Abstract

This paper presents a neural network based methodology for examining the learning of game-playing rules in never-before seen games. A network is trained to pick Nash equilibria in a set of games and then released to play a larger set of new games. While faultlessly selecting Nash equilibria in never-before seen games is too complex a task for the network, Nash equilibria are chosen approximately 60\% of the times. Furthermore, despite training the network to select Nash equilibria, what emerges are endogenously obtained bounded-rational rules which are closer to payoff dominance, and the best response to payoff dominance.

Keywords: rationality, learning, neural networks, normal form games, complexity.
JEL Classification: C72, D00, D83.

Strategy Learning in 3×3 Games by Neural Networks

Daniel Sgroi [1]
Churchill College & Department of Applied Economics
University of Cambridge

Daniel J. Zizzo
Christ Church College & Department of Economics
University of Oxford

1. Introduction

Consider how you would face the following challenge. You consider yourself to be a fair player of a certain board game, and have to decide what odds you place on your chances of winning versus a beginner. How would you change your odds of winning if you were told (a) your opponent rarely plays any board games at all; (b) your opponent is pretty good at chess and a few other board games; (c) your opponent is a well known grand master chess player, and known to be a very strong player of many other board games? Any sensible player would consider even a complete beginner with the sort of record of player (c) to be a tough challenge, and (a) to be the easiest of all. Now consider two common ways of modeling this situation in game theory. We could simply assert that something close to optimal play will always occur, so in cases (a), (b) and (c) previous experience would be an irrelevant factor. Or we could follow a more evolutionary approach, and care only about how many times our opponent has played the particular board game before: once again, we would then be uninterested in experience with any other game. In both cases, we get the counterintuitive result that how an agent plays a game is entirely independent of how well he plays in other games.

[1]Please address queries regarding this paper to Daniel Sgroi, Department of Applied Economics, Sidgwick Avenue, Cambridge, CB3 9DE, UK or email daniel.sgroi@econ.cam.ac.uk. Daniel Sgroi is grateful to AEA Technology for financial support and both authors would like to thank Michael Bacharach, Vince Crawford, Huw Dixon, Glenn Ellison, Jim Engle-Warnick, Susan Hurley, Gerry Mackie, Meg Meyer, Matthew Mulford, Raimondello Orsini, Andrew Oswald, Michael Rasmussen, Antonio Santamaura, Andrew Temple and participants to presentations in Cambridge, Copenhagen and Oxford.

So how can we bridge the gap between what seems sensible and what the two most common approaches deem reasonable? Another way to approach this is to ask: can a player learn how to play any $n \times n$ game by playing a finite set of other $n \times n$ games? So will experience of chess help to play checkers? Will experience as a bank manager help to be a store manager? Will experience in an oligopoly competing mainly in prices assist an entrant to an oligopoly that competes mainly in quantities? In most cases we would argue in the positive, and it is with this in mind that this paper develops a tentative framework to examine the learning of general game-playing strategies during the play of specific games.

This is a difficult task, and the methods used in this paper represent a beginning rather than the final word. In particular, the transformed question we address is simpler: can a specific model of a decision-maker trained to play well in a small set of 3×3 games be expected to play well in a new set of 3×3 games which it has never faced before, against a population of players all of which will play in Nash equilibrium strategies? 3×3 games were chosen as the represent the simplest class of $n \times n$ games which allow us to consider iterated deletion of dominated strategies. The method of investigation will be a neural network trained to play Nash equilibria coupled with its empirical testing in a set of new games. The choice of a neural network is in part justified by the large body of related literature in cognitive science and engineering, and in part by the similarity between our results and those seen in experimental testing on human subjects (Costa-Gomes *et al* 2001; Stahl and Wilson 1994). As to the limitations of the paper, we focus on a single neural network player in a population of Nash players, not a population of neural networks. The later would represent a different set of considerations but would be equally interesting. Second we consider a neural network trained to select Nash equilibria, not trained to maximize utility, so we are assuming Nash equilibria to be the best way to play. This is justified in part below, but clearly our results are only applicable where Nash equilibria are generally accepted to be the correct solution method. In particular this paper can be said to address a subsidiary question: even with direct exposure to the concept of Nash equilibrium taught through example will a neural network player stick with this solution concept or independently develop a new technique for solving new games? What we discover is that Nash equilibrium is just too complex a concept for a neural network to use in general in new environments. It is not difficult for the network to find Nash equilibria in specific games, but what is difficult is to learn to employ Nash as a general algorithm (on the basis of learning by example).

1.1. **Overview.** The next section details the model to be used, in particular defining the neural network player and the games he is to face. Section 3 details some existing results

in neural network and algorithm complexity theory which provide us with clear theoretical predictions about what to expect our neural network player to be able to achieve. Section 4 details the results of the empirical testing of an actual neural network designed within the parameters of the theoretical network. Section 5 considers the situation with multiple equilibria and section 6 concludes.

2. The Model

Here we characterize the environment into which our neural network player will be placed: a well-defined population of Nash players repeatedly playing a series of 3×3 normal form games.

For the purposes of this paper, a *random game,* G, is defined as a 3×3 normal form game of full information with a unique pure strategy Nash equilibrium and randomly determined payoffs taken from a uniform distribution with support $[0, 1]$. More formally we can define the simple game by a list $G = \langle N, \{A_i, u_i\}_{i \in N} \rangle$. We will restrict the number of players, indexed by i, to $N = 2$. A_i describes player actions available to a player in role i, with realized actions given by $a_i \in A_i$. In each game we consider the set of feasible actions available to each player to be of size 3. Feasible action combinations are given by $A = A_1 \times A_2$. Payoffs for both players are given by $u_i : A_i \mapsto \mathbb{R}$ which is a standard von Neumann-Morgenstern utility function. Payoffs are bounded, so $\exists Q \geq 0$ such that $\mid u_i(a) \mid \leq Q$ for all a. More specifically we consider the payoffs to be randomly drawn from a uniform $(0, 1)$ and then revealed to the players before they select an action, so $\forall_{i,a}$, $\sup u_i(a_i) = 1$. We will place one further restriction on the game, by requiring the existence of a single unique pure strategy Nash equilibrium. To summarize:

Definition 1. *A random game is a list* $G = \langle N, \{A_i, u_i\}_{i \in N} \rangle$*, such that* $N = 2$ *players meet to play a game playing realized action* $a_i \in A_i$*, where three action choices are allowed. The values of the payoffs,* $u_i : A_i \mapsto \mathbb{R}$ *are randomly drawn from a uniform* $(0, 1)$*, made known to the players before the game starts, and form a standard von Neumann-Morgenstern bounded utility function.*

Now consider a population of Q players playing a series of random games, indexed by $t \in \mathbb{N}^{++}$. We consider a pair to be drawn from our population and then forced to play a given random game. Define the unique pure strategy Nash strategy to be $\alpha_i \in A_i$ where a Nash strategy is defined as the unique choice of pure strategy by player i which, when taken in combination with the Nash strategy chosen by the other member of the drawn pair, form the unique Nash equilibrium in the random game. So defining the specific Nash strategy for

player i in a given random game to be α_i, and indexing the second player by j we have:

$$u_i\left(a_i = \alpha_i \mid a_j = \alpha_j\right) > u_i\left(a_i \neq \alpha \mid a_j = \alpha_j\right) \tag{2.1}$$

We can say immediately that player i, taken from the population of size Q, wishing to maximize u_i, must play the unique Nash strategy when drawn to play in G, and therefore the outcome will be the unique Nash equilibrium in G. Now define an *evolutionary stable strategy*:

Definition 2. *Let x and y be two mixed strategies from the set of mixed strategies in G. Now let $u(x, y)$ define the utility associated with the play of strategy x given the play of strategy y by the other player. The strategy x is said to be an evolutionary stable strategy (ESS) if $\forall_y\ \exists \varepsilon_y > 0$ s.t. when $0 < \varepsilon < \varepsilon_y$:*

$$u\left(x, (1-\varepsilon)\, x + \varepsilon y\right) > u\left(y, (1-\varepsilon)\, x + \varepsilon y\right) \tag{2.2}$$

Given the definition of G it is clear that:

Proposition 1. *The unique Nash strategy of G is an evolutionary stable strategy (ESS).*

Proof. This is straightforward from the definition of G. See Appendix A. ∎

We have a simple notion of a population of players playing Nash against each other and performing well in terms of their payoffs. We now consider the mutation of a proportion γ of the population into *neural network players*. By proposition 1 we know the remaining population will continue to play Nash strategies, if we let $\gamma \to 0$. We can retain this property by letting $Q \to \infty$ and setting γ to be fixed at a number strictly above zero, but finite. In particular, we can consider a single member of the population to be a neural network player, but let $Q \to \infty$ to bring γ arbitrarily close to zero. We can now examine the actions of this single neural network player content in the knowledge that the neural network's best reply to the population will be to play a Nash strategy. Future work could usefully generalize this further to consider a population of neural network players playing against each other. For now however, we examine the player upon which we shall focus for the rest of the paper and the learning technique we shall use: a neural network.

