BOUNDED-RATIONAL BEHAVIOR BY NEURAL NETWORKS IN NORMAL FORM GAMES

DANIEL J. ZIZZO AND DANIEL SGROI

Abstract. This paper presents a neural network model developed to simulate the endogenous emergence of bounded-rational behavior in normal form games with unique Nash equilibria. There exists an algorithm which, if learnt by a neural network, would enable it to perfectly select Nash equilibria in never before seen games. However, finding this algorithm is too complex a task for a biologically plausible neural network, and as such it will instead settle for converging to a *local error-minimizing algorithm*, i.e. an approximation to Nash in a subset of games. Computer simulations show that this imperfect algorithm will still allow the network to find the Nash equilibrium of a game approximately 60% of the time. Dominance gets closer to explaining the network's actual behavior than Nash. Wee see that the network does display some strategic awareness, decreasing in the levels of thinking required. The network goes for high payoff values, and considers the temptation of the other player of deviating from Nash. It plays better in higher stakes games, particularly if there is more conflict of interests with the other player. The behavioral heuristics carry over when it faces new classes of games, games with multiple and zero pure Nash equilibria, and in fact the network seems to find focal solutions in games with multiple equilibria.

Keywords: rationality, learning, neural networks, normal form games, complexity

JEL Classification Codes: C45, C72, C99, D00

1. Introduction

Consider a naive agent thrown into a world of Nash equilibrium players. For example, he may be an infant who, first, starts playing games with parents and other relatives, and later with a larger and larger circle of people; he also observes other people playing games, either in front of him, or (say) on television. He does not always face the same game: on the contrary, he

Date: 24th October 2000.
Various incarnations of this paper have received the comments and the advice of Michael Bacharach, Susan Hurley, Gerry Mackie, Matthew Mulford, Raimondello Orsini, Andrew Oswald, Antonio Santamaura and participants to presentations in Oxford and Cambridge. Unfortunately the blame for the final product has to rest only on the shoulders of the authors. D.J. Zizzo is particularly responsible for the conception and design of the network in Part I, and for most of Part II; D. Sgroi is particularly responsible for most of Part I, and for the formalization of the algorithms in Part II. This notwithstanding, the entire paper is fully the result of a concerted effort. D. Sgroi would like to acknowledge funding from the ESRC and AEA Technology. Neural network simulations were made using the TLearn software package (see Macleod, Plunkett, and Rolls (1998)).

either plays or observes other subjects playing a wide range of games. Will he learn to play Nash strategies right away as he grows up, when facing games never encountered before? If yes, at what success rate? If the success rate is less than 100% but higher than chance, is it because the agent has *endogenously* learnt rules of thumb allowing him to perform *in a satisficing way* when playing new games? The rules of thumb would be learnt endogenously insofar as the agent's behavior would have always been reinforced according to Nash (he always plays with Nash players). This would be an example of emergent bounded-rational behavior as in Simon (1955), Simon (1959) or Rubinstein (1998): rather than playing the optimal strategy, the agent achieves a "good enough" solution (hence, he is satisficing, in Simon's terminology). Moreover, this would be a model of bounded-rational behavior in games in which, differently from other models, for example, Osborne and Rubinstein (1998), rules of thumb emerge endogenously as a result of the learning process rather than being exogenously super-imposed on the agent.

The issue of the learnability of Nash equilibria play is also relevant in its own right. Playing a Nash strategy in a game has long been thought a bare minimum requirement for rational play in games, and is treated as such in countless theoretical and applied papers. The challenge is to provide "a compelling argument for Nash equilibrium" (Mailath (1998), p. 1351). In the behavioral reinforcement paradigm, what we typically have are agents capable of reaching Nash equilibrium (or otherwise) in specific games, after a feasible long-run dynamic path as in Roth and Erev (1995) and Roth and Erev (1998). In the evolutionary game theory paradigm, a good deal of recent work has gone into justifying Nash equilibrium as a stable point in a dynamic learning or evolutionary process, for example, Young (1993) or Kandori, Mailath, and Rob (1993); yet, by itself it does not provide structural models of learning as Mailath (1998) is at pains to point out. This has not gone unquestioned. Furthermore, what much of the work in both of these paradigms has in common is an emphasis on learning to play a Nash strategy in a particular game. Start with an arbitrary strategy in a game and determine whether agents converge to playing Nash strategies, and thus a Nash equilibrium, in the long-run. This paper is very different, with the emphasis being placed on how reasonable Nash behavior is in play *across a number of games*, or throughout a player's economic life. An implication of this is that these models typically make no prediction, or predict naive behavior, when *new games* are faced.[1] Conversely, here the stress will be on the ability of the 'grown-up' agent to play in a non-naive way in games never encountered before, and even belonging to classes of games never encountered before.

In this paper, economic situations are modelled through normal form games. The network is exposed to a series of example games where the Nash choice is highlighted as the optimal action. Then the trained network faces a *new* series of games and is left to choose what strategies it sees fit. A neural network *could* in theory learn to play games never seen before in a fully rational way, picking Nash equilibria at the very first attempt, having been trained on a set of example

[1]Stahl (1998) provides an exception.

games. However, if we use the most standard learning algorithm used in neural network research, and assert the need for a minimum amount of biological and cognitive plausibility, we can show that the Nash equilibrium learning problem is too complex for the neural network. This is *not* something that can be addressed simply by having a more powerful learning algorithm, since, if anything, this algorithm - backpropagation - is *stronger* than a biologically plausible algorithm can be, as shown in Macleod, Plunkett, and Rolls (1998). The non-learnability result implies that, *in practice*, a neural network is much more likely to learn a local error-minimizing algorithm or LMA. The LMA is an algorithm that, if followed, minimizes the network's error in a subset, but only in a subset, of cases. LMAs are interesting because they correspond to one or more behavioral heuristics that the bounded-rational agent has endogenously picked up to perform in a satisfacing way on the decision problem (see Sgroi (2000)).

We present a neural network model that plays (up to) 3×3 games. A computer simulation methodology is required to check whether, after training on games with unique pure Nash equilibria, the network remains entirely naive in playing *new* (i.e. never encountered before) games, or it learns to play Nash, or it learns some LMA allowing it to perform in a satisfacing way on the decision problem. The trained network is able to find pure Nash equilibria of games never encountered before around 60% of the times, an empirically reasonable success rate as shown in Stahl and Wilson (1994). We show that, however, criteria based on iterated deletion of strictly dominated strategies (including rationalizability) outperform Nash whenever they yield a unique solution.

We then develop a simple econometric technique to test what game features the network has learnt to recognize to address the Nash solving problem in a satisficing way, as part of its LMA. The network displays some strategic awareness, but this decreases in the levels of iterated deletion of dominated strategies required. The network goes for high payoff values. It takes into account of potential trembles of the other player. It plays better in higher stakes games, particularly if there is more conflict of interests between itself and the other player.

The trained network's behavioral heuristics allow it to play in a meaningful way not just on new games, but on new classes of games, namely games with multiple and zero pure Nash equilibria. Moreover, networks trained on different sets of games - all with a unique pure Nash equilibrium - display focal points, when encountering games with multiple equilibria.

It may perhaps be considered a paradox that the non-learnability result is not here really a problem with neural network modelling, but rather a virtue: it is what makes this approach potentially innovative. It allows us to study how a toolkit of bounded-rational knowledge emerges endogenously, simply as the outcome of the agent learning to cope with too difficult a problem. This toolkit can then be analyzed using relatively straightforward regression analysis.

Neural networks are statistical pattern recognizers, and computer simulations employing neural networks are now a standard way to model categorization and learning in computer science, engineering and cognitive psychology. As yet, however, relatively little has been done within game theory to capitalize on this research. A notable exception is Rubinstein (1993), who uses perceptrons to capture the notion of a limit to forward-looking behavior in monopolistic competition. Another exception is the work summarized in Cho and Sargent (1996), on the usage of networks for encoding dynamic strategies. None of this research, however, uses computer simulations. Conversely, we believe that computer simulations can be helpful to study what strategies *are likely* to be encoded in a network as a result of repeated exposure to examples, rather than what strategies can in principle be encoded. Some computer simulations work has been done using neural networks as models of learning in single games, for example, Hutchins and Hazelhurst (1991) or Macy (1996), in which case they can be seen as simple extensions of a Roth and Erev (1995) style learning processes. Zizzo (2000c) shows how, through repeated exposure to different economic environments, networks can help in the modelling of the endogenous determination of preferences.

1.1. **Overview.**

The rest of this paper is organized as follows. Section 2 provides a purely intuitive introduction to neural networks and some additional motivation to the paper.

Part I of the paper encompasses sections 3 through 6, dealing with mainly theoretical issues. Section 3 presents the game to be played, section 4 formally presents the neural network player, and section 5 examines the learnability of the Nash solution by the neural network; section 6 develops the notion of the NP-hardness of the network's task, ending with a formal definition of a rule of thumb within this framework.

Part II of the paper, incorporating sections 7 through 11, addresses what the network actually learns, using computer simulations and simple econometric techniques. Section 7 introduces the simulated neural network, and section 8 details several alternatives to the Nash solution concept as possible alternative candidates for the neural network's behavior. Computer simulations, and an econometric analysis of the results, are described in section 9 for games with unique equilibria, section 10 for games with multiple equilibria, and section 11 for games with no equilibria.

Section 12 concludes.

2. Introduction to 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.

2.1. **What is a Neural Network?**

A feature of biological brains is that the connections between neurons are of different strengths, and that they can either increase or decrease the firing rate of the receiving neuron. In neural networks, this is modelled by associating a connection weight to each connection. This weights the input from the sending node to the receiving node. Since the weight can be either positive or negative, the activation of a node will either increase or decrease the activation of the receiving node.

Figure 1 illustrates a simple example of a neural network. Networks can be usefully thought of as agents that receive external stimuli, process them, and produce an output. A typical network has an input layer of nodes receiving stimuli from the outside (as real numbers). This input is then transmitted to the nodes the input layer is connected to, multiplied by the respective connection weights. Each node on the downstream layer receives input from many nodes. The sum is then transformed according to the activation function and the result is transmitted to the nodes in the further downstream layer. In such a way, the network processes the input until it reaches the output nodes (the output layer), in the form of new real-valued numbers. The activation level of the output nodes expresses the outcome of network processing, i.e., the network's decision. A feature of biological brains is that the connections between neurons are of different strengths, and that they can either increase or decrease the firing rate of the receiving neuron. In neural networks, this is modelled by associating a connection weight to each connection. This weights the input from the sending node to the receiving node. Since the weight can be either positive or negative, the activation of a node will either increase or decrease the activation of the receiving node.

[insert figure 1 here]

In other words, what the network does is a complex nonlinear transformation mapping the input (a vector of numbers) into an output (another vector of numbers). Our network for economic decision-making receives the payoff values of the game as inputs. As output, it produces a strategy. We can then determine whether the network's choice is optimal, and thus rational, or not. Since we are dealing with normal form games of full information, the optimal choice will be the Nash equilibrium strategy.

Generally, the optimum parameter or set of parameters cannot be calculated analytically when the model is nonlinear, and so must rely on a form of numerical optimization. The network adjusts connection weights during training following a basic learning algorithm (effectively a numerical optimization technique) called backpropagation developed in Rumelhart, Hinton, and Williams (1986)), discussed in the next section.

Intuitively, the economic decision-maker tries to learn how to perform better in the task. The more disappointing the outcome, the deeper the change will be. If a player faces a Prisoner's Dilemma, and cooperates while the other player defects, he will not be very willing to repeat the experience in the future. In the training set, the *correct* answer will be that dictated by the Nash equilibrium. The greater the difference between the network's behavior and the optimal (Nash equilibrium) strategy, the greater the adjustment that the learning algorithm will trigger.

2.2. Learning by Example.

The key to the learning process is repeated exposure to examples. The players are assumed not to have perfect access to models of the real world, nor are any rules explicitly taught to the network; rather they are simply subjected to a sequence of example games, and then asked to assimilate what general knowledge they can from these examples to play new, never before seen, games.

This appears a plausible rule for the learning of more rational decision-making. On the one side, economic agents face a large amount of decisions throughout their life. On the other side, it is unlikely that most of them ever encounter teachers telling them explicitly what general algorithm to follow in playing normal form games: rather, they implicitly learn to generalize their economic know-how from the examples they experience and observe, and get reinforcement from (see Zizzo (2000b)). So, as in Roth and Erev (1995), the basic idea is that of psychological reinforcement. However, here reinforcement does not entail direct adjustments to economic behavior. Rather, it operates directly on connection weights, and only indirectly on behavior.

The difference may appear subtle but is crucial. The behavioral learner learns how to behave better in an economic situation, but will be completely naive as soon as it faces a new one: knowing how to perform well in a coordination game tells me nothing on how to perform optimally in, say, a Prisoner's Dilemma. Instead, given enough exposure to examples, the neural network learner is able to find a set of connection weights that enables it to perform optimally a majority of times even in economic situations never encountered before. In other words, it learns how to generalize its economic know-how.

2.3. Prototype vs. Exemplar-Based Categorization.

The problem of a network with n connections is to find an appropriate configuration of its connection weights in the n-dimensional space of their possible combinations (Clark (1993)). If there are only a few examples, the network will assimilate novel cases to the most similar one, producing a similar output; if there are many examples, the network implements prototypical categorization (Way (1997)). Prototypes are the results of a process that extracts specific complexes of features - the statistical central tendency information - from a set of examples. Prototypical categorization is forced upon the network by its property that knowledge tends to be distributed across various nodes and connection weights, i.e. different examples tend to be coded over the same units; it follows that the features in which the different examples are common tend to be reinforced, whereas their differences tend to cancel out (Smith (1996)). New cases are assimilated to the nearest prototype, and the choice associated to that prototype follows.

2.4. Many Games.

Most evolutionary games, or learning dynamics, are based on the assumption that only one game is of interest. They then look at whether a player can converge to Nash behavior within that game. If we then consider another game we have to reset the dynamic and start again, forgetting the long process of learning to play Nash completely, or else assume that having mastered one game the player will simply pick a Nash equilibrium perfectly without any further need to learn. This paper goes well beyond these two simplifications. The decision algorithm used by the player is formed out of a series of observed examples, the results being a decision-rule in which the emphasis is on learning how to play games *in general*.

