Nash equilibrium, Parallel genetic algorithms, Neural networks.

1. INTRODUCTION

This paper proposes a new method that can efficiently offer highly accurate approximations to feedback Nash equilibria of dynamic games which lack explicit solutions. We focus on feedback perfect state information pattern and parameterize each feedback rule by the weights of a suitable neural network, thereby transforming the players' strategy spaces from a set of rules to a set of neural net weights. For each network, an artificially intelligent player, a genetic algorithm (GA), is assigned to breed fitter weights and update other GAs via the computer shared memory as to its best-to-date feedback rule.

In any given generation, the game starts from a fixed set of initial states. Since we are searching for the stationary feedback rules, the same neural net weights are used to compute the payoffs. The raw fitness of a potential feedback rule is then calculated as the sum of all payoffs across all initial states. Our rationale for repeating the game from multiple initial states is two-fold: to

*Author to whom all correspondence should be addressed.

0898-1221/03/$ - see front matter Typeset by AMS-TEX
doi: 10.1016/S0898-1221(03)00378-X

avoid the dependence of the weight training on the initial conditions and also to speed up GA learning.

In essence, each GA trains a neural network online to best respond to the other players' best feedback rule in the previous generation. Initially, GAs start a blind search. However, as the search progresses, each player's network is *trained* to incrementally best respond to any policy rule of the others. The fittest weights are then communicated to each other via the computer shared memory. Equipped with the updated weights, each player is better able to anticipate the best response of the others for all potential rules in its population. The individuals that exploit this knowledge to their advantage are fitter, thus, reproducing faster. Ultimately, the fitter weights dominate respective populations also steering the other players' search for the best set of rules to the vicinity of the feedback Nash equilibrium.

The feedback Nash equilibrium is a desirable solution concept because, by definition, it is time-consistent. Unfortunately, however, feedback solutions are often analytically intractable. Explicit solutions are restricted to a class of special cases, for instance, a linear quadratic game where objective functions are quadratic, laws of motion are linear, and control variables are unconstrained. For problems with nonlinear function specifications and constrained decision variables, on the other hand, linearly parameterized numerical approximations, such as spline functions or radial basis functions (RBFs), are frequently employed. Other dynamic programming techniques which rely on the approximation of the Bellman equation are also often used. As the state space gets larger, however, the aforementioned methods quickly become unmanageable due to exponentially rising computer memory requirements also known as the curse of dimensionality. The proposed method in this paper is free of the so-called curse of dimensionality as both GAs and neural networks are inherently parallel structures thus are highly efficient users of computer memory.

The balance of the paper is as follows. In Section 2, we briefly discuss the concepts and solutions involved in dynamic games. Section 3 first presents a short overview of the neural networks and genetic algorithms, and then proceeds to show how parallel neural genetic algorithms can approximate the feedback Nash equilibria. Section 4 tests the algorithm on the dynamic oligopoly game provided in [1]. Conclusions follow.

2. DYNAMIC GAMES AND SOLUTION CONCEPTS

For presentational simplicity, consider the following generic two-player finite-horizon dynamic game in discrete time, $t \in T$. For players, $i = 1, 2$, let the m_i-dimensional state, $x_t^i \in X_t^i \subset \mathcal{R}^{m_i}$, evolve according to[1]

$$x_{t+1}^i = F_t^i\left(x_t^1, x_t^2, u_t^1, u_t^2\right), \qquad x_0^i \text{ is given}, \tag{1}$$

and where $u_t^i \in U_t^i \subset \mathcal{R}^{n_i}$ denotes the n_i-dimensional control vector for $t = \{0, 1, 2, \ldots, T-1\}$. Given any strategy of the other, player i chooses his strategy to maximize the objective functional

$$J^i\left(u^1, u^2\right) = \sum_{t=0}^{T-1} R_t^i\left(x_{t+1}^1, x_t^1, x_{t+1}^2, x_t^2, u_t^1, u_t^2\right). \tag{2}$$

The terms *open-loop* and *closed-loop* refer to the information structures in dynamic games. In the former case, players' information sets consist of the initial values of the state variables and time so that right at the beginning of the game, players announce and commit themselves, with no possibility of update or alteration throughout the game, to a path of controls which are functions of time and the initial states, $u^i = \{u_t^i(x_0^i)\}_{t=0}^{T-1}$. The relevant equilibrium concept in this class of strategies is referred to as *open-loop Nash equilibrium*. Open-loop Nash equilibria are only weakly time-consistent and therefore, in general, not subgame-perfect.[2]

[1]For numerical computation, infinite-horizon differential games can be discretized as in [2] so that the discussions that follow can be extended without any loss of generality.

[2]See, among others, [3–7] for classes of games in which open-loop Nash equilibria are subgame-perfect.

Note that by recursive substitution of equation (1) into (2), the objective functionals depend only on controls and the initial states

$$\bar{J}^i\left(u^1, u^2\right) = \sum_{t=0}^{T-1} \bar{R}_t^i\left(x_0^1, x_0^2, u_t^1, u_t^2\right). \tag{3}$$

Thus, a pair of open-loop strategies, (u^{1*}, u^{2*}), constitutes an open-loop Nash equilibrium of the game defined in equations (1) and (2) if and only if

$$\begin{aligned} \bar{J}^1\left(u^{1*}, u^{2*}\right) &\geq \bar{J}^1\left(u^1, u^{2*}\right), \\ \bar{J}^2\left(u^{1*}, u^{2*}\right) &\geq \bar{J}^2\left(u^{1*}, u^2\right), \end{aligned} \tag{4}$$

for all permissible open-loop strategies u^1 and u^2.

The above inequalities mean that given the initial date and state, each player's precommitted action path constitutes a best response to other players' chosen paths. However, this may not be true for the continuation of this strategy when viewed from any other intermediate time and state. Thus, in general, equilibrium policies generated by the open-loop decision rule are not subgame-perfect.

Under the closed-loop information structure, players' strategies may depend on the whole history of the dynamical system. Players do not precommit but revise their decisions as new information becomes available. Information sets of players in this case may consist of the values of state variables up to the current time (closed-loop perfect state information pattern), the initial and current values of the state variables (closed-loop memoryless information pattern) or the current values of the state variables (feedback perfect state information pattern). The relevant equilibrium in this case is called the *closed-loop Nash equilibrium*. In general, a plethora of closed-loop Nash equilibria exist when the information structure for at least one player is dynamic and involves memory [8]. In this case, the Nash equilibrium concept can be refined by requiring it to be subgame-perfect. Such an equilibrium is called a *feedback Nash equilibrium*.

In this paper, we focus on equilibria that are subgame-perfect, the so-called feedback Nash equilibria. Feedback Nash equilibrium is defined by inequalities (4) with the added requirement that they hold for all time and state pairs. Requirements for the feedback Nash equilibrium are compatible with any information pattern in which all players have access to the current value of the state variables [8]. However, in order to narrow the search space, we restrict ourselves to the feedback information structure and focus on time-invariant feedback rules. Thus, a control law, γ^i, for player i is a function of the current state of the system

$$u_t^i = \gamma^i\left(x_t^1, x_t^2\right). \tag{5}$$

A pair of closed-loop strategies $(\gamma^{1*}, \gamma^{2*})$ is a feedback Nash equilibrium of the game defined in equations (1) and (2) if and only if

$$\begin{aligned} \bar{J}^1\left(\gamma^{1*}, \gamma^{2*}\right) &\geq \bar{J}^1\left(\gamma^1, \gamma^{2*}\right), \\ \bar{J}^2\left(\gamma^{1*}, \gamma^{2*}\right) &\geq \bar{J}^2\left(\gamma^{1*}, \gamma^2\right), \end{aligned} \tag{6}$$

for all admissible closed-loop strategies γ^1 and γ^2.