2.1. **A Brief Primer on Neural Networks.** Neural networks can be loosely defined as *artificial intelligence models inspired by analogy with the brain and realizable in computer programs.* They typically learn by exposure to a series of examples (a training set), and adjustment of the strengths of the connections between its nodes. They are then able to do well not only on the original training set, but also when facing problems never encountered

before. Consider a neural network, or more simply C to be a machine capable of taking on a number of states, each representing some computable functions mapping from input space to output space, with two *hidden* layers of further computation between input and output. Hidden layers can be thought of as intermediate layers of computation between input and output. Since we see the input go in, and the output come out, but do not directly see the activity of intermediate layers, they are in some sense *hidden*. The following definition formalizes what we mean by a neural network.

Definition 3. *Define the neural network as* $C = \langle \Omega, X, Y, F \rangle$ *where* Ω *is a set of states,* $X \subseteq \mathbb{R}^n$ *is a set of inputs,* Y *is a set of outputs and* $F : \Omega \times X \mapsto Y$ *is a parameterized function. For any* ω *the function represented by state* ω *is* $h_\omega : X \mapsto Y$ *given by* $h_\omega(x) = F(\omega, x)$ *for an input* $x \in X$*. The set of functions computable by* C *is* $\{h_\omega : \omega \in \Omega\}$*, and this is denoted by* H_C*.*

Put simply, when the network, C, is in state ω it computes the function h_ω providing it is computable. In order to reasonably produce answers which correspond to a notion of correctness (in this case the unique Nash strategy in a 3×3 game), we need to *train* the network. First we will consider the form of the network in practice, and start by defining an activation function.

Definition 4. *An activation function for node* i *of layer* k *in the neural network* C *is of the logistic form*

$$a_i^k = \frac{1}{1 - \exp\left(-\sum_j w_{ij}^k u_{ij}^{k-1}\right)} \tag{2.3}$$

where u_{ij}^k *is the output of node* j *in layer* $k-1$ *sent to node* i *in layer* k*, and* w_{ij} *is the weight attached to this by node* i *in layer* k*. The total activation flowing into node* i*,* $\sum_j w_{ij}^k u_{ij}^{k-1}$*, can be simply defined as* t_i*.*

Consider a set of 18 input nodes each recording and producing as output a different value from the vector $x_k = (x_k^1, ..., x_k^{18})$. This neatly corresponds to the payoffs of a 3×3 game. Now consider a second set of 36 nodes (the first hidden layer). Each node in this second layer receives as an input the sum of the output of all 18 input nodes transformed by the activation function of node i in layer 2. All of the nodes in the second layer send this output a_i^2 to all nodes in the second hidden layer, which weights the inputs from all i of the first hidden layer, by the activation function to produce a_i^3. These numbers are sent to the final layer of two nodes to produce an output y which forms a 2-dimensional vector which represents the choice of strategy in a 3×3 game. To explain this representation of a strategy in a

3×3 game for the row player, the vector $(1, 0)$ would imply that the neural network player's choice is the pure strategy embodied by selecting the top row, $(0, 1)$ would imply the middle row, and $(0, 0)$ the bottom row. Of course there is nothing restricting the neural network from choosing values other than 0 or 1, so it might select $(0.8, 0.2)$ which would suggest that it is certain it does not wish to pick the bottom row strategy, and is fairly happy to pick the top row strategy, but still has some doubts about whether it is better than middle. Should the Nash equilibrium be $(1, 0)$ we would aim to train the network to get close to $(1, 0)$ in a sense to be defined below. The network's outputs will be interpretable as a probability vector only as long as their sum adds up to 1, so for example $(0.9, 0.5)$ is ruled out.

Training essentially revolves around finding the set of weights that is most likely to reproduce the actual Nash equilibrium of the game faced. During training C receives a sequence of random games until some stopping rule determines the end of the training at some round T (discussed below). The *training sample* consists of M random games. If $T > M$, then (some or all of) the random games in M will be presented more than once. The training sample vector $x_k = (x_k^1, ..., x_k^{18})$ consists of 18 real-valued numbers drawn from a uniform $(0, 1)$ representing the payoffs of a 3×3 game. The only restriction to this being that the numbers must ensure a unique pure strategy Nash equilibrium. If x_k fails to achieve this a new vector is selected and so on until this condition is satisfied. It is recorded in the first set of input nodes, and then sent and transformed by the two hidden nodes before an output y, a two-dimensional vector, is produced and represented in the final layer. This is repeated M times with a new set of inputs x_k and outputs y_k. Assume that each vector x_k is chosen independently according to a fixed probability distribution P_T on the set X subject to the requirement that the game has a unique Nash equilibrium. The probability distribution is fixed for a given learning problem, but it is unknown to C, and for our purposes will be taken to be a uniform $(0, 1)$. The information presented to C during training therefore consists only of several sequences of numbers.

Definition 5. *For some positive integer m, the network is given a training sample:*

$$x^M = \left(\left(x_1^1, ..., x_1^{18}\right), \left(x_2^1, ..., x_2^{18}\right), ..., \left(x_M^1, ..., x_M^{18}\right)\right) = (x_1, x_2, ..., x_M) \in X^M$$

The labelled examples x_i are drawn independently according to the probability distribution P_T subject to the condition that each vector x_t ensures the existence a unique Nash equilibrium in pure strategies. If this condition fails a new vector is drawn from P_T. A random training sample of length M is an element of X^M distributed according to the product probability distribution P^M.

Assume that $T > M$. In this case, training might be *sequential*: after $q \times M$ rounds (for any positive integer q s.t. $q \times M < T$), M is presented again, exactly in the same order of games. If training is *random without replacement*, it is less restricted to the extent that the order in which the random games are presented each time is itself random. If training is *random with replacement*, in each round the network is assigned randomly one of the random games in M, until round T. Having selected a sample sequence of inputs, x, and determined the unique Nash strategy associated with each, α, we need to consider how C *learns* the relationship between the two, to ensure that its output y will approach the Nash strategy. The method used is backpropagation. First let us define the error function and then go on to define backpropagation.

Definition 6. *Define the network's root mean square error ε as the root mean square difference between the output y and the correct answer α over the full set of $q \times M$ games where individual games are indexed by i, so our error function is:*

$$\varepsilon \equiv \left(\sum_{i=1}^{q \times M} (y_i - \alpha_i)^2 \right)^{\frac{1}{2}}$$

The aim is to minimize the error function by altering the set of weights w_{ij} of the connections between a typical node j (the sender) and node i (the receiver) in different layers. These weights can be adjusted to raise or lower the importance attached to certain inputs in the activation function of a particular node. The correct answer here is the vector associated with the unique Nash equilibrium in pure strategies. In principle we could use any other measure, including for example training the neural network to select the best or even worst outcome in terms of game payoff. This paper's focus however is narrowly limited to a study of a neural networks ability to learn to play Nash equilibria. Backpropagation is a form of numerical analysis akin to gradient descent search in the space of possible weights. Following Rumelhart *et al* (1986) we use a function of the following form.

$$\triangle w_{ij} = -\eta \frac{\partial \varepsilon}{\partial w_{ij}} = \eta k_{ip} o_{jp} \tag{2.4}$$