2.5. **Why backpropagation?** Backpropagation is the most standard learning algorithm used for computer simulations using neural networks: as such, it is a natural candidate to use. The basic intuition behind is that of psychological reinforcement: the economic decision-maker tries to learn how to perform better in the task, and the more disappointing the outcome (relative to the "correct" outcome), the deeper the change in connection weights will be. Backpropagation requires a teacher explicitly telling the correct answer during training, and this might appear too strong a requirement: it certainly makes backpropagation a more powerful algorithm than what is biologically plausible. Backpropagation is more powerful also in another sense: as we shall see, it adjusts individual connection weights using global information on how to best allocate output error. This is unlikely to occur in biological brains (Macleod, Plunkett, and Rolls (1998)). These limitations, however, should not be overstated: what they suggest is that backpropagation might be a plausible *upper bound* to the learning of biological neural networks of some given size. Conversely, stronger learning algorithms, of the kind used by White (1992) to show learnability, are not either biologically or cognitively plausible (White (1992), p. 161).[2] Hence, the non learnability result with backpropagation discussed in Part 1 should be taken seriously, and the methodology developed in Part 2 cannot be easily dismissed as an artificial product of too weak a learning rule. In practice, we do know that a neurotransmitter, dopamine, plays a role in biological neural networks analogous to that of the teacher in the backpropagation algorithm: the activation level of dopamine neurons works as a "behavioral adaptive critic", i.e. it tells the agent how to adapt its behavior to successfully deal with a task (Zizzo (2000c)).

2.6. **Empirical Success.** Neural networks are here treated as psychological models of how agents actually face, and learn to face, problems never encountered before. There is certainly evidence that children learn by example - either by direct experience or by observation of instances - as they grow up (see Bandura (1977)). More importantly, they are able not only to learn the examples observed (for example, to understand or utter words or sentences) but also to generalize from those examples (for instance, learn to talk: Plunkett and Sinha (1992)). In cognitive science, neural networks have been used as models, among other things, for how agents actually face pattern recognition and categorization, as in Taraban and Palacios (1994), for child development, as in Elman, Bates, Johnson, Karniloff-Smith, Parisi, and Plunkett (1996), for animal learning, as in Schmajuk (1997), and even arithmetic learning, as in Anderson (1998).Important analytical results (discussed in part I of the paper) exist on what the network *can* learn, but the focus of these computer simulation models is different: it is in showing what neural networks learn *in*

[2]We discuss this point more in depth below.

practice, and how they learn it. What networks learn in practice is also what, in this literature and hopefully in this paper, makes neural networks useful for psychological modelling. For example, a model of arithmetic learning that would predict the absence of mistakes is unlikely to be plausible when dealing with human subjects (Anderson (1998)); it follows that the fact that in practice the network converges to a solution algorithm with mistakes is more of interest than the fact that, in theory, an algorithm achieving a 0% mistake rate is learnable by the network.

Part I: The Theory of Neural Network Learning in Games

3. The Model

This section is devoted to a sequence of definitions, and a thorough explanation of the neural network model. The notion of *random games* will also be introduced, the key feature of which is that each game in a sequence will not have been played before by any player. This captures the notion of play in a new and changing environment and rules out standard theories of evolutionary learning in which the game to be repeatedly played is essentially unchanging.

3.1. **Basic Definitions.** 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:

$$u_i \left(a_i = \alpha_i \mid a_j = \alpha_j\right) > u_i \left(a_i \neq \alpha \mid a_j = \alpha_j\right) \tag{1}$$

We can say immediately that:

Proposition 1. *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.*

This is trivial given the definition of a Nash equilibrium. To say more we must first 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 $0 < \varepsilon < \varepsilon_y$:*

$$u\left(x, (1-\varepsilon)\, x + \varepsilon y\right) > u\left(y, (1-\varepsilon)\, x + \varepsilon y\right) \tag{2}$$

Now we can show that:

Proposition 2. *The unique Nash strategy of G is an evolutionary stable strategy (ESS).*

Proof. This 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 $u\left(\alpha_i \mid a_j = \alpha_j\right) > u\left(a_i \mid a_i \neq \alpha_i, a_j = \alpha_j\right)$ so by the uniqueness of α_i we know that any mix of α_i with $a_i \neq \alpha_i$ will reduce $u\left(\alpha_i, a_j\right)$ where $u\left(a_i, a_j\right)$ 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 $u\left(\alpha_i, a_j\right) > u\left(\beta_i, a_j\right)$ where $\beta_i \neq \alpha_i$. This in turn implies local superiority, so:

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

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. ∎

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 2 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 all other players will continue to play Nash strategies. Therefore, we can be assured that the neural network's best reply to the population will be to play a Nash strategy.

4. The Neural Network Player

Let us start with an intuitive account of the single *neural network player* in G. Consider a young economic agent with no prior experience of play in any game. This agent will, however, have a base of experience derived from a prior period spent learning "how to play". We might imagine a student's time at school or observation of how older members of the population play certain games. This agent has a store of observed example games, none of which will necessarily fit exactly with any game he will face in the future, but which might share certain similarities. We capture this notion of learning by example prior to entry into the economy, or marketplace of games, through the use of a *neural network*. We first "train" the agent by subjecting him to a series of example games with given actions and payoffs, we require him to be a utility maximizer, and then use a form of backpropagation to allow him to develop a form of pattern recognition which will enable him to "learn by example", and attempt to learn to play the Nash strategy in order to maximize his payoff. The question we ask is: can we introduce the network to a marketplace modelled by a sequence of random games filled with Nash players and expect the network to learn to play Nash? The key point is that he will be most unlikely to ever play the same game twice and will be most unlikely ever to have seen the identical game in his training period, so will his pattern recognition abilities be sufficient for him to intuitively recognize the Nash strategy in an entirely new game and play it? We are now switching out of the evolutionary framework and focusing on the play of a single player rather than the behavior of the population in general. We first need to add some formal definitions.

4.1. **Defining the Network.**

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.[3] The following definition formalizes this.

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*

[3]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*.

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*

$$a_i^k = \frac{1}{1 - \exp\left(-\sum_j w_{ij}^k u_{ij}^{k-1}\right)} \tag{3}$$

where u_{ij}^k is the output of node j in layer $k-1$ sent to node j 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 $x_k = \left(x_k^1, ..., x_k^{18}\right)$. 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 the pure Nash strategy is the top row $(0, 1)$ would imply the middle row, and $(0, 0)$ the bottom row.

4.2. Training the Network.

Training essentially revolves around finding the set of weights that is most likely to produce the desired output. 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.

Let us formally define the training sample. The vector $x_k = \left(x_k^1, ..., x_k^{18}\right)$ consists of 18 real-valued numbers drawn from a uniform $(0, 1)$ representing the payoffs of a 3×3 game. 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. 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:*

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

The labelled examples x_i are drawn independently according to the probability distribution 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 with 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 without 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.

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:*

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

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. Backpropagation is a form of numerical analysis akin to gradient descent search in the space of possible weights. Following Rumelhart, Hinton, and Williams (1986) we use a function of the form:

$$\triangle w_{ij} = -\eta \frac{\partial \varepsilon}{\partial w_{ij}} = \eta k_{ip} o_j \tag{4}$$

where w_{ij} is simply 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 the 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 o_{jp}. The higher o_{jp} is, 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:

$$k_{ip} = (g_{ip} - o_{ip})f'(t_{ip}) = (g_{ip} - o_{ip})o_{ip}(1 - o_{ip}) \tag{5}$$

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 equation 5, and update the weights of the connections between layer N and $N-1$, using equation 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:*

$$k_{ip} = f'(t_{ip}) \sum_{\beta} k_{\beta p} w_{\beta i} \tag{6}$$

to update the weights between layer $N-1$ and $N-2$, together with equation 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 equation 4 with a momentum function of the form:

$$\triangle w_{ij}^{t} = -\eta \frac{\partial \varepsilon^{t}}{\partial w_{ij}^{t}} + \mu \triangle w_{ij}^{t-1} \tag{7}$$

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 researcher, 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.

5. Inadequate Learning

We have now a clear idea of what the neural network is and the game that the neural network will face. 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 count as a subclass of 3×3.[4] 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).

5.1. Incomplete Neural Network Learning.

We now need to examine how well a neural network can do in theory, and fortunately various results exist in the literature to which we can refer. One of the most well-known results comes from Hornik, Stinchombe, and White (1989) reprinted in White (1992). Hornik, Stinchombe, and White (1989) show 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 they define as follows (in slightly amended form):

Definition 8. *(Hornik, Stinchombe, and White (1989)). 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, summarized in theorem 2.4 in Hornik, Stinchombe, and White (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 are three potential areas for failure:

1. *Inadequate learning*, or a failure of the learning dynamic to reach the global error-minimizing algorithm.

[4]While the neural network model was designed with 3×3 games in mind, since all payoff values are drawn from a uniform $[0, 1]$, it is straightforward to train the current network on 2×2 games, by putting zeros on all the values in the third row and third column of each game. Similarly we could construct 2×3 and 3×2 games by placing zeros in the appropriate row or column. The choice of a 3×3 is therefore much more general than the choice of 2×2, and can include several interesting and well-known 2×2 games.

2. *Inadequate network size*, or insufficient hidden units.
3. The presence of a stochastic rather than a deterministic relation between input and target.

Problem 3 can be extended to include poly-random functions (which cannot be distinguished from random functions by any polynomial approximation) but is still not a problem for the class of normal form games G. Problem 2 introduces a parallel mechanism for examining bounded-rational behavior. Along similar lines to the automata literature, we might restrict the number of hidden units in order to force bounded-rational behavior upon our player.[5] However, regardless of any attempts to raise the network size to cover for any potential problems, we cannot reasonably deal with problem 1: it is the problem of inadequate learning and the nature of the learning algorithm which is the focus of the rest of this section.

5.2. **Learning and Learnability.** A *learning algorithm* takes random training samples and acts on these to produce a hypothesis $h \in H$ 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 P_T on $Z = X \times Y$, if z is a training sample of length M drawn randomly according to the product distribution P^M, 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. A closely related definition is:

Definition 10. *We say that H is learnable if $\exists$ a learning algorithm for H.*

The function h_w 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 large enough time period we would hope that C will return a set of optimal weights which will in turn produce the algorithm h_w which will select the Nash strategy if $\exists$ a learning algorithm for selecting Nash equilibria (H in this case). Or alternatively if we wish

[5]Abreu and Rubinstein (1988) or Neyman (1985) provide examples of finite automata used in this way. In particular, the finite states of the automata proxy for the limited memories of bounded-rational players.

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.[6] 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.*

5.3. **Finding the Global Minimum.** 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 this approach leads only to a local minimum of the error function (see for example, Sontag and Sussmann (1989). White (1992), p. 160, 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." White (1992), chapter 9, theorem 3.1, 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. Generally, however, this is hardly surprising as backpropagation, for all its biologically plausibility, is after all a gradient descent algorithm.

The problem is exacerbated in the case of our neural network C as the space of possible weights is so large. Auer, Herbster, and Warmuth (1996) have shown that the number of local minima for this class of networks can be exponentially large in the number of network parameters. Sontag (1995) gives upper bounds for the number of such local minima, but the upper bound is unfortunately not tight enough to lessen the problem. 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.

6. Algorithm Complexity and Intractability

To fully understand the concept of NP-hardness first requires a primer in algorithm complexity theory. This section provides such a primer, and then moves on to give a proposition concerning the intractability of the Nash problem given in definition 11.[7]

[6]The exact algorithm for calculating the unique pure Nash strategy in a 3×3 normal form game with a single pure Nash equilibrium is given in the appendix.

[7]For even more detail see the classic Garey and Johnson (1979).

6.1. **A Formal Treatment.** We will start with one of the most standard machine intelligences in computer science, the *deterministic (one-tape) Turing machine* (or DTM). This is finite in the set of states (say Q), including one distinguished start state (q_0), and two halt states (q_Y and q_N), each corresponding to a "yes" or "no" answer to a decision-problem. The DTM will halt after it comes to a decision, either q_Y or q_N. We next need to consider a *language.*

Definition 12. *For any finite set Σ of symbols, denote by Σ^* the set of all finite strings of symbols from Σ. If Λ is a subset of Σ^*, we say that Λ is a language over the alphabet Σ.*

Call an abstract decision-problem Π, with a set D_Π of instances and a subset $Y_\Pi \subseteq D_\Pi$ of yes-instances. The correspondence between decision-problems and languages is brought about by *encoding schemes* used for specifying the problems instances whenever we intend to compute them.

Definition 13. *An encoding scheme e for a problem Π provides a way of describing each instance of Π by an appropriate string of symbols over some fixed alphabet Σ.*

Thus the problem Π and the encoding scheme e for Π partition Σ^* into three classes of strings: those that are not encodings of instances of Π, those that are encodings of instances of Π for which the answer is "no", and those that encode instances of Π for which the answer is "yes". This third class of strings is the language we associate with Π and e, setting:

$$\Lambda\left[\Pi, e\right] = \left\{ \sigma \in \Sigma^* : \begin{array}{c} \Sigma \text{ is the alphabet used by } e \text{, and } \sigma \text{ is the} \\ \text{encoding under } e \text{ of an instance of } I \in Y_\Pi \end{array} \right\}$$

If a result holds for the language $\Lambda\left[\Pi, e\right]$, then it holds for the problem Π under the encoding scheme e.