3. A BRIEF NOTE ON NEURAL NETWORKS AND GENETIC ALGORITHMS

We approximate players' feedback rules by feedforward neural networks which are then trained online synchronously by genetic algorithms in a parallel manner to search over all time-invariant equilibrium strategies. Neural network specification offers some advantages over the traditional

techniques, such as spline functions or RBFs. First, feedforward neural networks have been proven to be universal function approximators in the sense any L^2 function can be approximated arbitrarily well by a feedforward neural network with at least one hidden layer. (See, for example, [9,10].) Second, nonlinearly parameterized nature of feedforward neural networks allow them to use fewer parameters to achieve the same degree of approximation accuracy as opposed to linearly parameterized techniques which require an exponential increase in the number of parameters. Third, neural networks with sigmoid activation function at the output layer naturally deliver control bounds, while such bounds constitute a major problem for linearly parameterized techniques. Fourth, neural networks can easily be applied to problems which admit bang-bang solutions, while this constitutes a major difficulty for other conventional methods.

There are several good reasons for our choice of genetic algorithms to train the neural networks as well. First, as opposed to the gradient-descent methods, genetic implementations do not use the gradient information. Thus, they do not require the continuity and the existence of derivatives of the objective functionals and state transition functions. The only restrictions are that they be bounded. Naturally then, our method can be applied to a larger class of problems. Second, GAs are global search algorithms which start completely blind and learn as the game unfolds. Regardless of the initial parameter values, they ensure convergence to an approximate global optimum by exploiting the domain space and relatively better solutions through genetic operators. Gradient-descent methods, on the other hand, need gradient information and may stuck in a local optimum or fail to converge at all, depending on the initial parameters.

3.1. Neural Networks

Neural networks are information-processing paradigms that mimic highly interconnected, parallelly structured biological neurons. They are trained to learn and generalize from a given set of examples by adjusting the synaptic weights between the neurons.[3]

Consider an L layer (or $L-1$ hidden layer) feedforward neural network, with the input vector $z^0 \in \mathcal{R}^{r_0}$ and the output vector $\phi(z^0) = z^L \in \mathcal{R}^{r_L}$. As in [11], we refer to this class of networks as $\mathcal{N}^L_{r_0,r_1,\ldots,r_L}$. The recursive input-output relationship is given by

$$y^j = w^j z^{j-1} + v^j, \tag{7}$$

$$z^j = \hat{\sigma}_j\left(y^j\right) = \left(\sigma_j\left(y^j_1\right), \sigma_j\left(y^j_2\right), \ldots, \sigma_j\left(y^j_{r_j}\right)\right)', \tag{8}$$

where the connection and the bias weights are, respectively, $\omega = \{w^j, v^j\}$, with $w^j \in \mathcal{R}^{r_j \times r_{j-1}}$ and $v^j \in \mathcal{R}^{r_j}$ for $j = 1, 2, \ldots, L$. The dimension of y^j and z^j is denoted by r_j. The scalar activation functions, $\sigma_j(.)$ are usually sigmoids, e.g., $\sigma_j(.) = \tanh(.)$ or $\sigma_j(.) = 1/(1+\exp(-(.))$ in the hidden layers. At the output layer, the activation functions, $\sigma_L(.)$, can be linear, e.g., $\sigma_L(.) = (.)$, if the outputs have no natural bounds. If, however, they are bounded by $\theta_{\min} \le z^L \le \theta_{\max}$, then one may choose

$$\sigma_L(.) = \theta_{\min} + \frac{\theta_{\max} - \theta_{\min}}{1 + \exp(-(.))}. \tag{9}$$

Thus, the approximating function has the general representation

$$\phi\left(z^0, \omega\right) = \hat{\sigma}_L\left(w^L \hat{\sigma}_{L-1}\left(w^{L-1}\hat{\sigma}_{L-2}\left(\cdots\left(w^2\hat{\sigma}_1\left(w^1\zeta + v^1\right) + v^2\right) + v^3\right) + \cdots + v^{L-1}\right) + v^L\right). \tag{10}$$

3.2. Genetic Algorithms

The neural networks in our game algorithm are trained online by parallel genetic algorithms. That is, the interconnection weights between the neurons are incrementally adjusted by synchronous parallel GAs. A basic GA consists of iterative procedures, called generations. In each

[3]For the sake of compactness, the notation this section closely follows [11]. A well-documented theory of neural networks can be found in [12,13].

generation, say s, a GA maintains a constant size population, Pop(s), of candidate solution vectors to the problem at hand. Each individual in Pop(s) is coded as a finite-length string, usually over the binary alphabet ($\{0,1\}$). The initial population, Pop(0), is generally random.

At any generation, each individual in a population is assigned a 'fitness score' depending on how good a solution it is relative to the population. During a single reproduction phase, relatively fit individuals are selected from a pool of candidates some of which are recombined to generate a new generation. Better solutions breed faster while bad solutions vanish. Basic recombination operators are *mutation* and *crossover*.

Crossover randomly chooses two members ('parents') from the population, then creates two similar off-spring by swapping the corresponding segments of the parents. Crossover can be considered as a way of further exploration by exchanging information between two potential solutions. Mutation randomly alters single bits of the bit strings encoding individuals with a probability equal to the mutation rate pmut. It can be interpreted as experimenting to breed fitter solutions.

GAs are highly parallel mathematical structures. While they operate on individuals in a population, they collect and process vast amounts of information by exploiting the similarities in classes of individuals, which Holland [14] calls *schemata*. These similarities in classes of individuals are defined by the lengths of common segments of bit strings. By operating on n individuals in one generation, a GA collects information approximately about n^3 individuals [14].

Parallelism can be explicit as well in the sense that more than one GA can generate and collect data independently and that genetic operators may be implemented in parallel (see [15]). Parallel genetic algorithms are inspired by the biological evolution of species in isolated locales. To mimic this evolutionary process, a population is divided into subpopulations and a processor is assigned to each to separately apply genetic operators while allowing for periodic communication between them. Subpopulations, specialize on one portion of the problem and communicate among themselves to learn about the remainder. We exploit this idea in training the neural networks to approximate the feedback Nash equilibria.[4]

3.3. Approximation of Feedback Nash Equilibrium with Genetic Neural Networks

For training the neural networks, we propose the following algorithm. First, parameterize the feedback control law of player i in the game given by equations (1) and (2) by a neural network as

$$\gamma^i\left(x_t^1, x_t^2\right) = \phi^i\left(z_t^0, \omega^i\right), \tag{11}$$

where z_t^0 is the time t input vector to the network approximating the feedback control law of player i. It is an r_0-dimensional vector of the state variables at time t, such as $z_t^0 = (x_t^1, x_t^2)$ or $z_t^0 = (x_t^1, x_t^1, x_t^2)$. Note that the neural network architectures chosen to approximate the feedback rules of Player 1 and 2 and/or their input vectors may not be the same, depending on the game. For notational simplicity, however, we assume here that the input vectors for both networks are the same. Next, for each player i, use equation (11) in the state transition equation (1) to get x_{t+1}^i (thus, z_{t+1}^0) as

$$x_{t+1}^i = F_t^i\left(\phi^1\left(z_t^0, \omega^1\right), \phi^2\left(z_t^0, \omega^2\right), x_t^1, x_t^2\right), \tag{12}$$

where $x_0^i \in X_0^i$ is given. Substituting equation (11) and recursively substituting equation (12) into (2), we have

$$\tilde{J}^i\left(\omega^1, \omega^2\right) = \sum_{t=0}^{T-1} \tilde{R}_t^i\left(x_0^1, x_0^2, \phi^1\left(z_t^0, \omega^1\right), \phi^2\left(z_t^0, \omega^2\right)\right). \tag{13}$$

[4]In [2,16], parallel GAs are employed to approximate open-loop Nash equilibrium in discrete and differential games, respectively. In [17], GA parallelism is used to compute the Stackelberg equilibrium in sequential games.

Note that given the initial states, the strategy spaces of the players are now transformed from a set of rules to a set of neural net weights. Therefore, the weights of the networks can be adjusted to maximize each player's respective payoff given the weights of the other. The ability of the networks so trained to generalize, however, will be limited as the training depends upon the initial conditions of the game.