To define the terms: w_{ij} is the weight of the connection between the sending node j and receiving node i. As ε is the neural network's error, $\partial\varepsilon/\partial w_{ij}$ measures th

sensitivity of the neural network's error to the changes in the weight between i and j. There is also a *learning rate* given by $\eta \in (0, 1]$: this is a parameter of the learning algorithm and must not be chosen to be too small or learning will be particularly vulnerable to local

error minima. Too high a value of η may also be problematic as the network may not be able to settle on any stable configuration of weights. Define $\partial\varepsilon/\partial w_{ij} = -k_{ip}o_{jp}$ where o_{jp} is the degree of activation of the sender node j for a given input pattern p. The higher is o_{jp}, the more the sending node is at fault for the erroneous output, so it is this node we wish to correct more. k_{ip} is the error on unit i for a given input pattern p, multiplied by the derivative of the output node's activation function given its input. Calling g_{ip} the goal activation level of node i for a given input pattern p, in the case of the output nodes k_{ip} can be computed as:

$$k_{ip} = (g_{ip} - o_{ip})f'(t_{ip}) = (g_{ip} - o_{ip})o_{ip}(1 - o_{ip}) \tag{2.5}$$

since the first derivative $f'(t_{ip})$ of the receiving node i in response to the input pattern p is equal to $o_{ip}(1 - o_{ip})$ for a logistic activation function. Now assume that a network has N layers, for $N \geq 2$. As above, we call layer 1 the input layer, 2 the layer which layer 1 activates (the first hidden layer), and so on, until layer N the output layer which layer $N-1$ activates.

We can now define the backpropagation learning process.

Definition 7. *Using backpropagation, we first compute the error of the output layer (layer N) using eq. 2.5, and update the weights of the connections between layer N and $N-1$, using eq. 2.4. We then compute the error to be assigned to each node of layer $N-1$ as a function of the sum of the errors of the nodes of layer N that it activates. Calling i the hidden node, p the current pattern and β an index for each node of layer N (activated by i), we can use:*

$$k_{ip} = f'(t_{ip}) \sum_{\beta} k_{\beta p} w_{\beta i} \tag{2.6}$$

to update the weights between layer $N-1$ and $N-2$, together with eq. 2.4. We follow this procedure backwards iteratively, one layer at a time, until we get to layer 1, the input layer. A variation on standard backpropagation would involve replacing eq. 2.4 with a momentum function of the form:

$$\triangle w_{ij}^t = -\eta \frac{\partial \varepsilon^t}{\partial w_{ij}^t} + \mu \triangle w_{ij}^{t-1} \tag{2.7}$$

where $\mu \in [0, 1)$ and $t \in \mathbb{N}^{++}$ denotes the time index (an example game, vector x, is presented in each t during training).

Momentum makes connection changes smoother by introducing positive autocorrelation in the adjustment of connection weights in consecutive periods. The connection weights of

the network are updated using backpropagation until round T. T itself can be determined exogenously by the builder of the neural network, or it can be determined endogenously by the training process, i.e. training may stop when the network returns the correct output with ε lower than a given target value. Backpropagation is a complex though much used tool in neural network analysis and much is written elsewhere about its advantages and disadvantages (see for example, White 1992).

We conclude the section with a non-technical summary. A neural network is continuously shown and reshown a set of games until it recognizes the Nash equilibria of these games. Recognition comes in terms of a simple measure of mean squared error away from the vector representation of the pure Nash equilibrium. When it has achieved this or close enough to it (when it knows the best way to play in this set of games) it is shown some different games and asked to find the Nash equilibria for these without ever having seen these new games before. It can however use the algorithms (rules) it has already learned in order to allow it to choose correctly the Nash equilibria in those games which it has seen before (the training set). Referring back to the introduction, this is akin to a player becoming good at chess and several other board games, and then being asked to face a new set of games. What we now need to do is examine the existing literature to determine whether the player will in fact correctly select the Nash equilibria or whether the rules it has developed, while useful for the subset of games already faced, are not in fact a general set of rules which will allow it to always select Nash equilibria. So will the player who has mastered chess, checkers and several other board games become a good player of any board game?

3. Learnability Results

Many results exist in the algorithm complexity and computer science literature which have relevance for our neural network player. In particular a full set of results exist which stress the difficulty of the learning problem. This section highlights existing results which effectively answer the questions posed in the last section. One of the most well-known results comes from Hornik *et al* (1989) reprinted in White (1992). Examining this result we see that standard *feedforward* networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well in the p_μ *metric*, which is defined as follows:

Definition 8. *Let $\mathbb{R}$ be the set of real numbers, and $\mathbb{B}^r$ the Borel σ-field of $\mathbb{R}^r$. Let K^r be the set of all Borel measurable functions from $\mathbb{R}^r$ to $\mathbb{R}$. Given a probability measure μ on*

$(\mathbb{R}^r, \mathbb{B}^r)$ *define the metric* p_μ *from* $K^r \times K^r$ *to* $\mathbb{R}^+$ *by*

$$p_\mu(f,g) = \inf\{\varepsilon > 0 : \mu\{x :\mid f(x) - g(x) \mid > \varepsilon\} < \varepsilon\}$$

The main result, Theorem 2.4 from Hornik *et al* (1989) effectively concerns the *existence* of a set of weights which allow the perfect emulation of the algorithm that the neural network is attempting to learn. There is however a major area of potential worry. The network may experience inadequate learning, so the learning dynamic will fail to reach the global error-minimizing algorithm. It is this which will be the concern of much of this section. A *learning algorithm* takes random training samples and acts on these to produce a function $h \in H$ that, provided the sample is large enough is with probability at least $1-\delta$, ε-good (with ε defined as above) for P_T. It can do this for each choice of ε, δ and P_T. To define this more formally:

Definition 9. *Suppose that* H *is a class of functions that map* $X \mapsto Y$. *A learning algorithm* L *for* H *is a function* $L : \cup_{M=1}^{\infty} Z^M \mapsto H$ *from the set of all training samples to* H, *with the following property: for any* $\forall_{p\in(0,1),\delta\in(0,1)}\exists$ *an integer* $M_0(\varepsilon,\delta)$ *s.t. if* $M \geq M_0(\varepsilon,\delta)$ *then, for any probability distribution on* $Z = X \times Y$, *if* z *is a training sample of length* M *drawn randomly according to the product distribution* P_T, *then, with probability at least* $1-\delta$, *the hypothesis* $L(z)$ *output by* L *is such that* $er_p(L(z)) < opt_p(H) + \varepsilon$. *More compactly, for* $M \geq M_0(\varepsilon,\delta)$, $P^M\{er_p(L(z)) < opt_p(H) + \varepsilon\} \geq 1-\delta$.

To restate in slightly different terms we can define a function L as a learning algorithm, if $\exists$ a function $\varepsilon_0(M,\delta)$ *s.t.* $\forall_{M,\delta,P_T}$, with probability at least $1-\delta$ over $z \in Z^M$ chosen according to P^M, $er_p(L(z)) < opt_p(H) + \varepsilon_0(M,\delta)$, and $\forall_{\delta\in(0,1)}$, $\varepsilon_0(M,\delta) \to 0$ as $M \to \infty$. This definition stresses the role of $\varepsilon_0(M,\delta)$ which we can usefully think of as an estimation error bound for the algorithm L. Closely related is the definition of learnability:

Definition 10. *A learning algorithm* L *as defined in definition 9 takes random training samples and acts on these to produce a hypothesis* $h \in H$. *We say that the class of functions* H *is learnable if* $\exists$ *a learning algorithm* L *for* H.

So now we return full circle to our original definition of a neural network given in definition 3 and we see how crucial is the computability of our function h_ω (h for a given state ω). The final function h_ω produced by the neural network can be thought of as representing the entire processing of the neural network's multiple layers, taking an input vector x and producing a vector representation of a choice of strategy. Over a long enough time period we would hope that C will return a set of optimal weights which will in turn produce a function which

will select the Nash strategy if $\exists$ a learning algorithm for selecting Nash equilibria (H in this case). Or alternatively if we wish to attain some below perfect success rate, we can do so using a finite training sample, and the success rate will grow as the number of examples increases. This all crucially rests on the ability of backpropagation to pick out the globally error-minimizing algorithm for finding Nash equilibria. This now allows us to tightly define the learning problem faced by C.