Returning to our DTM, we say that a DTM program or algorithm L with input alphabet Σ accepts $\sigma \in \Sigma^*$ if and only if L halts in state q_Y when applied to input σ. We say a language Λ_M recognized by the program M is given by:

$$\Lambda_M = \{\sigma \in \Sigma^* : L \text{ accepts } \sigma\}$$

Now we are ready to consider the first complexity class, P:

$$P = \{\Lambda : \text{ there is a polynomial time DTM program } L \text{ for which } \Lambda = \Lambda_M\}$$

This requires some explanation. Polynomial time programs *run* in polynomial time, that is a program or algorithm of inputs of size n is bounded by a running time of $O\left(n^k\right)$ for some finite constant k. This is clearly restrictive in the sense that exponential time would provide a far more generous bound for large n and any given k, but in all other senses seems a generous notion of time. Returning to the set P we will say that a decision-problem Π belongs to P under the encoding scheme e if $\Lambda\left[\Pi, e\right] \in P$, that is if there is polynomial time DTM program M that "solves" Π (i.e. reaches a decision) under encoding scheme e. It is standard within mathematical complexity theory and computer science to omit further references to e as there is a high level of equivalence between most encoding schemes.

Now consider a *non-deterministic Turing machine* (NDTM). This differs from a DTM in one major way. The DTM essentially follows a program L (or algorithm) in an attempt to find a solution (decision) for the problem Π. However, a NDTM can instead "guess" a solution, and then follow a stage similar to the DTM's program stage, by attempting to verify whether the guess was correct. The NDTM program is defined in the same way starting with q_0 and continuing until a halting stage is reached, but this time the aim is to verify (or refute) the guess rather than record a solution. The NDTM is said to record an *accepting computation* if it ends with the outcome q_Y, so the guess is verified. All the other notation from the DTM carries over, including the notion of languages, and recognition. In a similar way we can therefore define the complexity class, NP:

$$NP = \{\Lambda : \text{ there is a polynomial time NDTM program } L \text{ for which } \Lambda_M = \Lambda\}$$

It should be noted that $P \subseteq NP$ since any deterministic algorithm L could be used as the checking algorithm in a non-deterministic computation. So $\Pi \in P \Rightarrow \Pi \in NP$. The relationship in the other direction is somewhat more complex. It is widely believed that $P \neq NP$, so NP is a larger set than P, but this has yet to be proven. The rationale behind what is a standard belief in computer science and mathematical complexity theory comes about because of the existence of a number of problems belonging to the theorized set $NP - P$. Consider the abstract problem Π^*, which cannot be solved by a DTM in polynomial time, so $\Pi^* \notin P$, and consider such a problem to be solvable in polynomial time by a NDTM, so $\Pi^* \in NP$. Furthermore, let us strengthen this by defining Π^* to be the *hardest possible problem* solvable in polynomial time by a NDTM, but not solvable by a DTM. Hardness in this sense has a very simple meaning: every other problem in NP can be polynomially reduced to Π^*. Therefore, if Π^* could be solved by a DTM then so could every other problem in NP and NP would collapse to P. In this way we could consider the set containing Π^* to be the set of the hardest problems in NP, so hard that if solvable by a DTM in polynomial time we could say without hesitation that $NP = P$; we call this set the set of NP-complete problems. We have one member of NP-complete, but there is no reason to believe there would only be one such hard problem; it might be that there exist many problems which have the property that their solution by a DTM would collapse the NP set. As it happens there are literally dozens of such problems, and Garey and Johnson (1979) list numerous examples. Although a major research effort exists in computer science to attempt to solve these polynomial-equivalent problems, none have ever been solved by a DTM in polynomial time, which is why the concept of NP-completeness is such a strong one within complexity theory. The last concept we need is that of NP-hardness. We say something is NP-hard if it is equivalent in hardness to a member of the set of NP-complete problems. So it is a member, or rewritten version of a member, of the set of NP-complete problems. Perhaps a simpler way of saying this is that the problem is essentially intractable.

6.2. **NP-hardness.** Finally we now have the tools we need to deal with the learnability of the problem faced by C. Theorem 25.5 from Anthony and Bartlett (1999) succinctly states the following (in a slightly amended form):

Theorem 1. *(Anthony and Bartlett, 1999). The problem given in definition 11 faced by the class of networks encompassing C is NP-hard.*

Anthony and Bartlett (1999) chapters 23 to 25, provides various 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 3. *C supplemented by the backpropagation learning dynamic will not be able to learn the Nash algorithm in polynomial time.*

Proof. Backpropagation as a form of gradient descent is a DTM-algorithm. Therefore if the problem to be faced is NP-hard, by the definition of NP-hardness any DTM will therefore fail to find the minimizing algorithm L in polynomial time. A direct application of theorem 1 demonstrates that the problem is in fact NP-hard and we have our proof. ∎

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, a DTM 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.[8] However, if we were to supplement the algorithm with a guessing stage, i.e. add something akin to grid search or one of several theorized additions or alternatives to backpropagation, then we might expect to find the absolute minimum in polynomial time.[9] To restate this in terms of the

[8]This is intuitively seen to be reasonable with reference to two results. Fukumizu and Amari (2000) shows that local minima will always exist in problems of this type, and Auer, Herbster, and Warmuth (1996) show that the number of local minima for this class of networks is exponentially large in the number of network parameters. In terms of the theory of NP-completeness, we might say the solution could 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.

[9]White (1992), chapter 10, discusses a possible method which can provide learnability, using an application of the theory of sieves. However, White (1992), p. 161, stresses: "The learning methods treated here are extremely computationally demanding. Thus, they lay no claim to biological or cognitive plausibility." The method is therefore useful for computing environments and practical applications, rather than the modelling of decision-making. In other words his method is an application of an NDTM method not a DTM method.

search for an algorithm capable of providing Nash equilibria in never before seen games, backpropagation cannot do this perfectly, while other far less biologically plausible methods involving processor hungry guess and verify techniques, can produce consistent results.

So our player will find a decision-making algorithm that will retain some error even at the limit, or to put this an alternative way, we may have to be content with an algorithm which is effective in only a subclass of games, so it optimizes network parameters only in a small subspace of the total space of parameters. In the case of normal form games we can summarize this section 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.

6.3. **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).[10]

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 14. *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 given in the appendix, 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 15. *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 $z \in Z^M$.*

[10]For more on this see Sgroi (2000).

LMAs can be interpreted as examples of rules of thumb that a bounded-rational agent is likely to employ (for example, Simon (1955) and 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.

Part II: Analysis of Neural Network Learning

The network is unlikely to learn to play Nash at 100% success rate facing new games: it is facing too complex (NP-hard) problem for it to be able to do so using backpropagation. What is then the trained network actually learning to do? This is not a question that can be addressed analytically: however, computer simulations may help, together with simple statistical and econometric techniques to analyze the results of the simulations.

We first focus on the performance of the Nash algorithm and the robustness of the results to different parameter combinations: the results validate the unlearnability result in Part 1, but they also show that the trained network has learnt *something*. In the following section we deepen the analysis by analyzing the performance of different decision algorithms. We then develop an econometric technique based on a regression analysis on the network error ε to investigate what game features characterize the LMA adopted by the network. Later we modify this econometric technique to see how the trained network fares on games with zero and multiple pure Nash equilibria (PNE).

6.4. **Learning to Play Nash.**

The training set consisted of $M = 2000$ games with unique PNE. Training was random with replacement, 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.[11] 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 considered an output value to be correct when it is within some range from the exact correct value.

[11]Details on convergence can be found in Zizzo (2000c).

If both outputs are within the admissible range, then the answer can be considered correct (e.g., Reilly (1995)). The ranges considered were 0.05, 0.25 and 0.5, in decreasing order of precision.

[insert table 1 here]

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's (1994) experiment, 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 learnt 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.

[insert table 2 here]

So it appears that C^* has learnt 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.

7. Alternatives to Nash

Our first strategy in trying to characterize the LMA employed by the trained network 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 $\mu = 0$.[12] Hence, 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 PNE considered in the previous section.

We consider the following algorithms in turn:

1. Minmax
2. Rationalizability
3. '0-level strict dominance' (0SD)
4. '1-level strict dominance' (1SD)
5. 'pure sum of payoff dominance' (PSPD)
6. 'maximum payoff dominance' (MPD)
7. 'nearest neighbor' (NNG)

7.1. **Minmax.**

Minmax is often considered a form of reservation utility, and can be defined as:

Definition 16. *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. An algorithm which looks for the minmax payoff is of course a local error-minimizing algorithm in the sense of definition 2 when the subclass of games faced is zero-sum, or by a minor reworking of utility, constant sum.

[12]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.

7.2. Rationalizability and Related Concepts.

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 players, 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. In this terminology 2SD would be full rationalizability or the full iterated deletion of strictly dominated strategies as defined above.[13]

7.3. Payoff Dominance.

PSPD and MPD are different ways of formalizing the idea that the agent might try to go for the largest payoffs, independently of strategic considerations.

In the case of PSPD, an action $a_i = a_{PSPD}$ is chosen by the row player according to:

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

Where $\triangle_j$ is defined as a perfect mix over all available strategies in A_j. Put simply, a_{PSPD} 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.

MPD is even more lowly in its required level of rationality. If following this algorithm, 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.

Both PSPD and MPD are strategically unsophisticated as candidate LMAs. What they catch is the basic idea of 'going for the highest pot of money'.

7.4. Nearest Neighbor.

The NNG is an algorithm that draws its roots from cognitive science. As we discussed in section 2, neural networks are considered models of the categorization process,

[13]It requires two levels of reasoning in the sense of Nagel (1995).

of how decision-makers classify input in order to produce the correct output. We also mentioned that one possible way in which networks may do this, when the sample is small enough, is by behaving like an exemplar-based model of categorization: that is, they may assimilate the instance to the nearest instance (example) encountered in the past, and solve the decision problem accordingly.

The best way to formalize this approach is not in terms of expected utility theory, but rather in terms of an alternative paradigm which emphasizes the role of past experiences in decision-making. Gilboa and Schmeidler (1995) provide such a paradigm in the form of *case based decision theory.* Altering the terminology to better match that used in this paper, Gilboa and Schmeidler (1995) consider G^P and A to be finite, nonempty sets, of games and strategies (or actions) respectively, with all acts available at all problems $p \in G^P$. X is a set of realized outcomes. x_0 is included within X as the result "this act was not chosen". The set of *cases* is $C \equiv G^P \times A \times X$, which lists all conceivable combinations of games, strategies and realized outcomes. When a player considers a current game, that player does so in terms of the game itself, possible strategies, and realized outcomes which spring from particular choices of strategy. Importantly, the player is not asked to consider hypothetical outcomes from untried stratagies, rather the player considers only past experiences of realized outcomes. To make this concrete, given a subset of cases $C_s \subseteq C$, denote its projection P by H. So,

$$H = H(C_s) = \left\{q \in G^P \mid \exists a \in A, x \in X, \text{ such that } (q, a, x) \in C_s\right\}$$

Where H denotes the history of games, and $C_s \subseteq C$ denotes the subset of cases recorded as memory by the player, such that (i) for every $q \in H(C_s)$ and $a \in A, \exists$ a unique $x = x_{C_s}(q, a)$ such that $(q, a, x) \in C_s$, and (ii) for every $q \in H(C_s), \exists$ a unique $a \in A$ for which $x_{C_s}(q, a) \neq x_0$. Memory is therefore a function that assigns results to pairs of the form (game, strategy). For every memory C_s, and every $q \in H = H(C_s)$, there is one strategy that was actually chosen at q with an outcome $x \neq x_0$, and all other potential strategies are assigned the outcome x_0. So, our agent has a memory of various games, and potential strategies, where one particular strategy was chosen. This produced a result $x \neq x_0$, with other stratagies having never been tried, so given the generic result x_0.

When faced with a problem, our agent will examine her memory, C_s, for some similar problems encountered in history, H, and assign these past problems a value according to a similarity function $s(p, q)$. These past problems each have a remembered action with a given result, which can be aggregated according to the summation $\sum_{(q,a,x) \in C_s} s(p, q) u(x)$, where $u(x)$ evaluates the utility arising from the realized outcome x. Decision making is simply a process of examining past cases, assigning similarity values, summing, and then computing the act a to maximize $U(a) = U_{p,C_s}(a) = \sum_{(q,a,x) \in C_s} s(p, q) u(x)$.

Under this framework, the NNG algorithm examines each new game from G^P, and attempts to find the specific game p from the training set (which proxies for memory) with the highest valued similarity function. In this paper, similarity is computed 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. This sum of squares is a measure of dissimilarity between the new game and each game in the training set; if, in the limit, there were two identical games (which is never the case), the value would be exactly equal to 0. The game with the lowest dissimilarity index is defined as the *nearest neighbor*. The NNG algorithm looks for the game with the lowest dissimilarity index and chooses the unique pure NE corresponding to the nearest neighbor. This is somewhat different from the case based decision theory optimization which involves a summation over all similar problems, but serves as a first approximation, and could be further generalized to incorporate a more complex similarity function.

7.5. **Existence.** 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.[14] 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.

We define a game as *answerable* by an algorithm if a unique solution exists. Table 3 lists the number and percentage of non 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 $\mu = 0$.

[insert table 3 here]

7.6. **Algorithm Performance.** Table 4 describes how well the various algorithms fare on the testing set.

[insert table 4 here]

Algorithms are classified into two groups, according to their performance. MPD, PSPD, Minmax and, most interestingly, NNG fare worse than Nash on the data. 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. 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. One must recognize, though, two limitations to this result. Firstly, partial nearest neighbor effects cannot be excluded in principle on the basis of table 4. Secondly,

[14]The only potential (but unlikely) exception is if the nearest neighbor is not unique, because of two (or more) games having exactly the same dissimilarity index. The exception never held with our game samples.

perhaps C^* does not simply rely on the nearest neighboring game; perhaps it relies on a weighted average of most similar games, although placing the most weight on the nearest game: this more sophisticated version of exemplar-based categorization would not be fully captured by the NNG.

Rationalizability, 0SD and 1SD outperform Nash for the games they can solve in a unique way. 0SD, 1SD and rationalizability predict C^*'s behavior in 80.98%, 76.25% and 74.36% of their answerable games, respectively: this is 8-14% above Nash, and even more striking considering that these algorithms were not taught to the network. It does appear as if C^* were able to do some strategic thinking (it still gets about three quarters of the rationalizable games exactly right), but the more the level of iterations (one relative to zero, full rationalizability relative to one) the more difficult it is for the network to do this properly.