Thus, our next task is to devise a method so that the trained networks can generalize over a wider set of initial states. Towards that, we note that if a stationary feedback policy of a player maximizes its respective payoff for any given initial state, then it must also maximize the sum of payoffs over a set of initial states. Defining a set of initial states as $\Pi \subset X_0^1 \times X_0^2$, for any given set of weights and initial states, $(x_0^1, x_0^2) \in \Pi$, we can generate the sample paths for the states (thus, the input vectors at any time $t \in [0, T-1]$) and feedback policies from (11) and (12), as $\{x_t^i\}_{t=0}^{T-1}$, and $\{\phi^i(z_t^0, \omega^i)\}_{t=0}^{T-1}$, respectively. Thus, the sum of payoffs over all initial states is

$$\hat{J}^i\left(\omega^1, \omega^2\right) = \sum_{(x_0^1, x_0^2) \in \Pi} \sum_{t=0}^{T-1} \hat{R}_t^i\left(x_0^1, x_0^2, \phi^1\left(z_t^0, \omega^1\right), \phi^2\left(z_t^0, \omega^2\right)\right). \tag{14}$$

Hence, if the neural nets approximating the stationary feedback policies are trained so as to maximize this sum, then they will have a better generalizing capacity.

In passing, we note that many neural networks can parameterize the feedback rule $\gamma^i(x_t^1, x_t^2)$. There exists no hard-and-fast rule of choosing a network architecture other than a systematic trial and error approach. While a network architecture with too many layers and neurons may be very time consuming and may not offer significant improvement over an architecture with fewer layers and neurons, too few layers and neurons may result in poor approximations. As a general rule, simpler architectures are more preferable because they learn faster.

To approximate the feedback rules, we assign GA^i to Player i $(i = 1, 2)$ to train the neural networks as represented in equation (11).[5] At any generation $s \in S$, a GA^i operates on a constant size population, N_i, of neural net weights

$$\mathrm{Pop}^i(s) = \left\{\omega_1^i(s), \omega_2^i(s), \ldots, \omega_b^i(s), \ldots, \omega_{N_i}^i(s)\right\},$$

where $\omega_b^i(s) \in \mathrm{Pop}^i(s)$ represents a vector of potential weights approximating the optimal feedback rule of Player i.

GA^1 evaluates each individual $b \in \mathrm{Pop}^1(s)$ by computing its raw fitness,

$$\hat{J}^i\left(\omega_b^1(s), \omega^{2*}(s-1)\right) = \sum_{(x_0^1, x_0^2) \in \Pi} \sum_{t=0}^{T-1} \hat{R}_t^i\left(x_0^1, x_0^2, \phi^1\left(z_t^0, \omega_b^1(s)\right), \phi^2\left(z_t^0, \omega^{2*}(s-1)\right)\right),$$

where $\omega^{2*}(s-1)$ stands for the vector of best-to-date weights in the population of GA^2 in the previous generation. GA^2, on the other hand, computes the raw fitness, $\hat{J}^2(\omega^{1*}(s-1), \omega_d^2(s))$, for each individual $d \in \mathrm{Pop}^2(s)$ given the best-to-date weights, $\omega^{1*}(s-1)$, of GA^1.

The search is initialized from random populations, $\mathrm{Pop}^1(0)$ and $\mathrm{Pop}^2(0)$. Given a random $d \in \mathrm{Pop}^2(0)$, GA^1 finds the best performing individual, b, such that

$$\hat{J}^1\left(\omega_b^1(0), \omega_d^2(0)\right) \geq \hat{J}^1\left(\omega_g^1(0), \omega_d^2(s)\right),$$

for $g = 1, 2, \ldots, b-1, b+1, \ldots, N_1$, and updates GA^2 with $\omega^{1*}(0) = \omega_b^1(0)$. Likewise, GA^2 simultaneously informs GA^1 about $\omega^{2*}(0)$. Next, using the evolutionary operators, a new generation of populations are formed from the relatively fit individuals. Their fitness scores are recalculated in the light of the previous choices of the opponent, and best performing individuals are exchanged once again.

[5] If players have more than one feedback rule, as in our example, then they are assigned a GA for each.

The above procedures are repeated for a number of generations. That is, at any generation s. GA^1 proceeds with the search if there exists a b such that

$$\hat{J}^1\left(\omega_b^1(s), \omega^{2*}(s-1)\right) \geq \hat{J}^1\left(\omega_g^1(s), \omega^{2*}(s-1)\right),$$

for $g = 1, 2, \ldots, b-1, b+1, \ldots, N_1$. Similarly, GA^2 continues the search if there exists a d such that

$$\hat{J}^2\left(\omega^{1*}(s-1), \omega_d^2(s)\right) \geq \hat{J}^2\left(\omega^{1*}(s-1), \omega_g^2(s)\right),$$

for $g = 1, 2, \ldots, d-1, d+1, \ldots, N_2$.

As the search evolves, fitter individuals proliferate, thanks to the reproduction and crossover operators, until $s' \leq S$ whence for any $s \geq s'$, there exist *no* individuals $b \in \text{Pop}^1(s)$ and $d \in \text{Pop}^2(s)$ such that

$$\hat{J}^1\left(\omega_b^1(s), \omega_F^2\right) > \hat{J}^1\left(\omega_F^1, \omega_F^2\right)$$

and

$$\hat{J}^2\left(\omega_F^1, \omega_d^2(s)\right) > \hat{J}^1\left(\omega_F^1, \omega_F^2\right),$$

where ω_F^1 and ω_F^2 are the weights that best approximates the feedback equilibrium policies, γ^{1*} and γ^{2*}, respectively.

The following pseudocode outlines the steps involved in our parallel GA search for the feedback Nash equilibrium in a two-player dynamic game.

```
procedure GA^1
begin
 Randomly initialize Pop^1(0);
 copy initial weights to shared memory;
 synchronize;
 evaluate Pop^1(0);
 s = 1;
 repeat
   select Pop^1(s) from Pop^1(s-1);
   copy best best weights to shared memory;
   synchronize;
   crossover and mutate Pop^1(s);
   evaluate Pop^1(0);
   s = s + 1;
 until(termination condition);
end;
```

```
procedure GA^2
begin
 Randomly initialize Pop^2(0);
 copy initial weights to shared memory;
 synchronize;
 evaluate Pop^2(0);
 s = 1;
 repeat
   select Pop^2(s) from Pop^2(s-1);
   copy best weights to shared memory;
   synchronize;
   crossover and mutate Pop^2(s);
   evaluate Pop^2(0);
   s = s + 1;
 until(termination condition);
end;
```

As for the potential difficulties in GA training, the so-called competing conventions problems may arise since structurally different networks can functionally be equivalent. Genetic algorithms operate on genotypes which represent a network structure. Consequently, structurally different networks are represented by different genotypes. If some structures are functionally equivalent, then crossover operator may degenerate the search by creating inferior offsprings. Specifically, the farther apart are the weights of a node and nodes of different layers located on a chromosome, more likely it is for the standard one-point crossover operator to disrupt them [18]. Thus, we place all incoming weights of a node and all nodes side by side to help resolve this problem.

Furthermore, crossover operation may also become destructive when more than one feedback rules are trained by a single GA. An explicit parallelism such as ours will alleviate this problem as each feedback policy is evolved by a separate GA. In the oligopoly game between two firms,

each player has two feedback rules. Thus, we have four GAs evolving four separate populations and updating each other synchronously.

A word of caution is in order about the selection operator as well. Note that at any generation, s, the players are supplied with only a limited number of sample paths to train their neural nets so that they have to learn online. Consequently, the weights that out-perform others early in the search may actually do poorly over the range of the other players' strategy spaces. Moreover, given the other player's previous rules, there may exist more than one unique vector of weights in the population mapping into the player's same best responses. Again, the search may stagnate if the rule which performs poorly over the range of other player's strategies is copied to the memory. An elitist selection strategy to form new generations will fail on both accounts. Moreover, the search terrain for the neural network generally is highly nonlinear. Thus, it becomes imperative that a selection procedure be adopted that will sustain the evolutionary pressure.

In our simulations, we use *fitness rank selection* as the selection procedure. With fitness rank selection, individuals are first sorted according to their raw fitness, and then using a linear scale reproductive fitness scores assigned according to their ranking. Rank selection prevents premature convergence since the raw fitness values have no direct impact on the number of offspring. The individual with the highest fitness may be much superior to the rest of the population or it may be just above the average; in any case, it will expect the same number of offspring. Thus, superior individuals are prevented from taking over the population too early causing a false convergence. The players' difficulties with their search may be further compounded due to the fact that it may be over a highly nonlinear terrain. Thus, the likelihood that the search may get stuck at a local optimum is quite high. Rank selection performs better under both conditions.