Definition 11. *C, using the learning algorithm given by definition 7 faces a training sample of size $M \times q$. The Nash problem is to find an algorithm as defined in definition 9 for which $\varepsilon_0(M, \delta) \to 0$ as $M \to \infty$ where the error function ε is as defined in definition 6.*

Having established the problem we now need to verify that the algorithm which successfully collapses $\varepsilon_0(M, \delta)$ to zero is indeed learnable. While backpropagation is undoubtedly one of the most popular search algorithms currently used to train feedforward neural networks, it is a gradient descent algorithm and therefore as shown in Sontag and Sussmann (1995) this approach leads only to a local minimum of the error function. White (1992, page 160) succinctly makes the point: "...Hoornik *et al* (1989) have demonstrated that sufficiently complex multilayer feedforward networks are capable of arbitrarily accurate approximations to arbitrary mappings... An unresolved issue is that of "learnability", that is whether there exist methods allowing the network weights corresponding to these approximations to be learned from empirical observation of such mappings". He then provides a theorem which summarizes the difficulties inherent in backpropagation: he proves that backpropagation can get stuck at local minima or saddle points, can diverge, and cannot even be guaranteed to get close to a global minimum (Theorem 3.1 in White, 1992).

The problem is exacerbated in the case of our neural network C as the space of possible weights is so large. Furthermore, the number of local minima for this class of networks can be exponentially large in the number of network parameters (Auer *et al*, 1996). The upper bound for the number of such local minima is calculable, but it is unfortunately not tight enough to lessen the problem (Sontag, 1995). In fact, as the probability of finding the absolute minimizing algorithm (the Nash algorithm) is likely to be exponentially small, the learning problem faced by C falls into a class of problems known in algorithm complexity theory as NP-hard. There exists a literature on the complexity of computing an automaton to play best response strategies in repeated games, see for example Ben-Porath (1990) or Gilboa (1988), but in all cases the problem examined is one of computing best response algorithms in a repeated game context, not when facing never before played one-shot games. Nevertheless the literature establishes the precedent of using algorithm complexity to help determine

the reasonableness of certain strategies (automata) employed by bounded players, and in many cases establishes the hardness of computing Nash strategies. Within the computer science literature there are recent results which are directly applicable to the problem we examine. Amending slightly an existing theorem widely known in the literature (see for example Anthony and Bartlett (1999), Theorem 25.5) we have:

Theorem 1. *Problems of the type given in definition 11 faced by the class of networks encompassing C are NP-hard.*

There exist several forms of this theorem for different types of network including the feedforward class of which C is a member. The following proposition is simply a restatement of theorem 1 with reference to the particular problem faced by C.

Proposition 2. *C supplemented by the backpropagation learning dynamic will not be able to learn the Nash algorithm in polynomial time.*

Gradient descent algorithms attempt to search for the minimum of an error function, and backpropagation is no exception. However, given the prevalence of local minima, they cannot consistently solve the problem in definition 9 and find an absolute minimum. The basins of attraction surrounding a local minimum are simply too strong for a simple gradient descent algorithm to escape, so looking back to definition 9 we cannot expect $\varepsilon_0(M, \delta) \to 0$ as $M \to \infty$, and in turn we cannot consider the task facing the network to be *learnable* in the sense of definition 10. Two further results in the literature are worth noting. Local minima will always exist in problems of this type (Fukumiza and Amari, 2000), and the number of local minima for this class of networks is exponentially large in the number of network parameters (Auer *et al*, 1996). In terms of the theory of NP-completeness, the solution can be found in exponential time, but not in polynomial time. For any network with a non-trivial number of parameters, such as C, the difference is great enough for the term intractable to be applied to such problems. Backpropagation cannot consistently find an algorithm capable of providing Nash equilibria in never before seen games.

Other far less biologically plausible methods involving processor hungry guess and verify techniques, can produce consistent results. If we were to supplement the algorithm with a guessing stage, i.e. add something akin to grid search, or a subtle application of the theory of sieves, then we could hope to find the absolute minimum in polynomial time (White, 1992). However, White (1992), page 161 tells us that such methods "... lay no claim to biological or cognitive plausibility", and are therefore not desirable additions for the modelling of decision-making.

Returning to backpropagation only, our player will find a decision-making algorithm that will retain some error even at the limit, so we may have to be content with an algorithm which is effective in only a subclass of games, optimizing network parameters only in a small subspace of the total space of parameters. In the case of normal form games we can summarize what can be extracted from the existing literature for our particular problem as: *our player will almost surely not learn the globally error-minimizing algorithm for selecting Nash equilibria in normal form games.* However, we can reasonably assume that some method will be learned, and this should at least minimize error in some subset of games corresponding to the domain of some local error-minimizing algorithm.

3.1. **Local Error-Minimizing Algorithms.** Given that backpropagation will find a local minimum, but will not readily find an absolute minimizing algorithm in polynomial time, we are left with the question, what is the best our neural network player can hope to achieve? If we believe the neural network with a large, but finite training set nicely models bounded-rational economic agents, but cannot flawlessly select Nash strategies with no prior experience of the exact game to be considered, this question becomes: what is the best a bounded-rational agent can hope to achieve when faced with a population of fully rational agents?

In terms of players in a game, we have what looks like bounded-rational learning or satisficing behavior: the player will learn until satisfied that he will choose a Nash equilibrium strategy sufficiently many times to ensure a high payoff. We label the outcome of this bounded-rational learning as a local error-minimizing algorithm (LMA). More formally, consider the learning algorithm L, and the 'gap' between perfect and actual learning, $\varepsilon_0(M,\delta)$. Recall that Z^M defines the space of possible games as perceived by the neural network.

Definition 12. *If $\exists$ a function $\varepsilon_0(M,\delta)$ s.t. $\forall_{M,\delta,P_T}$, with probability at least $1-\delta$ over all $z \in Z^M$ chosen according to P^M, $er_p(L(z)) < opt_p(H) + \varepsilon_0(M,\delta)$, and $\forall_{\delta\in(0,1)}$, $\varepsilon_0(M,\delta) \to 0$ as $M \to \infty$ then this function is defined the global error-minimizing algorithm (GMA).*

This simply states that for all possible games faced by the network, after sufficient training, the function will get arbitrarily close to the Nash algorithm, collapsing the difference to zero. This clearly requires an algorithm sufficiently close to Nash to pick a Nash equilibrium strategy in almost all games.

Definition 13. *A local error-minimizing algorithm (LMA) will select the same outcome as a global error-minimizing algorithm for some $z \in Z^M$, but will fail to do so for all $z \in Z^M$.*

LMAs can be interpreted as examples of rules of thumb that a bounded-rational agent is likely to employ in the spirit of Simon (1955) or Simon (1959). They differ from traditionally conceived rules of thumb in two ways. First, they do select the best choice in some subset of games likely to be faced by the learner. Second, they are learned *endogenously* by the learner in an attempt to maximize the probability of selecting the best outcome. The 'best' outcome can be determined in terms of utility maximization or a reference point, such as the Nash equilibrium.

4. Testing the Network

In practical terms we can construct a simulated network and test to see whether this network matches our theoretical predictions. The training set is simply a sequence of vector pairs (x, y) where the inputs $x \in X$ correspond to the set of actions A_i for N players in M random games, and the outputs to the payoffs $u_i : A \mapsto \mathbb{R}$ for N players for each of the actions. We set $M = 2000$, $N = 2$ and restrict the action set by assuming a 3×3 normal form game. This restriction is done without loss of generality: potentially any finite normal form could be modelled in a similar way, while 2×2, 2×3 and 3×2 games could be modeled by forcing rows or columns of zero. We then allow the network to play 2000 further random games never encountered before, selecting a single input and recording a single output. Since we force each game to contain a unique Nash equilibrium in pure strategies and we restrict the network's choice to be in pure strategies, we can then check the network's *success rate* as defined by the proportion of times the network selected the Nash strategy to within a given threshold of mean squared error (as defined in definition 6). For example if the correct output is $(1, 0)$ and the neural network returns $(0.99, 0)$ it easily meets an $\varepsilon = 0.05$ threshold.