C^*'s behavior is best *describable* by algorithms based on iterated deletion of strictly dominated strategies. However, it is not equally clear that these alternatives to Nash fully describe the network's LMA. This is true not only because the network still fails to apply these methods perfectly, but also and more importantly because the network can still play reasonably well in games that are non answerable according to the alternatives: this explains why, if one considers the overall set of 2000 games rather than just the answerable games, Nash is still the single best predictor. An objection to this result is that the right half of Table 3 is biased against the comparative success of the algorithms vs. Nash: it does not consider the fact that C^* might be playing randomly on the non answerable games and, in so doing, it would still get an average 1/3 of these games correct by chance. However, even if we augment rationalizability, 0SD and 1SD with random play on their non answerable games, and so we credit 1/3 of their non answerable games as correct, it is still the case that Nash outperforms. Augmented rationalizability gets closest, with 57.68% of the 2000 games correct (with $\gamma = 0.02$): this is still 9 points below Nash (see Table 4).

We might conjecture that the successful alternatives to Nash are successful because they characterize game features that are also considered by the LMA achieved by C^*, but, also, that the LMA achieved by C^* does not simply coincide with these successful alternatives. We now need to make this conjecture more precise.

8. Game Features

A second strategy that we may use to gather information about the network's LMA is to analyze the game features that it has learnt to detect. If the LMA uses certain game features that it exploits to perform well on the game-solving task, then C^* will perform better on games that have those game features to a high degree. This means that the network error ε will be less for games having these features. We can use this intuition to develop an econometric technique allowing us to get information on the game features detected by C^* to address the task, and, hence, on what C^* has actually learnt. The econometric technique simply consists in running tobit regressions of game features on ε; tobit regressions need to be used because the distribution is truncated at 0, the lowest possible ε.

In a prototype-based view of categorization, we could say that games in which those features were to be present to a high degree would be prototypical games, i.e. games that the network i would be able to classify (perfectly or almost perfectly) as best examples of games associated with playing an action from A_i. In an exemplar-based view of categorization, if the network were to be sensitive to the specific examples encountered in the training set, and if the Nash equilibrium were also the nearest neighbor, we would expect that the network would perform significantly better in these games than otherwise. This would be true regardless of whether weight is given only to the nearest neighbor game in the exemplar-based algorithm, as we would expect the most weight to be given to the nearest neighbor (neighbor 1) relative to neighbor $2, 3, ..., n$.

In our case, we can use the average ε of the thirty neural networks trained with $\eta = 0.5$ and $\mu = 0$, and achieving a convergence level $\gamma = 0.02$, as the dependent variable. 30 observations presented 0 values, implying a perfect performance by the neural network whatever the random seed.

The game features that were used are listed in figure 3 together with the results; they can be classified in three groups:

1. *Algorithm related features.* These are dummy variables equal to 1 when the feature is present, and to 0 otherwise. MPD and Minmax Existence look at whether a unique MPD or Minmax solution exists for the game; the "Same As" variables look at whether the algorithm (e.g., PSPD) has the same prediction as the Nash strategy for the game. Existence variables are not defined for algorithms that always exist (e.g., NNG). In the case of the strict dominance algorithms, we chose instead to use three dummy variables for the cases in which zero and exactly zero, one and exactly one, two and exactly two iteration levels are required to achieve a unique solution: these dummies are represented by "Strict Dominance: Level 0 Sufficient", "Strict Dominance: Need for Level 1" and "Strict Dominance: Need for Level 2", respectively. NE Action 1 and 2 are simply dummies equal to 1 when the Nash strategy is actually 1 (Top) or 2 (Middle), respectively.

2. *Payoff and Temptation variables.* These variables relate to the size of the Nash equilibrium payoff for the network and the other player, and to the size of deviating from this equilibrium. Own Generic Temptation is a crude average of the payoff from deviating from Nash, assuming that the other player plays randomly. Max and Min Own Temptation are the maximum and minimum payoff, respectively, from deviating from Nash, taking the behavior of the other player as given. Clearly, while the Generic Temptation variable assumes no strategic understanding, the Max and Min Own Temptation variables do assume some understanding of where the equilibrium of the game lies. Ratio variables reflect the ratio between own and other's payoff in the Nash equilibrium, minus 1: if the result is positive, then the Positive NE Ratio takes on this value, while the Negative NE Ratio takes a value of 0; if the ratio is negative, then the Positive NE

Ratio takes a value of 0, while the Negative NE Ratio takes on the negative value, in absolute terms.

3. *General game features.* These variables are mostly based on the moments of the game payoff distribution. We consider the mean, standard deviation, skewness and kurtosis of the game payoffs for each game; we also consider the difference between their values observed for each game and their average value across the 2000 games. The Game Harmony Index is presented in the appendix, and discussed in Zizzo (2000a). It is a measure of how harmonious or disharmonious the players' interests are in the game: it is equal to 0 if the game is perfectly harmonious, such as in the case of a pure coordination game; it take greater values the greater the conflict of interests. The index is derived from the Gini coefficient of income distribution and is bounded between 0 and 1.

[insert table 5 here]

Table 5 presents the results of three tobit regression models: likelihood-ratio tests accept the reduction to the simplest model, Model 3.

The results yield a wealth of information on what the network is actually doing:

1. *Go for high numbers, especially if they are yours.* The network gives better answers when the NE is associated with a high payoff - particularly for itself. The Same As MPD, the Same As PSPD and the Strict Dominance variables all work in the same direction. The coefficients on these variables are relatively small, though: this suggests that, although the network's behavior can be best described by the strict dominance algorithms (relative to the others) in the context of games with a unique PNE, the network may actually be picking game features associated with strict dominance, rather than simply following strict dominance as a rule of thumb.

2. *Feel and fear trembling hands.* The greater the temptation, the greater the chance of deviating from the right answer. The fear of the other player's temptation may be related to an expectation of potential trembles by the other player, trembles that might be assumed greater the greater the temptation. Again, more weight is given to one's own temptation than to the other player's, but the network does appear to give some strategic weight to the temptation of the other. More weight is also given to deviations from Nash playing taking the action of the other player as given, another sign of some strategic reasoning in feeling or fearing the temptation of random trembles.

3. *The greater the strategic complexity, the greater the difficulty.* The coefficients on one's own temptation from deviating from Nash taking the action of the other player as given is higher than the coefficient on the other's temptation taking one's own action as given; however, the coefficient on the other's generic temptation is higher than that on one's own generic temptation. The own Nash deviation temptation requires just a model of the other player's as taking one's own action as given; whereas a consideration of the other player's Nash deviation temptation requires one more

level of strategic understanding (in the sense of Nagel (1995)), namely a model of the other player having a model of oneself as taking the other player's action as given. Faced with a more difficult task, the neural network puts more weight on the other player's generic temptation. There are two other signs that, the greater the strategic complexity, the greater the difficulty for the neural network. One is that the network puts more weight on one's own than on the other's payoff; the other is that the coefficient on Strict Dominance: Need for Level 0 is significantly higher (in absolute terms) than the one on Need for Level 1, which is higher (again, in absolute terms), albeit insignificantly so, than the one on Need for Level 2.

4. *High stakes provide 'motivation' to the agent.* C^* finds difficult to process payoff values distant from the mean payoff value (of 0.50154), but finds still more difficult to process games for low stakes (because of the negatively signed Game Harmony $\times$ Mean term; the negative Skewness term is also indicative). This is an interesting and plausible prediction: in any laboratory experiment setting subjects have to be motivated enough by the stakes at play; similarly, C^* makes smaller mistakes when the stakes are higher, as if 'motivation' were required.

5. *Keeping game harmony constant, an increase in payoff variance induces less correct answers.* Insofar as it is not correlated to greater game disharmony, a greater standard deviation is likely to proxy an increase in the variance in one's own, or the other player's, possible payoffs. The prediction here is that the agent may be less keen playing the Nash strategy than he otherwise would be, if there is an increase in the variance in one's own possible payoffs.

6. *When the game is for high enough stakes, higher game disharmony induces more correct answers.* An increase in the *Game Harmony Index* (GI) implies greater game disharmony. In Model 3, this feeds into the error ε through two channels: one, positively signed, is the Standard Deviation, because an increase in GI is also likely to increase SD (Pearson $r = 0.526$); the other, negatively signed, is the GI Index $\times$ Mean term. The two effects operate in opposite directions and, also, the second channel will be stronger the higher the mean. In order to analyze which effect is likely to dominate, we ran an OLS regression of SD on GI; the coefficient on GI is 0.19 (S.E.=0.007), and used this as a proxy for $(\Delta\text{SD})/(\Delta\text{GI})$. We analyzed how, for various mean levels, GI increases between 0.05 and 0.5 affect ε; figure 2 plots the results for ΔGI=0.1, 0.3 and 0.5. The results are ambiguous for low mean payoff values, but as soon as the game has stakes high enough the GI Index $\times$ Mean effect dominates and produces substantial decreases in ε. This effect of game disharmony in improving the likelihood of Nash strategy play is an interesting prediction for experimental settings. In thinking about situations in which rational agents find difficult to decide, economists often think at games with high or perfect game harmony (e.g. the Prisoner's Coordination Game of Sugden (1993)).

[insert figure 2 here]

7. *Nearest-neighbor effects exist but are limited.* The coefficient on *Same As Nearest Neighbor* is significant and with the expected sign, but the effect is not large. While more sensitive to examples than a pure prototype model would be, C^* follows more a prototype-based approach to the categorization of the new games it encounters.

8. *Other processing features.* Kurtosis has a small but significant positive effect on ε. The network also has a slightly higher error with actions 1 and 2, quite possibly because of its greater familiarity in producing zeros than ones as outputs.

In conclusion, C^* appears to have found ways to get around the problem of attempting to find a Nash strategy in never before seen games. They rely on a plausible mix of payoff dominance and potential trembles, and on some strategic awareness that decreases in the strategic complexity required. While not being taught to the network, they are games features that correspond to emergent behavioral heuristics characterizing the LMA endogenously chosen by the bounded-rational agent.

9. Multiple Equilibria

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

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, our best describing algorithm for the single PNE case, then the network should be unable to choose among the plurality of PNE, as these all correspond to rationalizable solutions. On the basis of section 8, however, we hypothesize that this will not be the case, even if the network has never faced games with multiple PNE in the training stage.

9.1. Focal Points.

A second, stronger hypothesis is that 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 PNE in games with multiple PNE. Why this should be the case is different according to whether we view networks as working mainly as an exemplar or as a prototype-based model of categorization. In an exemplar-based perspective, 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 PNE: hence, there will be a focal solution. However, one might wonder whether the differences in random seeds are really so important, particularly given the fact that training is random but with replacement: perhaps, we are dealing with basically the same neural networks in each case, and so the finding of focal points may 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 works mainly as a prototype-based model of categorization, 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 strong 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; in addition, 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 works mainly as a prototype-based model of categorization, and that, therefore, it will display not only w-focal points but also s-focal points. Since it retains some (intuitively plausible) 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.

The testing set was made of 2000 games again, namely 100 games with three PNE and 1900 games with two PNE. 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 PNE: 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, PNE.

For the three PNE 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 PNE 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 2 PNE dataset under the second null, $\chi^2 = 30785.17, 1898$ *d.f.*, for Set 1,[15] and $\chi^2 = 23985.49, 1899$ *d.f.* for Set 2. In

[15]One game had to be removed in relation to this test because the corresponding expected values were zero for the second null.

all cases, using χ^2 tests the null hypotheses are strongly rejected (at $p < 0.001$) for both Set 1 and Set 2.

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 PNE or three PNE games, suggesting that the importance of focal points is somehow lower with s-focal points, as it might be expected by the limited exemplar effect. Nevertheless, the strong evidence for s-focal points suggests once again that the network is mainly reasoning as a prototype-based model of categorization. Different neural networks, trained on the same game distribution although on different games, must be displaying focal points because they have learnt 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 the number of decided choice tends to be focal on specific choices, we have not shown that the number of decided choices is high in the first place in games with multiple PNE. 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 PNE.

9.2. **Features of Focal Games.** What are the game features that make a solution focal? Unfortunately, we cannot use the econometric technique discussed in section 8, because in games with multiple equilibria there is not a correct solution relative to which compute ε. We therefore need to develop a different technique. Let us define three data-points for each game, in correspondence to each action: one data-point corresponding to the number of decided choices from playing action 1, one from playing action 2 and one from playing action 3. This allows to obtain, in principle, a dataset of 6000 observations, in correspondence to the 2000 games: let us label these observations as *NDecided1* if they are based on the number of decided choices with Set 1, and *NDecided2* if they are based on the number of decided choices with Set 2. We can now do ordered probit regressions of a variety of game and action features on NDecided1 and NDecided2, in order to determine what makes the network choose an action rather than another one.[16]

Many of the features considered are identical or similar to the ones previously considered; a few are new ones:

1. Algorithm related features. As before, these are dummy variables equal to 1 when the feature is present, and to 0 otherwise. Same As 0SD and 1SD refer to the case in which the action is dominated according to 0 or 1 iteration level strict dominance.[17] Same As NE When 2NE marks

[16]It might be better to estimate an ordered probit model with random effects, but unfortunately we were unable to compute it. OLS regression models with random effects do not pass the Hausman specification test. OLS regression models with fixed effects display insignificant fixed effects, the joint deletion of which is easily accepted with F tests. Similar considerations apply, and similar results results were obtained, in relation to the work of the next section.
[17]More than one iteration is never needed with games with multiple PNE.

the case in which the action corresponds to one of exactly two PNE present in the game - for the games with three PNE, each action is a pure Nash equilibrium, so it would not be a useful marker. Conversely, Presence of 3NE marks the games with three PNE. Same As Utilitarian Best is a marker for when the action corresponds to the best pure Nash equilibrium from a utilitarian perspective (e.g., that of the sum of the payoffs).