Finally, GAs may be computationally more time consuming as compared with the conventional training techniques. Nevertheless, we would like to emphasize the contribution of our method in that it is an *online* algorithm that can tackle problems that are otherwise analytically and computationally intractable.

4. AN EXAMPLE: A DYNAMIC OLIGOPOLY GAME WITH CAPITAL INVESTMENT

Consider two firms producing two goods that are close substitutes. The goods are assumed to be infinitely perishable so that there is no possibility of a carryover. In each period t, firms must decide as to how much to produce, q_t^i and how much to invest, x_t^i. Prices are determined by short-term market clearing conditions (Cournot competition) resulting in inverse demand functions

$$p_t^i = \tilde{D}^i\left(q_t^1, q_t^2\right). \tag{15}$$

The capital accumulation of firm i $(i = 1, 2)$ is described by

$$k_{t+1}^i = x_t^i + (1 - \varepsilon)\, k_t^i, \qquad k_0^i \text{ given}, \tag{16}$$

where $0 < \varepsilon < 1$ is the constant depreciation rate.

The cost of production $C^i(q_t^i, k_t^i)$ depends on the quantity produced and the firm i's capital stock. Investments add to the following period's level of capital with an adjustment cost $h^i(x_t^i)$. Now, firm i solves

$$\max_{x_t^i \geq 0,\, q_t^i \geq 0} J^i = \sum_{t=0}^{\infty} \delta^t \left[p_t^i q_t^i - C^i\left(k_t^i, q_t^i\right) - h^i\left(x_t^i\right)\right] \tag{17}$$

subject to equations (12) and (13), where $0 < \delta < 1$ is the fixed discount rate.

Miranda and Vedenov [1] use the Chebychev orthogonal collocation method to solve the Bellman functional equations of the dynamic duopoly game above. The method approximates the Bellman value functions using an unknown linear combination of Chebychev polynomial basis

functions. The unknown coefficients are then fixed by requiring the Bellman equations to be satisfied, not at all possible points, but a small number of judiciously chosen "collocation" nodes. By selecting an equal number of nodes and known basis functions, an approximate solution to the Bellman equations are computed by solving a finite-dimensional nonlinear root finding problem using standard function iteration, Newton, or quasi-Newton methods.

Using our algorithm, we search for a stationary feedback Nash equilibrium of the above game, that is, a profile of time-invariant feedback investment and production policy rules that yield a Nash equilibrium in every proper subgame.

For the game given above, linear-quadratic method requires linear inverse demand function and quadratic production and investment costs that are symmetric with respect to some points. This also implies that the effect of capital stock must be restricted to parallel shifting of the marginal cost curve (cf. [19]). Our solution algorithm, on the other hand, restricts neither the structure of these functions, nor the way capital affects the production costs.

In the following simulations, the dynamic duopoly game is parameterized as in [1]. The assumed inverse demands $\tilde{D}^i$ $(i = 1, 2)$ are log-log functions of both quantities:

$$\log p_t^1 = \log A_1 + \epsilon_{11} \log q_t^1 + \epsilon_{12} \log q_t^2,$$
$$\log p_t^2 = \log A_2 + \epsilon_{21} \log q_t^1 + \epsilon_{22} \log q_t^2.$$

We further suppose that the cost of production is a simple Cobb-Douglas function of quantity produced and capital stock

$$C^i\left(q_t^i, k_t^i\right) = a_i \left(q_t^i\right)^{\alpha} \left(k_t^i\right)^{\beta}, \qquad \alpha \geq 0, \quad \beta \leq 0.$$

The costs of investments, on the other hand, are symmetric and assumed to be simple power functions

$$h^i\left(x_t^i\right) = \frac{c_i}{\theta} \left(x_t^i\right)^{\theta}.$$

The simulation parameters are

$$A_1 = A_2 = 1.5, \quad \epsilon_{11} = \epsilon_{22} = -0.5, \quad \epsilon_{12} = \epsilon_{21} = -0.2, \quad a_1 = a_2 = 0.25,$$
$$\alpha_1 = \alpha_2 = 1.5, \quad \beta_1 = \beta_2 = -0.75, \quad c_1 = c_2 = 1.5, \quad \theta = 0.25.$$

As for the neural net architectures, after a round of experimentations, we adopt a fully connected feedforward network from $\mathcal{N}_{2,2,1}^2$ to parameterize the feedback production rule, $q^i(k_t^1, k_t^2)$, as

$$q^i\left(k_t^1, k_t^2\right) = \phi^i\left(\nu_t^0, \omega^{i,q}\right), \tag{18}$$

for each player $i = 1, 2$ (see Figure 1). The input vector to the networks at time t is $\nu_t^0 = (k_t^1, k_t^2)$. In the hidden layer, the activation functions are linear. At the output level, the squashing functions are as in equation (9) except that we also let GAs search $\theta_{\min}$ and $\theta_{\max}$ rather than fixing them ahead.

Feedback investment policies, $x^i(k_t^1, k_t^2)$, on the other hand, are represented by a disconnected feedforward neural network from $\mathcal{N}_{4,4,1}^2$ (see Figure 2) as

$$x^i\left(k_t^1, k_t^2\right) = \varphi^i\left(\tau_t^0, \omega^{i,x}\right). \tag{19}$$

The input vector to the networks at time t is $\tau_t^0 = (k_t^1, k_t^1, k_t^2, k_t^2)$. We use sigmoid activation functions at both the hidden and the output layers.

Two GAs are then assigned to player i: $\mathrm{GA}^{i,q}$ trains the neural net parameterizing the feedback production rule, $\phi^i(\nu_t^0; \omega^{i,q})$, and $\mathrm{GA}^{i,x}$ evolves the neural net parameterizing the feedback

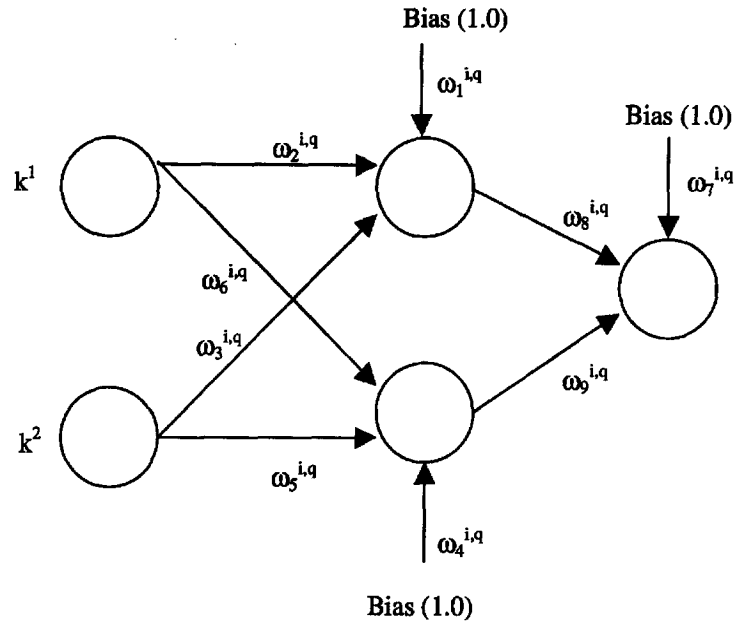

Figure 1. Neural network architecture for production policy rule.