The training set consisted of $M = 2000$ games with unique pure Nash equilibria. Training was random with replacement (subject to the unique Nash equilibrium condition), and continued until the error ε converged below 0.1, 0.05 and 0.02, i.e. three *convergence levels* γ were used: more than one convergence level was used for the sake of performance comparison. Convergence was checked every 100 games, a number large enough to minimize the chance of a too early end of the training: clearly, even an untrained or poorly trained network will get an occasional game right, purely by chance. The computer determined initial connection weights and order of presentation of the games according to some 'random seed' given at the start of the training. To check the robustness of the analysis, C was trained 360 times, that is once for every combination of 3 learning rates η (0.1, 0.3, 0.5), 4 momentum rates μ (0, 0.3, 0.6 and 0.9) and 30 (randomly generated) random seeds. Convergence was always obtained, at least at the 0.1 level, except for a very high momentum rate. Details on convergence can

be found in Appendix B. We will henceforth call the simulated network C^* once trained to a given convergence level.

C^* was tested on a set of 2000 games with unique Nash equilibria never encountered before. We restricted ourselves to training with games with unique pure strategy Nash equilibria because of the technical need for backpropagation to be able to work with a unique solution. This appeared a lesser evil relative to having to postulate refinements to Nash in order to discriminate among multiple equilibria, and hence make the analysis dependent on these refinements. Still, the next section goes some of the way to examining the role of multiple equilibria. We considered an output value to be correct when it is within some range from the exact correct value. If both outputs are within the admissible range, then the answer can be considered correct, see Reilly (1995). The ranges considered were 0.05, 0.25 and 0.5, in decreasing order of precision.

[TABLE I]

Table 1 displays the average performance of C^* classified by γ, η and μ. It shows that C^* trained until $\gamma = 0.1$ played exactly (i.e., within the 0.05 range) the Nash equilibria of 60.03% of the testing set games, e.g. of 2000 3×3 games never encountered before. This fits well with the 59.6% average success rate of human subjects newly facing 3×3 games in Stahl and Wilson (1994), although one has to acknowledge that the sample of games they used was far from random. With an error tolerance of 0.25 and 0.5, the correct answers increased to 73.47 and 80%, respectively.

Further training improves its performance on exactness - the 0.02-converged C^* plays exactly the Nash equilibria of a mean 66.66% of the games - but *not* on "rough correctness" (the 20% result appears robust). This suggests (and indeed further training of the network confirms) that there is an upper bound on the performance of the network.

Table 1 also shows that, once C converges, the degree it makes optimal choices is not affected by the combination of parameters used: the average variability in performance across different learning rates is always less than 1%, and less than 2% across different momentum rates. This is an important sign of robustness of the analysis.

We compared C^*'s performance with three null hypotheses of zero rationality. Null1 is the performance of the entirely untrained C: it checks whether any substantial bias towards finding the right solution was hardwired in the network. Null2 alternates among the three pure strategies: if C^*'s performance is comparable to Null2, it means all it has learned is to be decisive on its choice among the three. Null3 entails a uniformly distributed random

choice between 0 and 1 for each output: as such, it is a proxy for zero rationality. Table 2 compares the average performance of C^* with that of the three nulls. Formal t tests for the equality of means between the values of C^* and of each of the nulls (including Null2) are always significant (P<0.0005). C^*'s partial learning success is underscored by the fact, apparent from Tables 1 and 2, that when C^* correctly activates an output node it is very likely to categorize the other one correctly, while this is not the case for the nulls.

[TABLE II]

4.1. **Is the Neural Network using Alternatives to Nash?** It appears that C^* has learned to generalize from the examples and to play Nash strategies at a success rate that is significantly above chance. Since it is also significantly below 100%, the next question we must address is how to characterize the LMA achieved by the trained network. Our first strategy is to ask ourselves whether there are simple alternatives to Nash capable of describing what the network does better than Nash, on the games over which they are uniquely defined. Given the robustness of our analysis in the previous section to different combinations of η and μ, in this and the next sections we just focus on the case with $\eta = 0.5$ and $\mu = 0$. The robustness of the results with different parameter combinations ensures that *this particular choice* is not really relevant. In any event, it was driven by two considerations: 1. any momentum greater than 0 has hardly any real psychological justification, at least in this context; 2. given $\mu = 0$, a learning rate of 0.5 had systematically produced the quickest convergence.

For testing we used the 30 networks trained with the 30 different random seeds but with the same learning (0.5) and momentum (0) rates. Using these 30 networks, we tested the average performance of the various algorithms on the same testing set of 2000 new games with unique pure Nash equilibria considered in the previous section.

We consider the following algorithms or alternative solution methods in turn: 1) minmax; 2) rationalizability; 3) '0-level strict dominance' (0SD); 4) '1-level strict dominance' (1SD); 5) 'pure sum of payoff dominance' (L1); 6) 'best response to pure sum of payoff dominance' (L2); 7) 'maximum payoff dominance' (MPD); 8) 'nearest neighbor' (NNG). 1 and 2 are well known, and 3 and 4 are simply levels of reasoning in the rationalizability process (rationalizability equating to '2-level strict dominance' in a 3×3 game). 5, 6, 7 and 8 need a little more explanation and Appendix C contains some definitions and further information about these algorithms. Intuitively, MPD corresponds to going for the highest conceivable payoff for itself; L1 to choosing the best action against a uniformly randomizing opponent;

L2 to choosing the best action against a L1 player. Finally, a NNG player responds to new situations by comparing them to the nearest example encountered in the past, and behaves accordingly.

We define a game as *answerable* by an algorithm if a unique solution exists. Table 3 lists the number and percentage of answerable games (out of 2000) according to each algorithm, averaged out across the 30 neural networks trained with different random seeds, $\eta = 0.5$ and $\mu = 0$.

[TABLE III]

Table 3 also lists the percentage of games where the unique solution coincides with the pure Nash equilibrium of the game. In order to determine how an agent following a non-Nash algorithm would behave when faced with the testing set, we need to make an auxiliary assumption with regards to how the agent would be playing in non-answerable games. We assume that, in non-answerable games, the agent randomizes over all the actions (two or three, according to the game) admissible according to the non-Nash algorithm (e.g., in the case of rationalizability, all the non-dominated actions): if the admissible actions are two or three and one of them is the Nash equilibrium choice, the agent will get it right 1/2 or 1/3 of the times on average, respectively. The right column of Table 3 adjusts accordingly the expected success rate of the non-Nash algorithm in predicting Nash. What we get is the degree to which the various algorithms are good or bad LMAs.

Some findings emerge. Our set of candidate LMAs typically can do better than how a zero rational agent simply playing randomly across all choices and games would do. More strategically sophisticated LMAs can do better than less strategically sophisticated ones. Rationalizability, 0SD and 1SD are limited in their ability in predicting Nash by the limited number of corresponding answerable games. The most successful algorithms in predicting Nash are first L2, then rationalizability and finally L1. L2 and L1 combine, in different proportions, low strategic sophistication with considerable Nash predictive success in our set of 3×3 games. They have also been found as the best predictors in normal-form games of behavior by experimental subjects in Costa-Gomes *et al* (2001). L2 is particularly impressive in predicting Nash in our set of 3×3 games.

On the basis of these considerations, we hypothesize that the LMA played by C^* may be describable to a significant degree by L2 and also possibly L1, among the non-Nash algorithms we have considered. In our interpretation, though, we do not rule out the possibility that C^* does more than simply following any of the non-Nash algorithms of Table 3. Still, if true, it

would match the predictive success of L2 and L1 in experimental data in Costa-Gomes *et al* (2002). This matching would be the more striking because they did not include 3×3 games in their experiment.

Table 4 shows how well the various algorithms can describe C^*'s behavior on the testing set. We consider both the success rate as a percentage of the answerable games or of the full testing set, and an adjusted success rate to allow once again for random play over admissible strategies in the non-answerable games.

[TABLE IV]

NNG fares considerably worse than Nash on the data: indeed, it does worse in predicting C^*'s behavior than it does in predicting Nash (see Table 3). We should not be surprised by the fact that the NNG still gets about half of the games right according to the 0.02 convergence level criterion: it is quite likely that similar games will often have the same Nash equilibrium. Partial nearest neighbor effects cannot be excluded in principle on the basis of Table 4. However, the failure of the NNG algorithm relative to Nash suggests that - at least with a training set as large as the one used in the simulations ($M = 2000$) - the network does not reason simply working on the basis of past examples.