2. *Payoff and Temptation variables.* We need to modify these variables because we are considering the desirability of each action, not just of PNE actions. We reformulate the variables in terms of Best Response (BR): given that the neural network chooses a particular action, what is the strictly best response of the other player? BR outcomes will be defined for all actions in which the top payoff for the column player in correspondence to the action being considered is strictly higher than the others. This is the case for all but three observations in our sample; in the regression analysis, we drop these three observations and restrict ourselves to a sample of $n = 5997$ observations. We also add two sets of new variables. First, we introduce temptation variables (Own/Other's BR Max/Min Temptation) on the basis of the (min/max, own/other's) BR payoffs from playing the other action. Second, we introduce two interaction terms: Game Harmony $\times$ Own Temptation and Game Harmony $\times$ Other's Temptation.

3. *General game features.* These are similar to those previously considered.

[insert tables 6 and 7 here]

Tables 6 and 7 present the result of ordered probit regressions on NDecided1 and NDecided2; the restrictions entailed by the simpler models in both figures are, once again, accepted using likelihood-ratio tests. There are some differences between the two figures - particularly, game harmony appears a somehow better predictor of NDecided2 than NDecided1 -, but in general the picture is quite similar.

9.3. **Results.** We can now summarize some results.

1. *Go for high numbers, especially if they are yours.* This appears to be still true, as shown by the coefficient values on Own and Other's Payoff, Same as PSPD, Same As MPD and Same as Utilitarian Best. As before, more weight is put on one's own payoff than on the other player's. On the other hand, consideration of potential trembles may make the network weigh the other player's welfare, both directly (Other's Payoff) and indirectly - utilitarianism appears a salient feature to choose among PNE.

2. *Feel and fear the trembling hands.* This is still true, as shown by the generally negative significant temptation effects. Coefficients are mostly larger for one's own temptation, but the reverse is true for the minimum temptation of the other player.

3. *Strategic awareness.* The neural network appears capable of detecting relevant strategic features, as shown not only by the weight put on most BR temptation variables, but also by Same

As Utilitarian Best (which requires the utilitarian solution to be a PNE) and, more importantly, by Same As NE When 2 NE. Considering the positive coefficient on Presence of 3NE, this means that, plausibly, the network makes more decided choices in relation to PNE actions when there are 2 PNE, than when there are 3 PNE, and last when there is not a PNE. Additional weight is given if the action is strictly dominant, even if just so using 1-level iteration of strictly dominated strategies.

4. *High stakes provide 'motivation' to the agent.* This is still true, as shown by the large coefficient on Mean.

General game features appear to play less of a role in tables 6 and 7 than they did in table 5. This is not really surprising: table 6 was a regression in relation to the *best* (i.e. Nash equilibrium) action only, whereas here we are regressing in relation to all actions, good and bad, and these will share the same general game features. Hence, the only way this may matter is in increasing the overall level of decided choices in the game - but not really in making the neural network more decisive towards one action relative to another.

In conclusion, C^* displays both w-focal points and s-focal points in games with multiple PNE. It does so because it tries to apply to these games, as much as possible, the same LMA that it has learnt to apply to games with unique PNE in a satisficing way. The existence of s-focal points corroborates once again the view that C^* mainly reasons on the basis of prototypical categorization; moreover, most of the game features it has learnt do not yield to undecidability paradoxes when facing games with multiple PNE (as, say, the PNE or rationalizability logic would), and so can be applied successfully to coordinate on focal choices. These results are the more striking because C^* never faced a single game with multiple PNE during training.

10. Games with No Equilibria

Unfortunately, the models of the previous section cannot be compared to those of section 8 in two important respects: (i) the usage of different endogenous variables, which has led to the usage of partially different exogenous variables (e.g., based on the concept of Best Response); (ii) the application to different sets of games; games with a unique PNE in one case, and games with multiple PNE in the other case. A third limitation of the previous analysis is that, although we have analyzed cases with 1, 2 or 3 PNE, for sake of generality we should also analyze games with 0 PNE. After all, if the claims of the previous section are correct, we would expect the neural network to play in a meaningful bounded-rational way also in this set of games.

In this section we try to address these three concerns. To do this, we computed 6000 NDecided1 observations in relation to the games with unique PNE analyzed in sections 6 through 8. We also computed 2000 games with 0 PNE, and used these games to test the thirty networks trained with $\eta = 0.5$, $\mu = 0$ and thirty different random seeds; we were then able to derive 6000 NDecided1 observations in relation to these games with 0 PNE. Again we exclude the observations in relation to which there is not a strictly dominant Best Response on the part of the column player: this leaves us with $n = 5993$ with the games with 1 PNE and $n = 5983$ with the games with 0 PNE. We then ran ordered probit regressions using the same set of regressors as much as possible.[18] There are less regressors with 0 PNE games because, unavoidably, there are more strategic features that cannot be exploited, either by us, or by the network. The average number of decided choices per action in the 0 PNE dataset is 8.28, lower not only than games with a unique PNE but also than games with multiple PNE. Nevertheless, it is still a considerably high amount, given that the maximum is 10.

[insert tables 8 and 9 here]

Tables 8 and 9 contain the results of the ordered probit regressions, in relation respectively to games with 1 and 0 PNE. The neural network still tends to go for high numbers, especially for itself. Same as 0SD is wrongly (i.e., negatively) signed in table 8, but this is really just a reduction of the positive effect that zero-iteration dominant action has on NDecided1. This is true, first, because Same as 1SD will always be equal to 1 when Same as 0SD is equal to 1, and the coefficient on Same as 1st is large and positive; second, because, in the present sample, 0SD is always a sufficient condition for PSPD and MPD, which implies an additional and large positive effect. Similarly, the occasional wrong signs on the Temptation variables can be explained because of a partial counterbalancing of the large interaction effects with Game Harmony. This is not the case, however, for Own and Other's Min Temptation with games with 0 PNE.

Game Harmony plays a significant role in games with unique and zero PNE (unlike games with multiple PNE). In line with the findings of section 9, a higher game harmony index, i.e. greater game disharmony, induces more decisive choices.

11. Conclusions

This paper has presented a neural network model to simulate the endogenous emergence of bounded-rational behavior in a normal form game framework. Potentially any finite normal form could be modelled in this way, though we have concentrated on 3×3 games, and noted that 2×2, 2×3 and 3×2 games count as a subclass of 3×3. The inclusion of a neural network player in a population of Nash players does not change the behavior of the Nash players, and the 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 (NP-hardness) of the Nash problem: effectively, the Nash

[18]Some differences are made necessary by the way the datasets are constructed: for example, we clearly cannot use Same As NE as a regressor in the dataset of games with 0 PNE! Similarly, since the utilitarian criterion used in section 8 was made dependent on a choice among PNE, it cannot be introduced in the datasets with either 1 or 0 PNE.

algorithm is intractable by a network that uses learning algorithms, such as backpropagation, with a minimum of 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 'satisfacing' 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 learnt by the bounded-rational agent.

The simulation results suggest a figure of around 60% success on games never encountered before, as compared with 33% as the random success benchmark or the 59.6% experimental figure from Stahl and Wilson (1994). Such simulations also indicate that solution concepts other than Nash and based on dominance get closer to explaining the simulated network's actual behavior. We develop simple econometric techniques based on regression on the network error and on the 'decidedness' of the network's choice to characterize the local error-minimizing algorithm it has achieved. The network displays some strategic awareness, but this is not unbounded, and is decreasing in the levels of iterated deletion of dominated strategies required. The network goes for high payoff values. It takes into account potential trembles due to the temptation of the other player of deviating from Nash. It plays better in higher stakes games, particularly if there is more conflict of interests between itself and the other player.

The trained network's behavioral heuristics carry over to a relevant degree when it faces not just new games, but new classes of games, namely games with multiple and zero pure Nash equilibria. Moreover, networks trained on different games - all with a unique pure Nash equilibrium -, are able to coordinate on the same focal solution, when encountering games with multiple equilibria.

Our results suggest that, perhaps paradoxically, the fact that the network converges to a local error-minimizing algorithm is not a problem but a virtue. It is what makes the research approach used in this paper - based on neural network simulations, and econometric techniques developed to analyze the results of these simulations - potentially interesting. It has allowed us to model and make predictions about bounded-rational behavior in normal form games, with rules of thumb emerging endogenously as a result of the learning process rather than being exogenously superimposed on the agent.

Appendix

A. The Nash Algorithm. Consider a 2 player 3×3 normal form game G with payoff matrix X,

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} \end{pmatrix}$$

Where payoffs are in pairs, so x_1 and x_2 are the payoffs to players 1 and 2 respectively from the realised outcome (top, left). Strategies are $a_i \in \{a_i^1, a_i^2, a_i^3\}$ for players $i = 1, 2$. More specifically, for the row player ($i = 1$), $a_1^1 =$ "*top*", $a_1^2 =$ "*middle*", and $a_1^3 =$ "*bottom*", and for the column player ($i = 2$), $a_2^1 =$ "*left*", $a_2^2 =$ "*middle*" and $a_2^3 =$ "*right*". Now let the vector y denote the choice of strategy for player 1 so $y = (1,0) \Leftrightarrow a_l = a_l^1$, $y = (0,1) \Leftrightarrow a_l = a_l^2$ and $y = (0,0) \Leftrightarrow a_l = a_l^3$. Now express the unique Nash strategy for player 1 as y^*. The Nash algorithm for 3×3 games with a unique Nash strategy is then:

$$(8) \qquad y^* = (1,0) \text{ if } \begin{cases} (x_1 > \max\{x_7, x_{13}\} \ \& \ x_2 > \max\{x_4, x_6\}) & \text{or} \\ (x_3 > \max\{x_9, x_{15}\} \ \& \ x_4 > \max\{x_2, x_6\}) & \text{or} \\ (x_5 > \max\{x_{11}, x_{17}\} \ \& \ x_6 > \max\{x_2, x_4\}) \end{cases}$$

$$(9) \qquad y^* = (0,1) \text{ if } \begin{cases} (x_7 > \max\{x_1, x_{13}\} \ \& \ x_8 > \max\{x_{10}, x_{12}\}) & \text{or} \\ (x_9 > \max\{x_3, x_{15}\} \ \& \ x_{10} > \max\{x_8, x_{12}\}) & \text{or} \\ (x_{11} > \max\{x_5, x_{17}\} \ \& \ x_{12} > \max\{x_8, x_{10}\}) \end{cases}$$

$$(10) \qquad y^* = (0,0) \text{ if } \begin{cases} (x_{13} > \max\{x_1, x_7\} \ \& \ x_{14} > \max\{x_{16}, x_{18}\}) & \text{or} \\ (x_{15} > \max\{x_3, x_9\} \ \& \ x_{16} > \max\{x_{14}, x_{18}\}) & \text{or} \\ (x_{17} > \max\{x_5, x_{11}\} \ \& \ x_{18} > \max\{x_{14}, x_{16}\}) \end{cases}$$

The restrictions on G ensure that $y^* \neq \emptyset$ and at most one of expressions 8, 9 and 10 are true.

B. The Gini-Based Index of Game Harmony.[19] Let a_{sj} be the payoff of player s for some given normal form game outcome j (out of m possible outcomes), where the game has n players. The Gini-based index of Game Harmony can be computed in three steps. The first step is *ratio-normalisation* of payoffs, by finding, for each payoff value, $a_{sj}^* = a_{sj} / \sum a_{sj}$, i.e., by dividing each payoff by the sum of all possible payoffs for the player. Ratio-normalisation is essential because, otherwise, the measure would mirror the average payoff distribution of the game, but not the

[19]For more on this see Zizzo (2000a).

extent to which the players' interests in achieving one or another game outcome are the same or are in conflict.

The second step is to compute the payoff distribution index for each possible outcome j, using the Gini Index formula (Gini, 1910), but multiplied by $n/(n-1)$ to ensure boundedness between 0 and 1 with non-negative payoffs: labeling this normalised index as I_j^G for some outcome j, and ordering subjects from "poorest" (i.e., that with the lowest payoff) to "wealthiest" (i.e., that with the highest payoff), we can define I_j^G as:

$$I_j^G = \frac{n}{n-1}\frac{2}{n\sum_{s=1}^{n} a_{sj}^*}\sum_{s=1}^{n} s\left(a_{sj}^* - \frac{\sum_{s=1}^{n} a_{sj}^*}{n}\right) = \frac{1}{n-1}\frac{2}{\sum_{s=1}^{n} a_{sj}^*}\sum_{s=1}^{n} s\left(a_{sj}^* - \frac{\sum_{s=1}^{n} a_{sj}^*}{n}\right)$$

The third step is to find the normalised Gini-based game harmony index GH_G as a simple average of the I_j^G, so $GH_G = (1/m)\sum_{j=1}^{m} I_j^G$. In the case considered in this paper, $n = 2$ and $m = 9$ (since games are 3×3).

References

Abreu, D., and A. Rubinstein (1988): "The Structure of Nash Equilibria in Repeated Games with Finite Automata," *Econometrica*, 56, 1259–1282.

Anderson, J. A. (1998): "Learning Arithmetic with a Neural Network: Seven Times Seven is about Fifty," in *Methods, Models and Conceptual Issues: An Invitation to Cognitive Science, vol. 4*, ed. by D. Scarnborough, and S. Sternberg, pp. 255–299. MIT Press, Cambridge, Mass. and London.

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

Auer, P., M. Herbster, and M. K. Warmuth (1996): "Exponentially Many Local Minima for Single Neurons," in *Advances in Neural Information Processing Systems 8*, ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasslemo, pp. 316–322. MIT Press, Cambridge, Mass. and London.

Bandura, A. (1977): *Social Learning Theory.* Prentice-Hall, Englewood Cliffs.

Cho, I.-K., and T. J. Sargent (1996): "Neural Networks for Encoding and Adapting in Dynamic Economies," in *Handbook of Computational Economics, Volume 1*, ed. by H. M. Amman, D. A. Kendrick, and J. Rust. Elsevier, North Holland.

Clark, A. (1993): *Associative Engines: Connectionism, Concepts, and Representational Change.* MIT Press, Cambridge, Mass.

Elman, J. L., E. A. Bates, M. H. Johnson, A. Karniloff-Smith, D. Parisi, and K. Plunkett (1996): *Rethinking Innateness: A Connectionist Perspective on Development.* MIT Press, Cambridge, Mass.