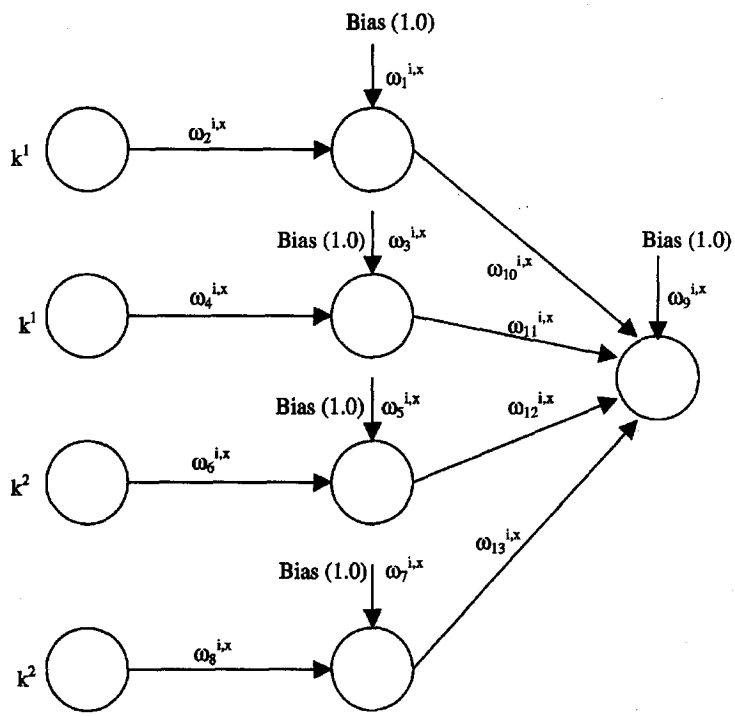

Figure 2. Neural network architecture for investment policy rule.

investment rule, $\varphi^i(\tau_t^0; \omega^{i,x})$. To obtain k_{t+1}^i (thus the input vectors at time $t+1$), we use equation (19) in the state transition equation (16) for each player i as

$$k_{t+1}^i = \varphi^i\left(\tau_t^0, \omega^{i,x}\right) + (1-\varepsilon)\, k_t^i, \tag{20}$$

where k_0^i is given. Now, substituting equations (18) and (19) into equation (17), the objective

function to be maximized subject to equation (20), $\phi^i \geq 0$ and $\varphi^i \geq 0$ becomes

$$\tilde{J}^i\left(\omega^{1,q},\omega^{2,q},\omega^{1,x},\omega^{2,x}\right)=\sum_{t=0}^{T-1}\delta^t\left[p_t^i\left(\phi^1\left(\nu_t^0,\omega^{1,q}\right),\phi^2\left(\nu_t^0,\omega^{2,q}\right)\right)\phi^i\left(\nu_t^0,\omega^{i,q}\right)\right.$$
$$\left.-C^i\left(k_t^i,\phi^i\left(\nu_t^0,\omega^{i,q}\right)\right)-h^1\left(\varphi^i\left(\tau_t^0,\omega^{i,x}\right)\right)\right]. \tag{21}$$

In any generation, the duopoly game is repeatedly played starting from the given set of initial states:

$$\Pi=\{(2,2),(2,6),(2,12),(6,2),(6,6),(6,12),(12,2),(12,6),(12,12)\}.$$

For any given set of weights and the initial states, $(k_0^1,k_0^2)\in\Pi$, the sample paths for the states, production and investment feedback policies are generated from the above maximization as $\{k_t^i\}_{t=0}^{T-1}$, $\{\phi^i(\nu_t^0,\omega^{i,q})\}_{t=0}^{T-1}$, and $\{\varphi^i(\tau_t^0,\omega^{i,x})\}_{t=0}^{T-1}$, respectively. Thus, GAs search the weights of production and investment networks to maximize the sum of the payoffs over all initial states

$$\hat{J}^i\left(\omega^{1,q},\omega^{2,q},\omega^{1,x},\omega^{2,x}\right)=\sum_{(k_0^1,k_0^2)\in\Pi}\sum_{t=0}^{T-1}\delta^t\left[p_t^i\left(\phi^1\left(\nu_t^0,\omega^{1,q}\right),\phi^2\left(\nu_t^0,\omega^{2,q}\right)\right)\phi^i\left(\nu_t^0,\omega^{i,q}\right)\right.$$
$$\left.-C^i\left(k_t^i,\phi^i\left(\nu_t^0,\omega^{i,q}\right)\right)-h^1\left(\varphi^i\left(\tau_t^0,\omega^{i,x}\right)\right)\right], \tag{22}$$

subject to equation (20), $\phi^i \geq 0$ and $\varphi^i \geq 0$.[6]

For each initial condition, the game is assumed to become stationary after 100 periods and the stationary tails are appended to the truncated objective functionals as in [2].

In every generation $s \in S$, $\mathrm{GA}^{i,j}$ $(i=1,2;\ j=q,x)$ operates on a constant size population, $N_{i,j}$, of neural net weights

$$\mathrm{Pop}^{i,j}(s)=\left\{\omega_1^{i,j}(s),\omega_2^{i,j}(s),\ldots,\omega_b^{i,j}(s),\ldots,\omega_{N_{i,j}}^{i,j}(s)\right\},$$

where $\omega_b^{i,j}(s)\in\mathrm{Pop}^{i,j}(s)$ is the weights of a potential neural network, that approximates the player i's optimal production policy for $j=q$ and optimal investment policy for $j=x$.

$\mathrm{GA}^{1,q}$ evaluates each individual $b\in\mathrm{Pop}^{1,q}(s)$ by computing its raw fitness

$$\hat{J}^1\left(\omega_b^{1,q}(s),\omega^{1,x*}(s-1),\omega^{2,q*}(s-1),\omega^{2,x*}(s-1)\right)$$
$$=\sum_{(k_0^1,k_0^2)\in\Pi}\sum_{t=0}^{T-1}\delta^t\left[p_t^1\left(\phi^1\left(\nu_t^0,\omega_b^{1,q}(s)\right),\phi^2\left(\tau_t^0,\omega^{2,q*}(s-1)\right)\right)\phi^1\left(\nu_t^0,\omega_b^{1,q}(s)\right)\right.$$
$$\left.-C^1\left(k_t^1,\phi^1\left(\nu_t^0,\omega_b^{1,q}(s)\right)\right)-h^1\left(\varphi^1\left(\nu_t^0,\omega^{1,x*}(s-1)\right)\right)\right],$$

where $\omega^{1,x*}(s-1)$, $\omega^{2,q*}(s-1)$, and $\omega^{2,x*}(s-1)$ are the previous best weights in the populations of $\mathrm{GA}^{1,x}$, $\mathrm{GA}^{2,q}$, and $\mathrm{GA}^{2,x}$, respectively.

Similarly, other GAs, $\mathrm{GA}^{1,x}$, $\mathrm{GA}^{2,q}$, and $\mathrm{GA}^{2,x}$, compute the raw fitnesses, $\hat{J}^1(\omega^{1,q*}(s-1),\omega_d^{1,x}(s),\omega^{2,q*}(s-1),\omega^{2,x*}(s-1))$ for all $d\in\mathrm{Pop}^{1,x}(s)$, $\hat{J}^2(\omega^{1,q*}(s-1),\omega^{1,x*}(s-1),\omega_e^{2,q}(s),\omega^{2,x*}(s-1))$ for all $e\in\mathrm{Pop}^{2,q}(s)$, and $\hat{J}^2(\omega^{1,q*}(s-1),\omega^{1,x*}(s-1),\omega^{2,q*}(s-1),\omega_f^{2,x}(s))$ for all $f\in\mathrm{Pop}^{2,x}(s)$, respectively.

The search starts with random populations, $\mathrm{Pop}^{1,q}(0)$, $\mathrm{Pop}^{1,x}(0)$, $\mathrm{Pop}^{2,q}(0)$, and $\mathrm{Pop}^{2,x}(0)$. In our simulations, the interconnection weights for investment and production neural nets are searched in the interval $[-5,5]$, with the exception of $\theta_{\min}$ and $\theta_{\max}$ for which the search intervals are $\theta_{\min}\in[0,3]$ and $\theta_{\max}\in[0,15]$.

[6]When a string representing the weights of a network violates the control bounds, it is punished by a high penalty; namely $-1000000*k_t^i*k_t^i$.

The genetic operators were done using the public domain Genesis 5.0 package [20] in parallel. We compile Genesis 5.0 on a SUN SPAC-1000 running Solaris 2.5. We run the experiment 50 times. In every run, we use *population sizes* of 50, *crossover rates* of 0.60, and *mutation rates* of 0.001 for each GA.