Rationalizability, 0SD and 1SD outperform Nash for the games they can solve in a unique way. 0SD, 1SD and rationalizability exactly predict C^*'s behavior in 80.98%, 76.25% and 74.36% of their answerable games, respectively: this is 8-14% above Nash. The fact that C^* still gets three quarters of all rationalizable games exactly right suggests that it is able to do some strategic thinking. However, the network can still play reasonably well in games that are not answerable according to 0SD, 1SD and rationalizability: hence, Nash still outperforms over the full testing set.

L2 is the best algorithm in describing C^*'s behavior, with a performance comparable to rationalizability, 0SD and 1SD for answerable games but, unlike those, with virtually all the games answerable. It predicts C^*'s behavior exactly 76.44% over the full testing set. L1 performs worse than L2, but its performance still matches rationalizability over the full testing set. The fact that L2 outperforms L1, while being more strategically sophisticated, confirms that C^* behaves as if capable of some strategic thinking.

5. Multiple Equilibria

In this section we still consider the network trained purely on games with unique equilibria, but ask what its behavior will be when faced not just with new games, but with *a new class*

of games: games with multiple Nash equilibria. If C^* has learned to categorize games according to some game features, we would expect the network to apply a similar set of tools as much as possible, when faced with games with multiple equilibria. This, of course, may be impossible if C^*'s LMA were inapplicable to this context. For example, if C^* just followed iterated deletion of strictly dominated strategies, then the network should be unable to choose among the plurality of equilibria as these each represent rationalizable solutions.

We begin with a hypothesis: the network will be displaying focal points. We interpret this in the present context as meaning that *different* networks should tend to converge to the same equilibria in games with multiple equilibria. Why this should be the case is different according to whether we view networks as working mainly by assimilating new games to already encountered instances, or otherwise. In the first case, different networks, trained under different random seeds but with the same training set and parameters, will tend to choose the action corresponding to the solution of the nearest neighbor to the game with multiple equilibria: hence, there will be a focal solution. However, one might wonder whether the differences induced by the usage of different random seeds are really so important: perhaps, we are dealing with basically the same neural networks in each case, and so finding focal points might be considered uninteresting as a model of what might be focal in the real world. We shall talk in this case about focal points in a weak sense, or *w-focal points*.

If the neural network instead learns mainly through exploiting general game features (as the analysis so far has suggested), we would expect focal points even if the network has been trained with different training sets, as long as they are drawn from the same distribution. This is because the latter condition is sufficient to ensure that the different neural networks will extract about the same game features. We shall talk in this case about focal points in a stronger sense, or *s-focal points*.

We considered two sets of neural networks. The first set of thirty networks (*Set 1*) is the standard group considered in the previous sections, trained with the same training set, $\eta = 0.5$, $\mu = 0$, but with different random seeds. The second set of thirty networks (*Set 2*) was trained again with $\eta = 0.5$ and $\mu = 0$, but varying not only the random seed but also the training set in each case; training was random without replacement. Thirty training sets of $M = 2000$ games each drawn from a uniform distribution [0,1] were used. On the basis of the results from our previous sections, we hypothesize that the network will display not only w-focal points but also s-focal points. If the network retains some sensitivity to examples, however, we might expect the percentage of neural networks converging to w-focal points to be slightly higher than the one converging to s-focal points. On the basis of the experimental

results in Costa-Gomes *et al* (2001) we also conjecture that C^* will lean towards choices that correspond to the L2 and L1 solutions.

The testing set was made of 2000 games again, namely 100 games with three equilibria and 1900 games with two equilibria. Let us call a choice "decided" if both outputs are within 0.25 of a pure strategy value. Let us then consider the number of decided choices (between 0 and 30) corresponding to each action $(1, 2, 3)$, for each game. We can formulate two null hypotheses for the absence of focal points in terms of the distribution of decided choices across actions. According to the first null hypothesis, the player would simply choose randomly which action to take, i.e. the player would be entirely naive in facing games with multiple equilibria: in this case, we would expect the number of decided choices to be the same across actions, and we shall take them as equal to the average number of decided choices. According to the second null hypothesis, the agent would be able to detect the pure Nash equilibrium, but would only be able to choose among them randomly. On average, in this case we would expect the same number of decided choices for each pure Nash equilibrium. Clearly, this second null hypothesis can be meaningfully distinguished from the first only in the case of games with two, rather than three, equilibria.

For the three equilibria dataset ($n = 100$) under both nulls, $\chi^2 = 2531.256, 198$ $d.f.$, for Set 1, and $\chi^2 = 1853.324, 198$ $d.f.$, for Set 2. For the 2 equilibria dataset ($n = 1900$) under the first null, $\chi^2 = 67653.74, 3798$ $d.f.$, for Set 1, and $\chi^2 = 56174.93, 3798$ $d.f.$. For the two equilibria dataset under the second null, $\chi^2 = 30785.17, 1898$ $d.f.$, for Set 1, and $\chi^2 = 23985.49, 1899$ $d.f.$ for Set 2. One game had to be removed in relation to the test on Set 1 because the corresponding expected values were zero for the second null. In all cases, using χ^2 tests the null hypotheses are strongly rejected (at $p < 0.001$) for both Set 1 and Set 2: Zizzo and Sgroi (2000) employ regression analysis to show that various game features applicable for play in games with unique Nash equilibrium extend to play in games with multiple equilibria.

Hence, the network is displaying not only w-focal points but also s-focal points. Interestingly, the χ^2 is lower with Set 2 than with Set 1 in a comparable sample of two equilibria or three equilibria games, suggesting that the importance of focal points is somewhat lower with s-focal points, as it might be expected by a limited nearest neighbor effect. Nevertheless, the strong evidence for s-focal points suggests that different neural networks, trained on the same game distribution although on different games, must be displaying focal points because they have learned to detect the same game features and so they tend to choose the same solution. A criticism of this conclusion might be that, although we have shown that choices tend to be focal on specific outcomes, we have not shown that the number of

decided choices is high in the first place in games with multiple equilibria. However, the number of decided choices only drops from an average of 8.97 per game action in the unique pure Nash equilibrium dataset to 8.52 with Set 1 and 8.49 with Set 2: taking into account that, if all choices were "decided", the value should be equal to 10, it is apparent that the neural network is quite decided in general in its choices, and is only slightly more indecisive in games with multiple equilibria.

6. Conclusions

This paper presented a neural network model designed to capture the endogenous emergence of bounded-rational behavior in normal-form games. Potentially any finite normal-form could be modelled in this way, though we have concentrated on 3×3 games, the smallest class of $n \times n$ games that can be subjected to iterated deletion. A neural network player, having seen a sufficiently large sample of example games in which the Nash outcome was highlighted, *could* potentially learn the Nash algorithm. However, this is highly unlikely because of the complexity of the Nash problem: effectively, the Nash algorithm is intractable by a network that uses learning algorithms, such as backpropagation, with some biological and cognitive plausibility. Hence, the network is much more likely to find some simpler way to solve the problem, that allows it to get sufficiently close in a large enough number of cases to leave the network satisfied that it has found a suitable way of playing new games. This local error-minimizing algorithm would allow the network to achieve a 'satisficing' level of success in finding a Nash equilibrium in a never-before-seen game, though it would not achieve 100% success. It would correspond to one or more behavioral heuristics endogenously learned by the bounded-rational agent.

The simulation results suggest a figure of around 60% success on games never encountered before, as compared with the 33% random success benchmark or the 59.6% experimental figure from Stahl and Wilson (1994). Such simulations also indicate that solution concepts other than Nash get closer to explaining the simulated network's actual behavior: pure sum of payoff dominance and the best response to this strategy. These strategies, under the respective names of L1 and L2, are those most observed in the laboratory with normal-form games in the study by Costa-Gomes *et al* (2001). This correspondence is the more interesting because Costa-Gomes *et al* use game matrices of different dimensionality from 3×3 (namely, 2×2, 2×3, 3×2, 4×2, and 2×4): this suggests that our reliance on 3×3 games is not seriously restrictive in practice. Further, in our data L2 performs better than L1, possibly because it is a considerably more successful theoretical tool in predicting Nash, while being

computationally only moderately more demanding. The network displays focal points in games with multiple Nash equilibria.