Fukumizu, K., and S. Amari (2000): "Local Minima and Plateaus in Hierarchical Structures of Multilayer Perceptrons," *Neural Networks*, 13, 317–327.

Garey, M. R., and D. S. Johnson (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Bell Telephone Laboratories, Incorporated, United States of America.

Gilboa, I., and D. Schmeidler (1995): "Case Based Decision Theory," *Quarterly Journal of Economics*, 109, 605–639.

Hornik, K., M. B. Stinchombe, and H. White (1989): "Multi-Layer Feedforward Networks Are Universal Approximators," *Neural Networks*, 2, 359–366.

Hutchins, E., and B. Hazelhurst (1991): "Learning in the Cultural Process," in *Artificial Life II*, ed. by C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, pp. 689–705. Addison-Wesley, Redwood City, California.

Kandori, M., G. J. Mailath, and R. Rob (1993): "Learning, Mutation and Long-Run Equilibria in Games," *Econometrica*, 61, 29–56.

Macleod, P., K. Plunkett, and E. T. Rolls (1998): *Introduction to Connectionist Modelling of Cognitive Processes.* Oxford University Press, Oxford.

Macy, M. (1996): "Natural Selection and Social Learning in Prisoner's Dilemma: Co-adaptation with Genetic Algorithms and Neural Networks," in *Frontiers in Social Dilemma Research*, ed. by W. B. G. Liebrand, and D. M. Messick, pp. 235–265. Verlag, Berlin.

Mailath, G. J. (1998): "Do People Play Nash Equilibrium? Lessons from Evolutionary Game Theory," *Journal of Economic Literature*, 36, 1347–1374.

Nagel, R. (1995): "Unraveling in Guessing Games; An Experimental Study," *American Economic Review*, 85, 1313–1326.

Neyman, A. (1985): "Bounded Rationality Justifies Cooperation in the Finitely Repeated Prisoners' Dilemma Game," *Economics Letters*, 19, 227–229.

Osborne, M. J., and A. Rubinstein (1998): "Games with Procedurally Rational Players," *American Economic Review*, 88, 834–847.

Plunkett, K., and C. Sinha (1992): "Connectionism and Development Theory," *British Journal of Developmental Psychology*, 10, 209–254.

Reilly, R. (1995): "A Connectionist Exploration of the Computational Implications of Embodiement," in *Proceedings of the Swedish Conference on Connectionism.* Lawrence Erlbaum, Hillsdale.

Roth, A. E., and I. Erev (1995): "Learning in Extensive-Form Games: Experimental Data and Simple Dynamic Models in the Intermediate Term," *Games and Economic Behavior*, 8, 164–212.

——— (1998): "Predicting How People Play Games: Reinforcement Learning in Experimental Games with Unique, Mixed Strategy Equilibria," *American Economic Review*, 88, 848–881.

Rubinstein, A. (1993): "On Price Recognition and Computational Complexity in a Monopolistic Model," *Journal of Political Economy*, 101, 473–484.

Rubinstein, A. (1998): *Modeling Bounded Rationality.* MIT Press, Cambridge, Mass. and London.

Rumelhart, D. E., R. J. Hinton, and R. J. Williams (1986): "Learning Representations by Back-Propogating Error," *Nature*, 323, 533–536.

Schmajuk, N. A. (1997): *Animal Learning and Cognition: A Neural Network Approach.* Cambridge University Press, Cambridge.

Sgroi, D. (2000): *Theories of Learning in Economics.* D.Phil. Thesis, University of Oxford.

Simon, H. (1955): "A Behavioral Model of Rational Choice," *Quarterly Journal of Economics*, 69, 99–118.

——— (1959): "Theories of Decision-Making in Economics and Behavioral Science," *American Economic Review*, 49, 253–283.

Smith, E. R. (1996): "What do Connectionism and Social Psychology Offer Each Other?," *Journal of Personality and Social Psychology*, 70, 893–912.

Sontag, E. D. (1995): "Critical Points for Least-Squares Problems Involving Certain Analytic Functions with Applications to Sigmoidal Nets," *Advances in Computational Mathematics*, 5, 245–268.

Sontag, E. D., and H. J. Sussmann (1989): "Backpropagation Can Give Rise to Spurious Local Minima even for Networks with No Hidden Layers," *Complex Systems*, 3, 91–106.

Stahl, D. (1998): "Population Rule Learning in Symmetric Normal-Form Games," Discussion paper, University of Texas at Austin, Center for Applied Research in Economics Working Paper.

Stahl, D. O., and P. W. Wilson (1994): "Experimental Evidence on Players' Models of Other Players," *Journal of Economic Behaviour and Organization*, 25, 309–327.

Sugden, R. (1993): "Thinking as a Team: Toward an Explanation of Nonselfish Behavior," in *Altruism*, ed. by E. F. Paul, F. D. Miller Jr., and J. Paul, pp. 69–89. Cambridge University Press, Cambridge.

Taraban, R., and J. M. Palacios (1994): "Exemplar Models and Weighted Cue Models in Catagory Learning," in *Catagorization by Humans and Machines*, ed. by G. V. Nakamura, R. Taraban, and D. L. Medin, pp. 91–127. Academic Press, San Diego.

Way, E. C. (1997): "Connectionism and Conceptual Structure," *American Behavioral Scientist*, 40, 729–753.

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

Young, H. P. (1993): "The Evolution of Conventions," *Econometrica*, 61(1), 57–84.

Zizzo, D. J. (2000a): "Game Harmony: A Short Note," Discussion paper, Oxford University, unpublished manuscript.

——— (2000b): "Implicit Learning of (Boundedly) Rational Behavior, commentary to K. E. Stanovish and R. F. West, Individual Differences in Reasoning: Implications for the Rationality Debate," *Behavioral and Brain Sciences*, 23.

——— (2000c): *Relativity-Sensitive Behaviour in Economics.* D.Phil. Thesis, University of Oxford.

D. J. Zizzo, Christ Church College, Oxford OX1 1DP, UK (email: daniel.zizzo@economics.ox.ac.uk) & D. Sgroi, Churchill College, Cambridge CB3 0DS, UK (email: daniel.sgroi@econ.cam.ac.uk).

URL: `http://www.chu.cam.ac.uk/~ds10006/home.htm`

FIGURE 1. An Example Neural Network

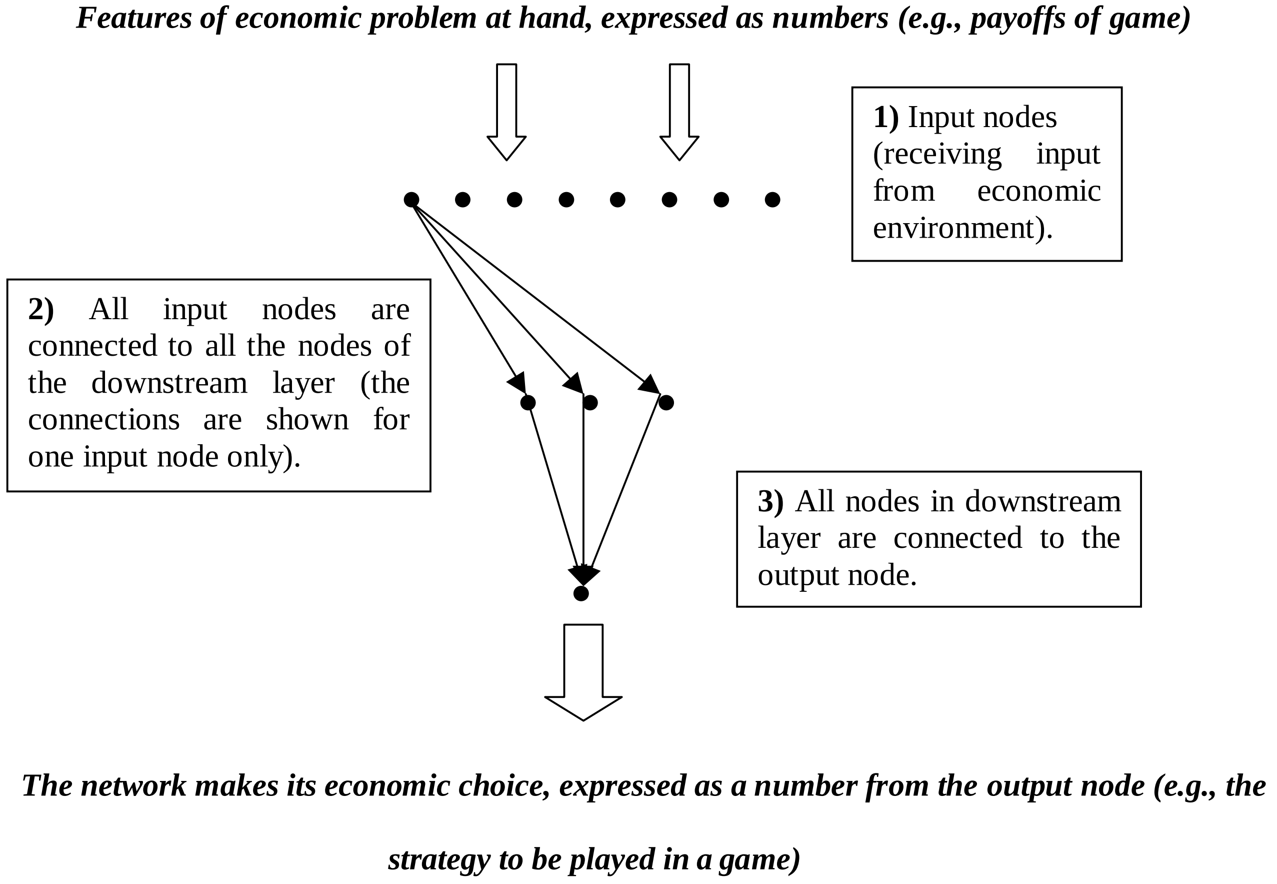

Features of economic problem at hand, expressed as numbers (e.g., payoffs of game)
1) Input nodes (receiving input from economic environment).
2) All input nodes are connected to all the nodes of the downstream layer (the connections are shown for one input node only).
3) All nodes in downstream layer are connected to the output node.
The network makes its economic choice, expressed as a number from the output node (e.g., the strategy to be played in a game)

TABLE 1. Percentage of Correct Answers

	At Least 1 Correct Output			Correct Answer Given		
Convergence	Error tolerance criterion			Error tolerance criterion		
Level γ	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

	At Least 1 Correct Output			Correct Answer Given		
Learning	Error tolerance criterion			Error tolerance criterion		
Rate η	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

	At Least 1 Correct Output			Correct Answer Given		
Momentum	Error tolerance criterion			Error tolerance criterion		
Rate μ	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

Percentage of games solved by the network under different combinations of η, γ and μ. The level of convergence γ simply measures how much 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			Correct Answer Given		
	Error tolerance criterion			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

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. Number and Percentage of Non Answerable Games

Non Nash Algorithm	**Non Answerable Games**	
	Number	**Percentage**
Minmax	5	0.25
Rationalizability	813	40.65
0 Level Strict Dominance	1634	81.7
1 Level Strict Dominance	1205	60.25
Pure Sum of Payoff Dominance	0	0
Maximum Payoff Dominance	8	0.4
Nearest Neighbor	0	0

Non answerable games are games for which a non Nash algorithm does not provide a unique, or any, solution. Percentage is equal to (Number of Non Answerable Games)/2000.

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**		
	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
Partial Overperformers						
0SD	90.83	87.79	80.98	16.62	16.06	14.82
1SD	87.63	84	76.25	34.83	33.39	30.31
Rationalizability	86.44	82.57	74.36	51.3	49.01	44.13
Underperformers						
PSPD	67.71	63.48	55.97	67.71	63.48	55.97
MPD	61.08	57.02	49.95	60.84	56.79	49.75
Minmax	57.75	53.83	46.89	57.6	53.7	46.77
Nearest Neighbor	62.97	58.78	51.48	62.97	58.78	51.48

% of Correct Answers Over Answerable Games = (Number of Correct Answers) / (Number of Answerable Games). 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). 0SD: ‘0-level strict dominance’. 1SD: ‘1-level strict dominance’. PSPD: ‘pure sum of payoff dominance’. MPD: ‘maximum payoff dominance’.