4.1. Results

The best weights for the policy functions over all runs are reported in Table 1. Figures 3–6 depict the best policy functions. Note that, even though Players 1 and 2 are symmetric and should have the same feedback policies, neural network specification and GA training result in slightly different policies. Thus, we report the best weights and graphs of the policy functions for both players.

Table 1. The best weights for the feedback policy functions.

Player 1				Player 2			
Production		Investment		Production		Investment	
$\omega_1^{1,q}$	−2.95	$\omega_1^{1,x}$	−1.61	$\omega_1^{2,q}$	−4.59	$\omega_1^{2,x}$	−0.12
$\omega_2^{1,q}$	3.07	$\omega_2^{1,x}$	−0.44	$\omega_2^{2,q}$	1.01	$\omega_2^{2,x}$	0.18
$\omega_3^{1,q}$	−1.07	$\omega_3^{1,x}$	−0.41	$\omega_3^{2,q}$	−0.23	$\omega_3^{2,x}$	2.00
$\omega_4^{1,q}$	4.30	$\omega_4^{1,x}$	0.33	$\omega_4^{2,q}$	−1.38	$\omega_4^{2,x}$	3.98
$\omega_5^{1,q}$	0.74	$\omega_5^{1,x}$	−1.20	$\omega_5^{2,q}$	0.88	$\omega_5^{2,x}$	0.14
$\omega_6^{1,q}$	−1.54	$\omega_6^{1,x}$	−0.17	$\omega_6^{2,q}$	−1.38	$\omega_6^{2,x}$	−0.17
$\omega_7^{1,q}$	−2.53	$\omega_7^{1,x}$	−3.57	$\omega_7^{2,q}$	1.42	$\omega_7^{2,x}$	−2.03
$\omega_8^{1,q}$	0.23	$\omega_8^{1,x}$	0.15	$\omega_8^{2,q}$	0.62	$\omega_8^{2,x}$	−0.56
$\omega_9^{1,q}$	0.29	$\omega_9^{1,x}$	2.53	$\omega_9^{2,q}$	0.48	$\omega_9^{2,x}$	4.90
$\theta_{\max}$	12.97	$\omega_{10}^{1,x}$	−0.90	$\theta_{\max}$	12.66	$\omega_{10}^{2,x}$	−1.61
$\theta_{\min}$	0.78	$\omega_{11}^{1,x}$	−2.21	$\theta_{\min}$	1.17	$\omega_{11}^{2,x}$	−3.54
		$\omega_{12}^{1,x}$	2.43			$\omega_{12}^{2,x}$	1.98
		$\omega_{13}^{1,x}$	−1.90			$\omega_{13}^{2,x}$	4.89

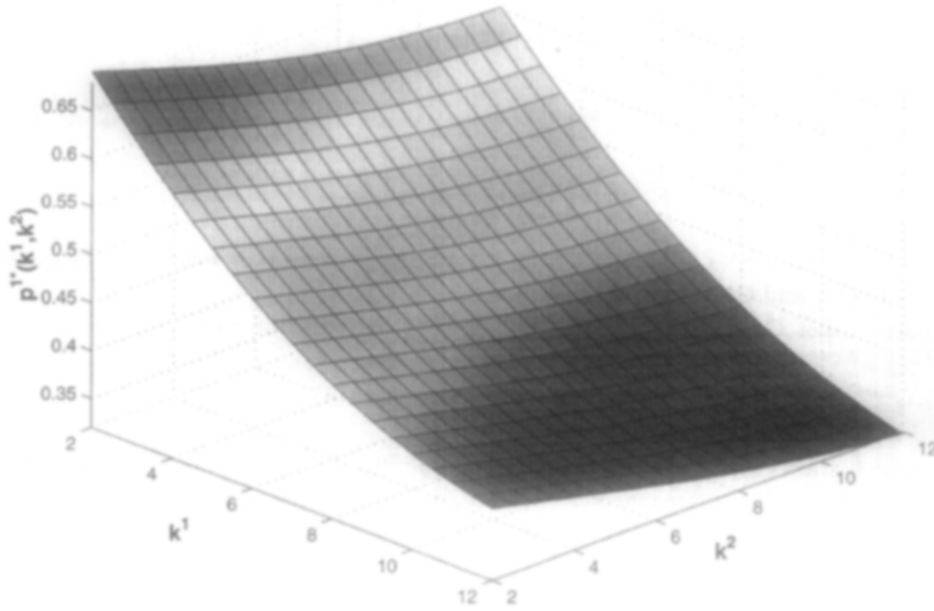

Figure 3. Player 1's equilibrium price, $p^{1*}(k^1, k^2)$.

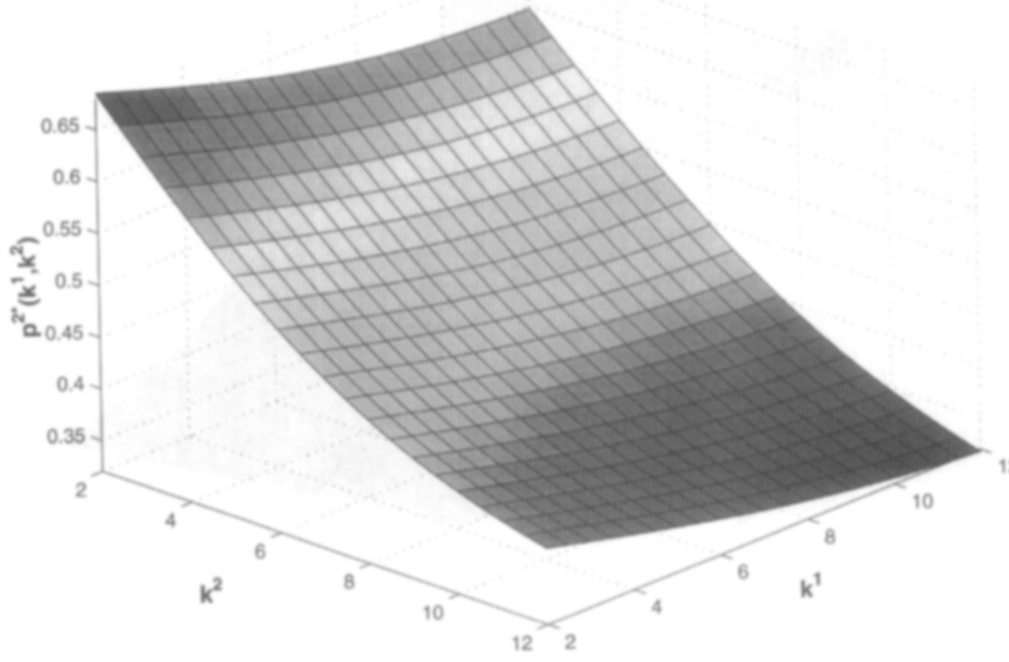

Figure 4. Player 2's equilibrium price, $p^{2*}(k^1, k^2)$.

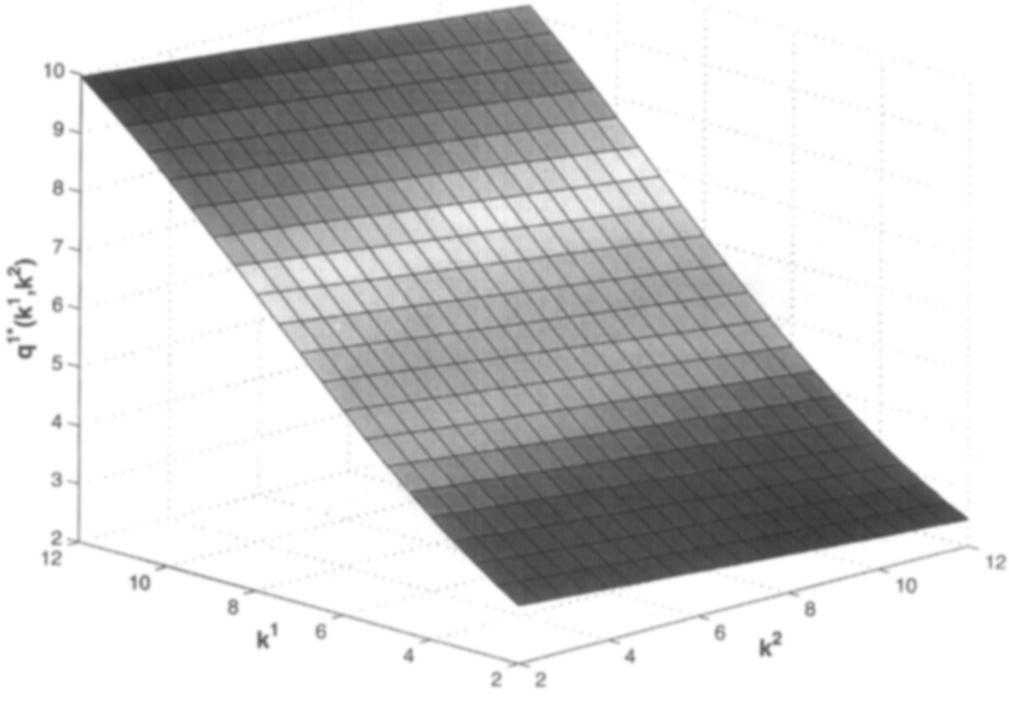

Figure 5. Player 1's equilibrium production levels, $q^{1*}(k^1, k^2)$.