A neural network cannot learn to pick out Nash equilibria faultlessly in a series of new games, even when it is capable of doing so in a finite subset of games. So a grand master chess player may be a tough chess opponent and will be a strong player when facing many new games, but there will equally be many times when he will play new games, make many errors, and face defeat. In the process of trying to learn to always pick out Nash equilibria, a neural network will stumble onto an alternative simpler set of rules which may look at first sight like some form of simple dominance. Indeed perhaps the most interesting finding in this paper is that a simulated neural network will behave so much like the experimental subjects in Costa-Gomes *et al* (2002) through the use of what seems like payoff dominance. However, much like real people, as our neural network diverges from rational (Nash) behavior it becomes increasingly difficult to tie down its underlying motivation: nevertheless, progress can be made. We suggest that, as we more understand the methods used by our neural network, so we may better hope to understand the errors made by game-players in the laboratory and in the real world.

References

Anthony, M. and Bartlett, P. L. (1999), *Neural Network Learning: Theoretical Foundations*, Cambridge University Press, Cambridge.

Auer, P., Herbster, M. and Warmuth, M. K. (1996), 'Exponentially many local minima for single neurons', in *Advances in Neural Information Processing Systems 8* (D. S. Turetzky, M. C. Mozer, M. E. Hasslemo, Eds.), pp 316–322, MIT Press, Cambridge, Mass. and London.

Ben-Porath, E. (1990), 'The complexity of computing a best response automaton in repeated games with mixed strategies', *Games & Economic Behavior,* 2, pp 1–12.

Costa-Gomes, M., Crawford, V. P. and Broseta, B. (2001), 'Cognition and behavior in normal-form games: an experimental study', *Econometrica,* 69, pp 1193-1235.

Fukumizu, K. and S. Amari (2000), 'Local minima and plateaus in hierarchical structures of multilayer perceptrons', *Neural Networks,* 13, pp 317–327.

Gilboa, I. (1988), 'The complexity of computing best response automata in repeated games', *Journal of Economic Theory*, 45, pp 342–352.

Hornik, K., Stinchombe, M. B. and H. White, H. (1989), 'Multi-layer feedforward networks are universal approximators', *Neural Networks*, 2, pp 359–366.

Reilly, R. (1995), 'A connectionist exploration of the computational implications of embodiment', in *Proceedings of the Swedish Conference on Connectionism*, Lawrence Erlbaum, Hillsdale, pp 237–257.

Rumelhart, D. E., Hinton, R. J. and Williams, R. J. (1986), 'Learning representations by back-propagating error', *Nature*, 323, pp 533–536.

Simon, H. (1955), 'A behavioral model of rational choice', *Quarterly Journal of Economics*, 69, pp 99–118.

Simon, H. (1959), 'Theories of decision-making in economics and behavioral science', *American Economic Review*, 49, pp 253–283.

Sontag, E. D. (1995), 'Critical points for least-squares problems involving certain analytic functions with applications to sigmoidal nets', *Advances in Computational Mathematics*, 5, pp 245–268.

Sontag, E. D. and Sussmann, H. J. (1989), 'Backpropagation can give rise to spurious local minima even for networks with no hidden layers', *Complex Systems*, 3, pp 91–106.

Stahl, D. O. and Wilson, P. W. (1994), 'Experimental evidence on players' models of other players', *Journal of Economic Behavior & Organization*, 25, pp 309-327.

Stahl, D. O. and Wilson, P. W. (1995), 'On players' models of other players: theory and experimental evidence', *Games & Economic Behavior*, 10, pp 218-254.

White, H. (1992), *Artificial Neural Networks: Approximation and Learning Theory*, Blackwell, Cambridge and Oxford.

Zizzo, D. J. and Sgroi, D. (2000), 'Bounded-rational behavior by neural networks in normal form games', D.P. 2000-W30. Nuffield College, University of Oxford.

Appendix

Appendix A

The proof is simply a restatement of the well-known result that local superiority implies evolutionary stability. Firstly, G has a unique Nash strategy by definition. Call this strategy α_i for player i. Now we know that $u(\alpha_i \mid a_j = \alpha_j) > u(a_i \mid a_i \neq \alpha_i, a_j = \alpha_j)$ so by the uniqueness of α_i we know that any mix of α_i with $a_i \neq \alpha_i$ will reduce $u(\alpha_i, a_j)$ where $u(a_i, a_j)$ is the payoff to player i from action $a_i \in A$ given player j plays $a_j \in A$. Therefore the Nash equilibrium of G must be strict. By strictness we know that $u(\alpha_i, a_j) > u(\beta_i, a_j)$ where $\beta_i \neq \alpha_i$. This in turn implies local superiority, so:

$$\lim_{\varepsilon \to 0} \left\{ (1-\varepsilon)\, u(\alpha_i, \alpha_j) + \varepsilon u\left(\alpha_i, \beta_j\right) \right\} > \lim_{\varepsilon \to 0} \left\{ (1-\varepsilon)\, u(\beta_i, \alpha_i) + \varepsilon u\left(\alpha_i, \beta_j\right) \right\}$$

By linearity of expected utility in probabilities this implies:

$$u\left(\alpha_i, (1-\varepsilon)\,\alpha_j + \varepsilon\beta_j\right) > u\left(\beta_i, (1-\varepsilon)\,\alpha_j + \varepsilon\beta_j\right) \text{ for } \varepsilon \to 0$$

Which is simply a restatement of the definition of an ESS given in definition 2.

Appendix B

The following table contains details on convergence of C trained with different random seeds and achieving various levels of convergence for different learning and momentum rates.

[TABLE V]

Only with a combination of high η and μ ($\eta = 0.5$ and $\mu = 0.9$) convergence was not achieved, even at $\gamma = 0.1$. A high momentum rate was particularly detrimental. A low learning rate ($\eta = 0.1$) slowed learning, particularly with $\mu \leq 0.3$, but did not prevent C from converging at $\gamma = 0.1$ or better.

A maximum of 600000 rounds was used to try to achieve convergence. In practice, though, $\gamma = 0.1$ convergence was usually reached in about 100000-200000 rounds.

Appendix C

This appendix contains a number of definitions and explanations of the algorithms or solution concepts used in the main text.

Consider the game G and the trained neural network player C^*. Index the neural network by i and the other player by j. The neural network's minmax value (or reservation utility) is defined as:

$$r_i = \min_{a_j}\left[\max_{a_i} u_i\left(a_i, a_j\right)\right]$$

The payoff r_i is literally the lowest payoff player j can hold the network to by any choice of $a \in A$, provided that the network correctly foresees a_j and plays a best response to it. Minmax therefore requires a particular brand of pessimism to have been developed during the network's training on Nash equilibria.

Rationalizability is widely considered a weaker solution concept compared to Nash equilibrium, in the sense that every Nash equilibrium is rationalizable, though every rationalizable equilibrium need not be a Nash equilibrium. Rationalizable sets and the set which survives the iterated deletion of strictly dominated strategies are equivalent in two player games: call this set S_i^n for player i after n stages of deletion. To give a simple intuitive definition, S_i^n is

the set of player i's strategies that are not strictly dominated when players $j \neq i$ are constrained to play strategies in S_j^{n-1}. So the network will delete strictly dominated strategies and will assume other players will do the same, and this may reduce the available set of strategies to be less than the total set of actions for i,resulting in a subset $S_i^n \subseteq A_i$. Since we are dealing with only three possible strategies in our game G, the subset can be adequately described as $S_i^2 \subseteq A_i$ with player j restricted to $S_j^1 \subseteq A_i$.

The algorithm 0SD checks whether all payoffs for the neural network (the row player) from playing an action are strictly higher than those of the other actions, so no restriction is applied to the action of player $j \neq i$, and player i's actions are chosen from $S_i^0 \subseteq A_i$. 1SD allows a single level of iteration in the deletion of strictly dominated strategies: the row player thinks that the column player follows 0SD, so chooses from $S_j^0 \subseteq A_j$, and player i's action set is restricted to $S_i^1 \subseteq A_i$. Both of these algorithms can be viewed as weakened, less computationally demanding versions of iterated deletion.

L1 and MPD are different ways of formalizing the idea that the agent might try to go for the largest payoffs, virtually independently of strategic considerations. If following MPD, C^* simply learns to spot the highest conceivable payoff for itself, and picks the corresponding row, hoping the other player will pick the corresponding column. In the case of L1, an action $a_i = a_{L1}$ is chosen by the row player according to:

$$a_{L1} = \arg\max_{a_i \in A_i} \{a_i \mid a_j = \triangle_j\}$$

Where $\triangle_j$ is defined as a perfect mix over all available strategies in A_j. Put simply, a_{L1} is the strategy which picks a row by calculating the payoff from each row, based on the assumption that player j will randomly select each column with probability $\frac{1}{3}$, and then chooses the row with the highest payoff calculated in this way. It corresponds to the definition of 'level 1 thinking' in Stahl and Wilson (1995).