TABLE 5. Tobit Regression on the Average Root Mean Square Error (*continues on next page*)

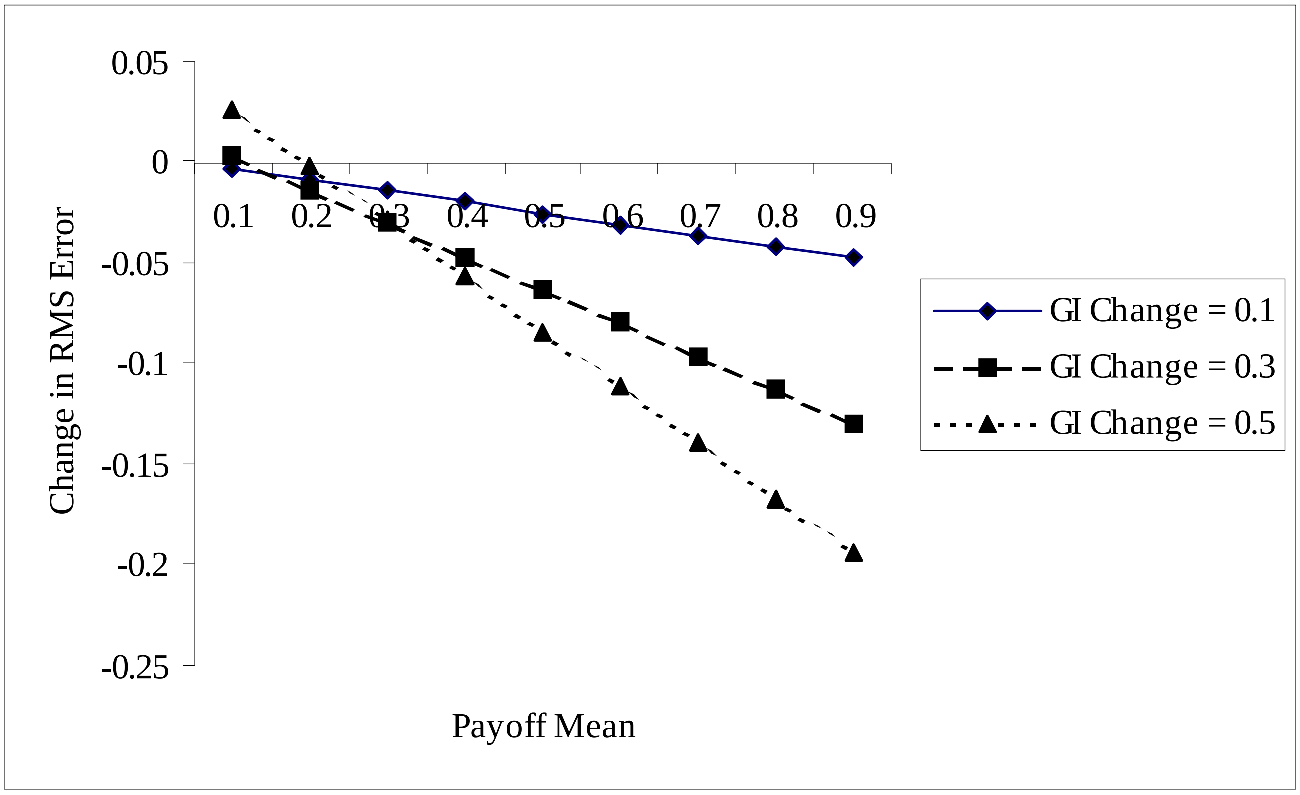
	Model 1				**Model 2**			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**	**Coef.**	**S.E.**	**Prob.**	**Sig.**
MPD Existence	0.037	0.061	0.549					
Minmax Existence	0.067	0.077	0.382					
Same As PSPD	-0.095	0.011	0	****	-0.098	0.01	0	****
Same As MPD	-0.037	0.01	0	****	-0.034	0.01	0	****
Same As Minmax	-0.005	0.011	0.615					
Same As Nearest Neighbour	-0.085	0.008	0	****	-0.085	0.008	0	****
Own NE Payoff	-0.339	0.063	0	****	-0.391	0.029	0	****
Other's NE Payoff	-0.237	0.049	0	****	-0.202	0.034	0	****
Positive Payoff Ratio	-0.025	0.013	0.057	*	-0.018	0.011	0.101	
Negative Payoff Ratio	0.058	0.062	0.35					
Own Generic Temptation	0.104	0.048	0.031	**	0.105	0.048	0.03	**
Other's Generic Temptation	0.317	0.055	0	****	0.317	0.054	0	****
Game Harmony Index	-0.311	0.66	0.638					
Mean	-0.326	0.314	0.3		-0.237	0.208	0.255	
Game Harmony * Mean	-0.667	0.779	0.392		-0.857	0.507	0.091	*
Deviation from Avg. Mean	0.192	0.147	0.19		0.174	0.13	0.181	
Standard Deviation	0.587	0.602	0.33		0.883	0.349	0.012	**
Deviation from Avg. SD	-0.147	0.274	0.591		-0.027	0.018	0.125	
Skewness	-0.03	0.018	0.096	*	-0.032	0.014	0.023	**
Kurtosis	-0.033	0.014	0.02	**	0.056	0.026	0.032	**
Deviation from Avg Skewness	0.055	0.026	0.034	**	0.016	0.014	0.264	
Deviation from Avg Kurtosis	0.019	0.015	0.224					
NE Action 1	0.024	0.01	0.012	**	0.025	0.01	0.01	**
NE Action 2	0.037	0.01	0	****	0.038	0.01	0	****
Strict Dominance: Level 0 Sufficient	-0.079	0.014	0	****	-0.082	0.013	0	****
Strict Dominance: Need for Level 1	-0.058	0.011	0	****	-0.059	0.011	0	****
Strict Dominance: Need for Level 2	-0.049	0.011	0	****	-0.05	0.011	0	****
Max Own Temptation	0.288	0.026	0	****	0.289	0.026	0	****
Min Own Temptation	0.038	0.022	0.084	*	0.039	0.02	0.055	*
Max Other's Temptation	0.142	0.025	0	****	0.143	0.025	0	****
Min Other's Temptation	0.002	0.024	0.946					
Game Harmony * S.D.	1.266	1.565	0.418		0.528	0.817	0.518	
Constant	0.404	0.286	0.158		0.386	0.085	0	****

TABLE 5. (*Continues from previous page*)

	Model 3			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**
MPD Existence				
Minmax Existence				
Same As PSPD	-0.098	0.01	0	****
Same As MPD	-0.034	0.01	0	****
Same As Minmax				
Same As Nearest Neighbour	-0.086	0.008	0	****
Own NE Payoff	-0.392	0.029	0	****
Other's NE Payoff	-0.201	0.034	0	****
Positive Payoff Ratio	-0.017	0.011	0.108	
Negative Payoff Ratio				
Own Generic Temptation	0.108	0.048	0.025	**
Other's Generic Temptation	0.32	0.054	0	****
Game Harmony Index				
Mean	-0.347	0.135	0.01	**
Game Harmony * Mean	-0.551	0.131	0	****
Deviation from Avg. Mean	0.213	0.121	0.077	*
Standard Deviation	1.13	0.201	0	****
Deviation from Avg. SD	-0.026	0.018	0.14	
Skewness	-0.025	0.013	0.05	*
Kurtosis	0.055	0.026	0.035	**
Deviation from Avg Skewness				
Deviation from Avg Kurtosis				
NE Action 1	0.025	0.01	0.011	**
NE Action 2	0.038	0.01	0	****
Strict Dominance: Level 0 Sufficient	-0.081	0.013	0	****
Strict Dominance: Need for Level 1	-0.059	0.011	0	****
Strict Dominance: Need for Level 2	-0.05	0.011	0	****
Max Own Temptation	0.288	0.026	0	****
Min Own Temptation	0.037	0.02	0.063	*
Max Other's Temptation	0.145	0.025	0	****
Min Other's Temptation				
Game Harmony * S.D.				
Constant	0.356	0.082	0	****

Log-Likelihood (Model 1): 662.536. Log-Likelihood (Model 2): 661.275. Log-Likelihood (Model 3): 660.263. LR Test (Model 1→Model 2): $\chi^2(7)=2.52$, P=0.925; LR Test (Model 1→Model 3): $\chi^2(9)=4.55$, P=0.872; LR Test (Model 2→Model 3): $\chi^2(2)=2.02$, P=0.364. Numbers are approximated to the third decimal value. ****, ***, ** and * stand for significance at the 0.001, 0.01, 0.05 and 0.1 levels, respectively.

FIGURE 2. Effect of a Change in Game Harmony on the Root Mean Square Error, for Different Payoff Mean Levels



The figure analyzes the approximate effect of an increase in the Game Harmony Index on the root mean square error, for different mean levels of the game payoff values. RMS Error: Root Mean Square Error. GI Change: change in the Game Harmony Index.

TABLE 6. Ordered Probit Regressions on NDecided1, Games with Multiple PNE, n=5997 (*continues on next page*)

	Model 1				Model 2			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As PSPD	0.549	0.041	0	****	0.548	0.041	0	****
Same As MPD	0.054	0.039	0.169		0.055	0.039	0.159	
Same As Minmax	0.066	0.038	0.078	*	0.067	0.038	0.076	*
Same As 0SD	-0.046	0.103	0.652					
Same As 1SD	0.151	0.103	0.145		0.117	0.072	0.104	
Same As Utilitarian Best	0.217	0.045	0	****	0.217	0.045	0	****
Positive Payoff Ratio	-0.042	0.029	0.151		-0.043	0.028	0.126	
Negative Payoff Ratio	0.23	0.192	0.231		0.228	0.191	0.233	
Own BR Max Temptation	-1.071	0.116	0	****	-1.078	0.113	0	****
Other's BR Max Temptation	-1.044	0.166	0	****	-1.058	0.157	0	****
Own BR Min Temptation	0.01	0.082	0.903		0.01	0.081	0.904	
Other's BR Min Temptation	-0.791	0.103	0	****	-0.797	0.102	0	****
Own BR Payoff	2.072	0.21	0	****	2.063	0.205	0	****
Other's BR Payoff	0.41	0.157	0.009	***	0.4	0.153	0.009	***
Own Max Temptation	-1.499	0.298	0	****	-1.488	0.09	0	****
Other's Max Temptation	-0.464	0.289	0.109		-0.403	0.085	0	****
Own Min Temptation	-0.478	0.099	0	****	-0.475	0.097	0	****
Other's Min Temptation	-0.705	0.089	0	****	-0.704	0.088	0	****
Own Generic Temptation	-2.143	0.336	0	****	-2.158	0.332	0	****
Other's Generic Temptation	-0.325	0.23	0.158		-0.328	0.23	0.153	
Presence of 3NE	0.315	0.083	0	****	0.316	0.082	0	****
Game Harmony Index	0.295	1.348	0.827					
Same As NE When 2NE	0.557	0.067	0	****	0.559	0.067	0	****
Game Harmony x Own BR Temptation	0.036	0.789	0.963					
Game Harmony x Other's BR Temptation	0.172	0.763	0.822					
Mean	6.227	1.258	0	****	5.997	0.669	0	****
Game Harmony x Mean	-0.722	3.156	0.819		0.03	0.439	0.946	
Deviation from Avg. Mean	-0.032	0.511	0.95					
Standard Deviation	-0.216	0.889	0.808					
Deviation from Avg. SD	-0.083	0.795	0.916					
Skewness	-0.076	0.065	0.247		-0.076	0.065	0.239	
Kurtosis	-0.041	0.054	0.449		-0.033	0.043	0.443	
Deviation from Avg Skewness	0.032	0.096	0.738		0.025	0.08	0.75	
Deviation from Avg Kurtosis	0.055	0.055	0.318		0.05	0.05	0.311	
NE Action 1	-0.087	0.037	0.018	**	-0.087	0.037	0.018	**
NE Action 2	-0.229	0.035	0	****	-0.229	0.035	0	****

TABLE 6. Ordered Probit Regressions on NDecided1, Games with Multiple PNE, n=5997 (*continues from previous page)*

	Model 3			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As PSPD	0.559	0.04	0	****
Same As MPD	0.064	0.038	0.095	*
Same As Minmax	0.073	0.037	0.052	*
Same As 0SD				
Same As 1SD	0.115	0.071	0.108	
Same As Utilitarian Best	0.219	0.045	0	****
Positive Payoff Ratio				
Negative Payoff Ratio				
Own BR Max Temptation	-1.061	0.11	0	****
Other's BR Max Temptation	-1.099	0.147	0	****
Own BR Min Temptation				
Other's BR Min Temptation	-0.8	0.099	0	****
Own BR Payoff	1.842	0.096	0	****
Other's BR Payoff	0.565	0.099	0	****
Own Max Temptation	-1.465	0.09	0	****
Other's Max Temptation	-0.403	0.085	0	****
Own Min Temptation	-0.49	0.096	0	****
Other's Min Temptation	-0.732	0.088	0	****
Own Generic Temptation	-1.943	0.277	0	****
Other's Generic Temptation				
Presence of 3NE	0.309	0.082	0	****
Game Harmony Index				
Same As NE When 2NE	0.555	0.064	0	****
Game Harmony x Own BR Temptation				
Game Harmony x Other's BR Temptation				
Mean	5.822	0.481	0	****
Game Harmony x Mean				
Deviation from Avg. Mean				
Standard Deviation				
Deviation from Avg. SD				
Skewness				
Kurtosis				
Deviation from Avg Skewness				
Deviation from Avg Kurtosis				
NE Action 1	-0.087	0.037	0.019	**
NE Action 2	-0.23	0.035	0	****

Log-Likelihood (Model 1): -13880.315. Log-Likelihood (Model 2): -13880.504. Log-Likelihood (Model 3): -13883.739. LR Test (Model 1→Model 2): $\chi^2(7)=0.38$, P=1; LR Test (Model 1→Model 3): $\chi^2(16)=6.85$, P=0.976; LR Test (Model 2→Model 3): $\chi^2(9)=6.47$, P=0.692. White robust estimators of the variance were used to compute standard errors. Numbers are approximated to the third decimal value. ****, ***, ** and * stand for significance at the 0.001, 0.01, 0.05 and 0.1 levels, respectively.