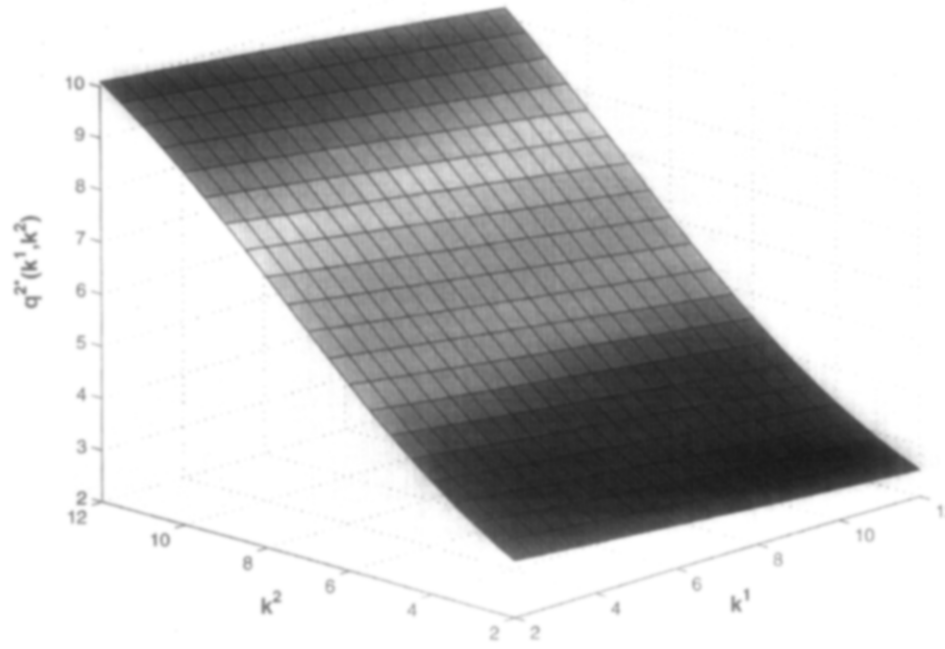

Figure 6. Player 2's equilibrium production levels, $q^{2*}(k^1, k^2)$.

Observe from Figures 3–6 that with an increase in players' own capital stocks, production costs diminish. Hence, equilibrium prices drop, while equilibrium quantities increase in player's own capital stock. An increase in the rival's capital stock, on the other hand, makes the rival more cost effective, and decreasing, therefore, both the equilibrium price and the equilibrium production levels of both players.

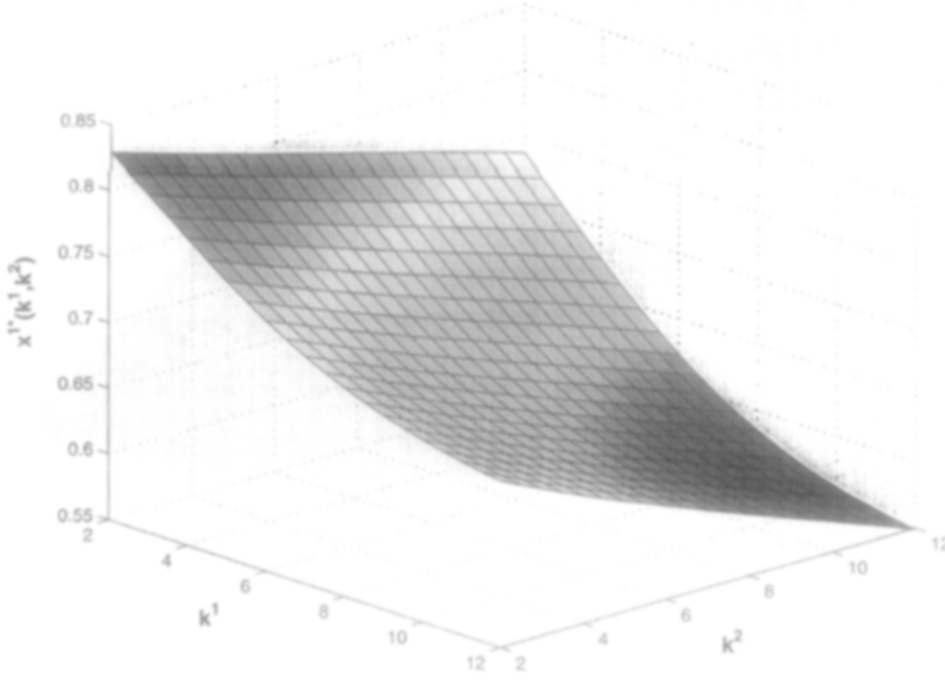

Figure 7. Optimal investment for Player 1, $x^{1*}(k^1, k^2)$.

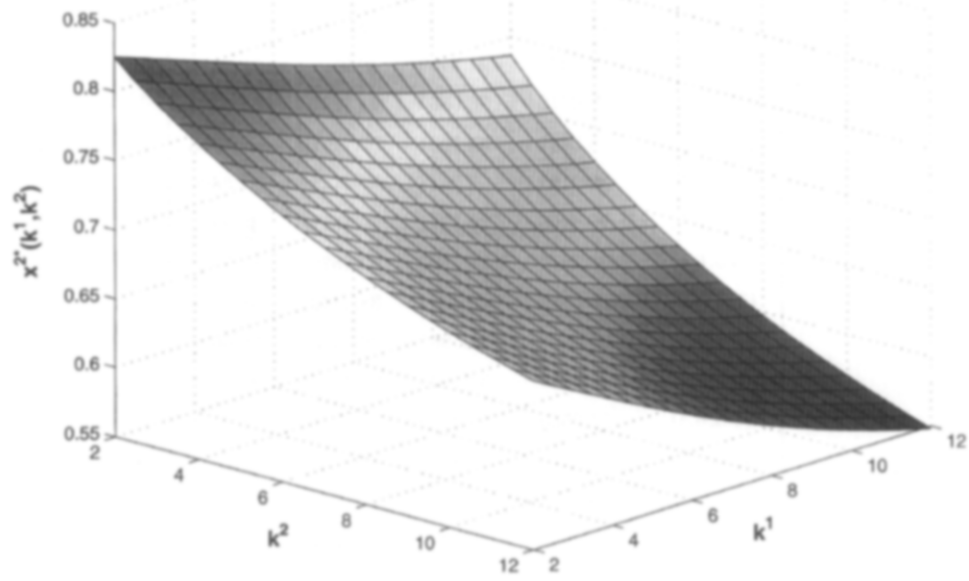

Figure 8. Optimal investment for Player 2, $x^{2*}(k^1, k^2)$.

Figures 7 and 8 show the optimal investment strategies as smooth decreasing functions of both capital stocks. When player's own capital stock is lower than the steady state, the optimal strategy is to invest as much as possible also paying heed to the fact that too fast an adjustment is costly. If, however, the initial capital stock is large relative to the steady state, it is optimal to allow the capital to depreciate down to the steady-state level. The steady state is determined by the conditions $x^i(k^1, k^2) = \varepsilon k^i$, $i = 1, 2$, that is, optimal investments are just sufficient to replace the worn out capital in the steady state.