A player that assumes that his opponent is a level 1 thinker will best-respond to L1 play, and hence engage in what, following Stahl and Wilson (1995) and Costa-Gomes *et al* (2002), we can label L2 (for 'level 2') play.

Finally, the NNG is an algorithm that is based on the idea that agents respond to new situations by comparing them to the nearest example encountered in the past, and behaving accordingly. The NNG algorithm examines each new game from the set of all possible games, and attempts to find the specific game from the training set (which proxies for memory) with the highest level of "similarity". We compute similarity by summing the square differences between each payoff value of the new game and each corresponding payoff value of each game of the training set. The higher the sum of squares, the lower the similarity between the new

game and each game in the training set. The NNG algorithm looks for the game with the lowest sum of squares, the *nearest neighbor*, and chooses the unique pure NE corresponding to this game.

Since, by construction, all games in the training set have a unique pure NE, we are virtually guaranteed to find a NNG solution for all games in the testing set. The only potential (but unlikely) exception is if the nearest neighbor is not unique, because of two (or more) games having exactly the same sum of squares. The exception never held with our game sample. Clearly, a unique solution, or indeed any solution, may not exist with other algorithms, such as rationalizability, 0SD and 1SD. A unique solution may occasionally not exist with other algorithms, such as MPD, because of their reliance on strict relationships between payoff values.

TABLE 1. Percentage of Correct Answers

Convergence Level γ	At Least 1 Correct Output Error tolerance criterion			Correct Answer Given Error tolerance criterion		
	0.05	0.25	0.5	0.05	0.25	0.5
0.1	85.12	91.76	94.31	60.03	73.47	80
0.05	87.26	92.24	94.31	64.12	74.75	80.09
0.02	88.52	92.51	94.25	66.66	75.47	79.96

Learning Rate η	At Least 1 Correct Output Error tolerance criterion			Correct Answer Given Error tolerance criterion		
	0.05	0.25	0.5	0.05	0.25	0.5
0.1	86.88	92.3	94.42	63	74.48	80.22
0.3	86.75	92.06	94.25	63.3	74.42	79.89
0.5	86.81	92.04	94.2	63.66	74.54	79.94

Momentum Rate μ	At Least 1 Correct Output Error tolerance criterion			Correct Answer Given Error tolerance criterion		
	0.05	0.25	0.5	0.05	0.25	0.5
0	86.87	92.36	94.5	62.86	74.63	80.47
0.3	86.77	92.27	94.42	62.89	74.49	80.22
0.6	86.91	92.09	94.23	63.73	74.53	79.9
0.9	86.6	91.52	93.8	64.05	74.05	79.04

Notes: Percentage of games solved by the network under different combinations of η, γ and μ. The level of convergence γ simply measures how correct we ask the network to be: the smaller it is, the stricter the criterion. The learning rate η is a coefficient that determines the speed of the adjustment of the connection weights when the network fails to play the Nash equilibrium behavior. A positive momentum rate μ introduces autocorrelation in the adjustments of the connection weights when successive examples are presented. The error tolerance criterion measures how close the answer given by the network must be to the exact answer in order to consider the answer right. The smaller the error tolerance criterion, the tighter it is. The numbers given under 'At least 1 Correct Output' are the % of cases in which at least 1 of the two output nodes is correct. The numbers given under 'Correct Answer Given' are the % of cases in which both output nodes are correct.

TABLE 2. Average Performance of the Trained Network versus Three Null Hypotheses

	At Least 1 Correct Output Error tolerance criterion			Correct Answer Given Error tolerance criterion		
	0.05	**0.25**	**0.5**	**0.05**	**0.25**	**0.5**
Trained *C**	**86.82**	**92.13**	**94.29**	**63.31**	**74.48**	**80.02**
Null1 (Untrained *C*)	**0**	**0.005**	**67**	**0**	**0**	**0**
Null2 (Strategy Switcher)	**78.85**	**78.85**	**78.85**	**22.8**	**22.8**	**22.8**
Null3 (Random)	**0.091**	**43.5**	**75.1**	**0.003**	**0.06**	**25.5**

Notes: Average performance of the trained *C** versus three null hypotheses. The smaller the error tolerance criterion, the tighter the criterion used to consider *C**'s strategy choice correct. The numbers given under 'At least 1 Correct Output' are the % of cases in which at least 1 of the two output nodes is correct. The numbers given under 'Correct Answer Given' are the % of cases in which both output nodes are correct.

TABLE 3. Answerable Games and Relationship to Nash

Non Nash Algorithm	Answerable Games (%)	Uniquely predicted Nash equilibria (%)	Expected performance in predicting Nash (%)
0 Level Strict Dominance	18.3	18.3	46.97
1 Level Strict Dominance	39.75	39.75	62.19
Rationalizability	59.35	59.35	74.78
L1	100	67	67
L2	99.75	88.4	88.48
Maximum Payoff Dominance	99.6	62.1	62.23
Minmax	99.75	58.55	58.63
Nearest Neighbor	100	62.8	62.8

Notes: Answerable games are games for which the Nash algorithm provides one, and exactly one, solution. The percentage is equal to (Number of Answerable Games)/2000. The central column considers the cases where these unique solutions coincide with the pure Nash equilibrium of the game. The right column adjusts this Nash predictive success of the non Nash algorithm by making an auxiliary assumption on the agent's play in games where the non Nash algorithm does not provide a unique solution: namely, we assume that the agent randomizes over all the choices (two or three, according to the game) admissible according to the non Nash algorithm (e.g., in the case of rationalizability, all the non dominated solutions).

TABLE 4. Describability of *C's Behavior by Non Nash Algorithms**

Algorithm	% of Correct Answers Over Answerable Games			% of Correct Answers Over Full Testing Set			Expected Performance Over Full Testing Set		
	γ=0.5	γ=0.25	γ=0.05	γ=0.5	γ=0.25	γ=0.05	γ=0.5	γ=0.25	γ=0.05
Nash	80.38	75.7	66.53	80.38	75.7	66.53	80.38	75.7	66.53
L2	85.11	82.23	76.54	84.9	82.03	76.35	84.98	82.11	76.44
L1	67.71	63.48	55.97	67.71	63.48	55.97	67.71	63.48	55.97
Rationalizability	86.44	82.57	74.36	51.3	49.01	44.13	64.16	61.94	57.18
Nearest Neighbor	62.97	58.78	51.48	62.97	58.78	51.48	62.97	58.78	51.48
MPD	61.08	57.02	49.95	60.84	56.79	49.75	60.84	56.79	49.76
Minmax	57.75	53.83	46.89	57.6	53.7	46.77	57.67	53.77	46.85
1SD	87.63	84	76.25	34.83	33.39	30.31	54.51	53.15	50.19
0SD	90.83	87.79	80.98	16.62	16.06	14.82	43.91	43.35	42.12

Notes: % of correct answers over answerable games = (number of correct answers) / (number of answerable games). Correct answers here implies giving the same answer as C*. Answerable games are games for which the algorithm identifies a unique solution. % of correct answers over full testing set = (number of correct answers) / (number of answerable games). Expected performance over full testing set: % of correct answers over full testing set + adjustment due to the assumption of randomization over admissible actions in non answerable games.

TABLE 5. Number of *C* Converged

Convergence Level $\gamma = 0.1$		Momentum Rate μ			
		0	0.3	0.6	0.9
Learning Rate η	0.1	30	30	30	30
	0.3	30	30	30	25
	0.5	30	30	30	0

Convergence Level $\gamma = 0.05$		Momentum Rate μ			
		0	0.3	0.6	0.9
Learning Rate η	0.1	27	30	30	30
	0.3	30	30	30	11
	0.5	30	30	30	0

Convergence Level $\gamma = 0.02$		Momentum Rate μ			
		0	0.3	0.6	0.9
Learning Rate η	0.1	0	1	30	30
	0.3	30	30	30	0
	0.5	30	30	30	0