4.2. Robustness

To test how good the trained networks generalize, the following initial states are chosen:

$$\tilde{\Pi} = \{(1.5, 1.5), (1.5, 3.5), (1.5, 6.5), (1.5, 12.5), (3.5, 1.5), (3.5, 3.5), (3.5, 6.5), (3.5, 12.5), (6.5, 1.5), (6.5, 3.5), (6.5, 6.5), (6.5, 12.5), (12.5, 1.5), (12.5, 3.5), (12.5, 6.5), (12.5, 12.5)\}.$$

Note that to measure the out of sample performance of the trained neural nets, the initial states in $\tilde{\Pi}$ are different from the ones in Π.

We measure the performance of our approximation by the following statistic:

$$\mathrm{Opt}^i = \frac{\sum\limits_{(k_0^1, k_0^2) \in \tilde{\Pi}} J^i\left(k_0^1, k_0^2\right) - \sum\limits_{(k_0^1, k_0^2) \in \tilde{\Pi}} J^{i*}\left(k_0^1, k_0^2\right)}{\sum\limits_{(k_0^1, k_0^2) \in \tilde{\Pi}} J^i\left(k_0^1, k_0^2\right)},$$

where J^{i*} is the payoff of player i using the stationary equilibrium policies using the weights in Table 1. J^is, on the other hand, are computed in from each pair initial states given in $\tilde{\Pi}$ for each player by retraining the neural nets 50 times. We report the best performance over all runs.

The values of J^i and J^{i*} are shown in Table 2. Note again that, as neural network specification and GA training result in slightly different policies even though the players are symmetric, we report Js and J^*s for both players. From Table 2, the calculated Opt^1 is 0.0029 and Opt^2 is 0.0017, which verifies that the trained networks are indeed robust.

Table 2. Out of sample performance.

(k_0^1, k_0^2)	J^{1*}	J^1	J^{2*}	J^2
(1.5, 1.5)	28.2976	28.2391	28.3655	29.1052
(1.5, 3.5)	27.3449	27.3456	30.9857	30.9451
(1.5, 6.5)	26.0716	26.0865	34.2395	34.1915
(1.5, 12.5)	24.3447	24.3481	39.7772	39.8923
(3.5, 1.5)	30.9162	30.7647	27.4118	27.3890
(3.5, 3.5)	29.8405	30.1272	29.9021	30.2212
(3.5, 6.5)	28.3816	28.3994	32.9879	32.7326
(3.5, 12.5)	26.3682	26.2482	38.2442	38.3447
(6.5, 1.5)	34.1958	34.6961	26.1653	25.8671
(6.5, 3.5)	32.9640	33.4026	28.4737	28.3119
(6.5, 6.5)	31.2560	31.2379	31.3178	31.4981
(6.5, 12.5)	28.8571	28.9319	36.1266	36.2030
(12.5, 1.5)	39.7452	39.8180	24.3984	24.3276
(12.5, 3.5)	39.2666	38.4976	26.4251	26.3665
(12.5, 6.5)	36.1114	36.5443	28.9063	28.8035
(12.5, 12.5)	32.9242	32.6501	32.9685	33.3310
Sum	495.8856	497.3373	496.6956	497.5303

5. CONCLUSION

We have developed a new method to efficiently approximate the feedback Nash equilibria in dynamic games. In the algorithm, players' feedback rules are represented by feedforward neural network architectures which are trained online synchronously by genetic algorithms in a parallel manner to search over all time-invariant equilibrium strategies.

Neural network specification offers important flexibility over the traditional techniques, such as spline functions or RBFs. There are several good reasons for our choice of genetic algorithms (GA) to train the neural networks as well. An important advantage in GA search is the fact that regardless of the initial parameter values, they ensure converge to an approximate global optimum by exploiting the domain space and relatively better solutions through genetic operators. Another feature of GAs which we exploit in our game algorithm is their explicit parallelism.

In the algorithm, each feedback rule is parameterized by the weights of a suitable neural network, thereby transforming the players' strategy spaces from sets of rules to sets of neural net weights. For each network, an artificially intelligent player, a GA, is assigned to breed fitter weights and update other GAs via the computer shared memory as to its best-to-date feedback rule. In any given generation, the game starts from multiple sets of initial states. As we search for the stationary feedback rules, the same set of neural net weights are employed to compute the overall payoffs as the raw fitness. Our rationale for repeating the game is two-fold: to avoid the dependence of the weight training on the initial conditions and also to speed up GA learning.

Essentially, each GA trains a neural network online to best respond to the other players' best feedback rule in the previous generation. The fittest weights are then communicated to each other via the computer shared memory. Equipped with the updated weights, each player is better able to anticipate the best response of the others for all potential rules in its population. The individuals which exploit this knowledge to their advantage are fitter, thus reproducing faster.

Ultimately, the fitter weights dominate the respective populations also steering the other players' search for the best set of rules to the vicinity of the feedback Nash equilibrium.

REFERENCES

1. M.J. Miranda and D.V. Vedenov, Numerical solution of dynamic oligopoly games with capital investment, *Economic Theory* **18**, 237–261, (2001).
2. N.M. Alemdar and S. Özyıldırım, A genetic game of trade, growth and externalities, *Journal of Economic Dynamics and Control* **22**, 811–832, (1998).
3. S. Clemhout and H.Y. Wan, Jr., A class of trilinear differential games, *Journal of Optimization Theory and Applications* **14**, 419–424, (1974).
4. E.J. Dockner, G. Feichtinger and S. Jørgensen, Tractable classes of nonzero-sum open-loop Nash differential games: Theory and examples, *Journal of Optimization Theory and Applications* **45**, 179–197, (1985).
5. C. Fershtman, Identification of classes of differential games for which the open-loop is a degenerated feedback Nash equilibrium, *Journal of Optimization Theory and Applications* **55**, 217–231, (1987).
6. A. Mehlmann and R. Willing, On nonunique closed-loop Nash equilibria for a class of differential games with a unique and degenerate feedback solution, *Journal of Optimization Theory and Applications* **41**, 463–472, (1983).
7. J. Reinganum, A class of differential games for which the closed loop and open loop Nash equilibria coincide, *Journal of Optimization Theory and Applications* **36**, 253–262, (1982).
8. T. Başar and G.J. Oldser, *Dynamic Noncooperative Game Theory*, Academic Press, New York, NY, (1982).
9. K. Funahashi, On the approximate realization of continuous mapping by neural networks, *Neural Networks* **2**, 183–192, (1988).
10. K. Hornik, M.H. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* **2**, 359–366, (1989).
11. K.S. Narenda and K. Parthasarthy, Identification and control of dynamical systems using neural networks, *IEEE Transaction on Neural Networks* **1** (1), 4–27, (1990).
12. R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Massachusetts, (1990).
13. J. Hertz, A. Krogh and A.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Massachusetts, (1991).
14. J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, (1975).
15. H. Mühlenbein, Darwin's continent cycle theory and its simulation by the prisoner's dilemma, In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, (Edited by F.J. Verela and P. Bourgine), pp. 236–244, (1992).
16. S. Özyıldırım, Three-country trade relations: A discrete dynamic game approach, *Computers Math. Applic.* **32** (5), 43–56, (1996).
17. N.M. Alemdar and S. Sirakaya, On-line computation of Stackelberg equilibria with synchronous parallel genetic algorithms, *Journal of Economic Dynamics and Control* **27**, 1503–1515, (2003).
18. J. Branke, Evolutionary algorithms for neural network design and training, Technical Report, No. 322, University of Karlsruhe, Institute AIFB, (1995).
19. K. Judd, Cournot vs. Bertrand: A dynamic resolution, Working paper, Stanford University, (1996).
20. J.J. Grefenstette, A user's guide to GENESIS version 5.0, Manuscript, (1990).
21. J. Mercenier and P. Michel, Discrete-time finite horizon approximation of infinite horizon optimization problems with steady-state invariance, *Econometrica* **62**, 635–656, (1994).