TABLE 7. Ordered Probit Regressions on NDecided2, Games with Multiple PNE, n=5997 (*continues on next page*)

	Model 1				Model 2			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As PSPD	0.664	0.041	0	****	0.668	0.04	0	****
Same As MPD	0.094	0.038	0.014	**	0.099	0.038	0.009	***
Same As Minmax	0.069	0.036	0.06	*	0.078	0.036	0.029	**
Same As 0SD	-0.074	0.105	0.484					
Same As 1SD	-0.02	0.104	0.847					
Same As Pareto	0.264	0.044	0	****	0.264	0.044	0	****
Positive Payoff Ratio	0.001	0.03	0.966					
Negative Payoff Ratio	0.03	0.191	0.876					
Own BR Max Temptation	-1.287	0.116	0	****	-1.257	0.112	0	****
Other's BR Max Temptation	-1.136	0.158	0	****	-1.122	0.147	0	****
Own BR Min Temptation	-0.092	0.079	0.247		-0.079	0.078	0.313	
Other's BR Min Temptation	-0.968	0.099	0	****	-0.97	0.096	0	****
Own BR Payoff	2.221	0.207	0	****	2.22	0.096	0	****
Other's BR Payoff	0.713	0.158	0	****	0.733	0.096	0	****
Own Max Temptation	-1.762	0.286	0	****	-1.629	0.088	0	****
Other's Max Temptation	-0.646	0.27	0.017	**	-0.718	0.083	0	****
Own Min Temptation	-0.532	0.094	0	****	-0.545	0.092	0	****
Other's Min Temptation	-0.647	0.089	0	****	-0.66	0.088	0	****
Own Generic Temptation	-2.307	0.319	0	****	-2.267	0.282	0	****
Other's Generic Temptation	-0.148	0.223	0.508					
Presence of 3NE	0.24	0.078	0.002	***	0.271	0.07	0	****
Game Harmony Index	0.954	1.293	0.461					
Same As NE When 2NE	0.458	0.063	0	****	0.493	0.053	0	****
Game Harmony x Own BR Temptation	0.333	0.767	0.664					
Game Harmony x Other's BR Temptation	-0.189	0.711	0.791					
Mean	6.966	1.209	0	****	6.045	0.545	0	****
Game Harmony x Mean	-1.972	3.057	0.519		0.324	0.393	0.409	
Deviation from Avg. Mean	-0.046	0.502	0.927					
Standard Deviation	0.224	0.853	0.792					
Deviation from Avg. SD	-0.289	0.782	0.712					
Skewness	-0.106	0.064	0.097	*	-0.094	0.062	0.129	
Kurtosis	-0.034	0.052	0.518					
Deviation from Avg Skewness	0.067	0.092	0.469		0.032	0.058	0.576	
Deviation from Avg Kurtosis	0.012	0.052	0.817					
NE Action 1	-0.084	0.036	0.021	**	-0.083	0.036	0.022	**
NE Action 2	-0.159	0.034	0	****	-0.159	0.034	0	****

TABLE 7. Ordered Probit Regressions on NDecided2, Games with Multiple PNE, n=5997 (*continues from previous page*)

	Model 3			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As PSPD	0.664	0.039	0	****
Same As MPD	0.096	0.037	0.01	**
Same As Minmax	0.078	0.036	0.029	**
Same As 0SD				
Same As 1SD				
Same As Pareto	0.263	0.044	0	****
Positive Payoff Ratio				
Negative Payoff Ratio				
Own BR Max Temptation	-1.278	0.111	0	****
Other's BR Max Temptation	-1.147	0.14	0	****
Own BR Min Temptation				
Other's BR Min Temptation	-0.958	0.095	0	****
Own BR Payoff	2.216	0.096	0	****
Other's BR Payoff	0.725	0.095	0	****
Own Max Temptation	-1.616	0.088	0	****
Other's Max Temptation	-0.707	0.082	0	****
Own Min Temptation	-0.555	0.091	0	****
Other's Min Temptation	-0.668	0.087	0	****
Own Generic Temptation	-2.341	0.266	0	****
Other's Generic Temptation				
Presence of 3NE	0.274	0.07	0	****
Game Harmony Index				
Same As NE When 2NE	0.514	0.051	0	****
Game Harmony x Own BR Temptation				
Game Harmony x Other's BR Temptation				
Mean	6.477	0.459	0	****
Game Harmony x Mean				
Deviation from Avg. Mean				
Standard Deviation				
Deviation from Avg. SD				
Skewness				
Kurtosis				
Deviation from Avg Skewness				
Deviation from Avg Kurtosis				
NE Action 1	-0.085	0.036	0.019	**
NE Action 2	-0.159	0.034	0	****

Log-Likelihood (Model 1): -14505.105. Log-Likelihood (Model 2): -14507.422. Log-Likelihood (Model 3): -14510.319. LR Test (Model 1→Model 2): $\chi^2(13)=4.63$, P=0.982; LR Test (Model 1→Model 3): $\chi^2(17)=10.43$, P=0.885; LR Test (Model 2→Model 3): $\chi^2(4)=5.79$, P=0.215. White robust estimators of the variance were used to compute standard errors. Numbers are approximated to the third decimal value. ****, ***, ** and * stand for significance at the 0.001, 0.01, 0.05 and 0.1 levels, respectively.

TABLE 8. Ordered Probit Regressions on NDecided1, Games with a Unique Pure Nash Equilibrium, n=5993 (*continues on next page*)

	Model 1				Model 2			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As NE	1.392	0.059	0	****	1.393	0.059	0	****
Same As PSPD	0.495	0.049	0	****	0.498	0.049	0	****
Same As MPD	0.28	0.04	0	****	0.28	0.04	0	****
Same As Minmax	0.046	0.043	0.288		0.047	0.043	0.274	
Same As Rationalisability	0.08	0.07	0.247		0.082	0.069	0.234	
Same As 0SD	-0.177	0.08	0.028	**	-0.176	0.08	0.028	**
Same As 1SD	0.265	0.075	0	****	0.261	0.074	0	****
Positive Payoff Ratio	-0.085	0.057	0.133		-0.074	0.056	0.191	
Negative Payoff Ratio	-0.181	0.207	0.382		-0.156	0.194	0.421	
Own BR Max Temptation	-1.453	0.101	0	****	-1.398	0.097	0	****
Other's BR Max Temptation	-0.93	0.164	0	****	-0.845	0.158	0	****
Own BR Min Temptation	0.211	0.09	0.019	**	0.226	0.089	0.011	**
Other's BR Min Temptation	-0.301	0.104	0.004	***	-0.285	0.103	0.006	***
Own BR Payoff	2.182	0.343	0	****	2.242	0.325	0	****
Other's BR Payoff	0.012	0.409	0.977					
Own Max Temptation	0.088	0.101	0.384		0.083	0.101	0.413	
Other's Max Temptation	-0.152	0.085	0.071	*	-0.139	0.084	0.099	*
Own Min Temptation	0.145	0.109	0.183		0.116	0.108	0.28	
Other's Min Temptation	-0.078	0.098	0.428		-0.088	0.098	0.369	
Own Generic Temptation	-4.081	0.308	0	****	-4.041	0.303	0	****
Other's Generic Temptation	-1.092	0.195	0	****	-1.104	0.195	0	****
NE Action 1	-0.103	0.037	0.006	***	-0.102	0.038	0.006	***
NE Action 2	-0.136	0.037	0	****	-0.136	0.037	0	****
Game Harmony Index	1.919	1.491	0.198		2.51	0.518	0	****
Game Harmony x Own BR Temptation	-2.057	0.672	0.002	***	-2.055	0.635	0.001	***
Game Harmony x Other's BR Temptation	-0.797	0.952	0.402		-0.638	0.423	0.131	
Mean	6.72	1.3	0	****	7.055	0.661	0	****
Game Harmony x Mean	0.747	3.234	0.817		-0.499	0.387	0.197	
Deviation from Avg. Mean	-0.438	0.556	0.431					
Standard Deviation	1.39	0.887	0.117					
Deviation from Avg. SD	0.112	0.878	0.898					
Skewness	0.105	0.07	0.133		0.125	0.068	0.068	*
Kurtosis	-0.015	0.059	0.799					
Deviation from Avg. Skewness	0.057	0.104	0.586					
Deviation from Avg. Kurtosis	0.016	0.063	0.795					

TABLE 8. Ordered Probit Regressions on NDecided1, Games with a Unique Pure Nash Equilibrium, n=5993 (*continues from previous page*)

	Model 3			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As NE	1.422	0.054	0	****
Same As PSPD	0.538	0.043	0	****
Same As MPD	0.281	0.039	0	****
Same As Minmax				
Same As Rationalisability				
Same As 0SD	-0.162	0.078	0.039	**
Same As 1SD	0.319	0.06	0	****
Positive Payoff Ratio	-0.02	0.038	0.592	
Negative Payoff Ratio	-0.371	0.135	0.006	***
Own BR Max Temptation	-1.342	0.091	0	****
Other's BR Max Temptation	-0.835	0.155	0	****
Own BR Min Temptation	0.255	0.088	0.004	***
Other's BR Min Temptation	-0.247	0.101	0.015	**
Own BR Payoff	2.026	0.285	0	****
Other's BR Payoff				
Own Max Temptation				
Other's Max Temptation	-0.183	0.072	0.011	**
Own Min Temptation				
Other's Min Temptation				
Own Generic Temptation	-3.959	0.285	0	****
Other's Generic Temptation	-1.161	0.193	0	****
NE Action 1	-0.103	0.038	0.006	***
NE Action 2	-0.136	0.037	0	****
Game Harmony Index	2.028	0.409	0	****
Game Harmony x Own BR Temptation	-2.016	0.623	0.001	***
Game Harmony x Other's BR Temptation				
Mean	6.974	0.634	0	****
Game Harmony x Mean				
Deviation from Avg. Mean				
Standard Deviation				
Deviation from Avg. SD				
Skewness	0.125	0.068	0.067	*
Kurtosis				
Deviation from Avg. Skewness				
Deviation from Avg. Kurtosis				

Log-Likelihood (Model 1): -10602.456. Log-Likelihood (Model 2): -10605.244. Log-Likelihood (Model 3): -10610.995. LR Test (Model 1→Model 2): $\chi^2(7)=5.58$, P=0.590; LR Test (Model 1→Model 3): $\chi^2(14)=17.09$, P=0.252; LR Test (Model 2→Model 3): $\chi^2(7)=11.51$, P=0.118. White robust estimators of the variance were used to compute standard errors. Numbers are approximated to the third decimal value. ****, ***, ** and * stand for significance at the 0.001, 0.01, 0.05 and 0.1 levels, respectively.

TABLE 9. Ordered Probit Regressions on NDecided1, Games with Zero Pure Nash Equilibria, n=5983 (*continues on next page*)

	Model 1				Model 2			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As PSPD	0.745	0.038	0	****	0.735	0.035	0	****
Same As MPD	0.579	0.033	0	****	0.58	0.033	0	****
Same As Minmax	-0.022	0.037	0.556					
Positive Payoff Ratio	-0.067	0.058	0.247		-0.081	0.056	0.146	
Negative Payoff Ratio	0.656	0.189	0.001	***	0.748	0.165	0	****
Own BR Max Temptation	-0.552	0.081	0	****	-0.563	0.079	0	****
Other's BR Max Temptation	0.122	0.131	0.355					
Own BR Min Temptation	-0.097	0.099	0.325					
Other's BR Min Temptation	0.295	0.092	0.001	***	0.317	0.088	0	****
Own BR Payoff	2.139	0.317	0	****	2.248	0.296	0	****
Other's BR Payoff	0.354	0.373	0.343					
Own Max Temptation	-0.296	0.073	0	****	-0.294	0.072	0	****
Other's Max Temptation	0.05	0.076	0.509					
Own Min Temptation	0.313	0.072	0	****	0.306	0.072	0	****
Other's Min Temptation	0.299	0.074	0	****	0.317	0.068	0	****
Own Generic Temptation	-1.818	0.2	0	****	-1.846	0.187	0	****
Other's Generic Temptation	-0.195	0.183	0.288		-0.116	0.175	0.506	
NE Action 1	-0.277	0.034	0	****	-0.277	0.034	0	****
NE Action 2	-0.202	0.033	0	****	-0.203	0.033	0	****
Game Harmony Index	2.685	1.257	0.033	**	1.818	0.349	0	****
Game Harmony x Own BR Temptation	0.512	0.586	0.383		0.518	0.586	0.377	
Game Harmony x Other's BR Temptation	-3.006	0.767	0	****	-2.309	0.298	0	****
Mean	0.208	0.927	0.822					
Game Harmony x Mean	-0.851	2.115	0.687					
Deviation from Avg. Mean	-0.303	0.426	0.476					
Standard Deviation	0.124	0.604	0.837					
Deviation from Avg. SD	-0.129	0.791	0.871					
Skewness	-0.041	0.057	0.475					
Kurtosis	0.027	0.051	0.593					
Deviation from Avg. Skewness	-0.02	0.088	0.821					
Deviation from Avg. Kurtosis	0.007	0.055	0.905					

Log-Likelihood (Model 1): -15596.259. Log-Likelihood (Model 2): -15598.954. Log-Likelihood (Model 3): -15600.497. LR Test (Model 1→Model 2): $\chi^2(14)=5.39$, P=0.980; LR Test (Model 1→Model 3): $\chi^2(17)=8.48$, P=0.955; LR Test (Model 2→Model 3): $\chi^2(3)=3.09$, P=0.378. White robust estimators of the variance were used to compute standard errors. Numbers are approximated to the third decimal value. ****, ***, ** and * stand for significance at the 0.001, 0.01, 0.05 and 0.1 levels, respectively.

TABLE 9. Ordered Probit Regressions on NDecided1, Games with Zero Pure Nash Equilibria, n=5983 (*continues from previous page*)

	Model 3			
Explanatory Variables	**Coef.**	**S.E.**	**Prob.**	**Sig.**
Same As PSPD	0.733	0.035	0	****
Same As MPD	0.58	0.033	0	****
Same As Minmax				
Positive Payoff Ratio				
Negative Payoff Ratio	0.663	0.157	0	****
Own BR Max Temptation	-0.56	0.079	0	****
Other's BR Max Temptation				
Own BR Min Temptation				
Other's BR Min Temptation	0.282	0.073	0	****
Own BR Payoff	2.336	0.191	0	****
Other's BR Payoff				
Own Max Temptation	-0.293	0.072	0	****
Other's Max Temptation				
Own Min Temptation	0.308	0.072	0	****
Other's Min Temptation	0.302	0.064	0	****
Own Generic Temptation	-1.865	0.186	0	****
Other's Generic Temptation				
NE Action 1	-0.277	0.034	0	****
NE Action 2	-0.204	0.033	0	****
Game Harmony Index	1.889	0.242	0	****
Game Harmony x Own BR Temptation				
Game Harmony x Other's BR Temptation	-2.119	0.255	0	****
Mean				
Game Harmony x Mean				
Deviation from Avg. Mean				
Standard Deviation				
Deviation from Avg. SD				
Skewness				
Kurtosis				
Deviation from Avg. Skewness				
Deviation from Avg. Kurtosis				

Log-Likelihood (Model 1): -15596.259. Log-Likelihood (Model 2): -15598.954. Log-Likelihood (Model 3): -15600.497. LR Test (Model 1$\rightarrow$Model 2): $\chi^2(14)=5.39$, P=0.980; LR Test (Model 1$\rightarrow$Model 3): $\chi^2(17)=8.48$, P=0.955; LR Test (Model 2$\rightarrow$Model 3): $\chi^2(3)=3.09$, P=0.378. White robust estimators of the variance were used to compute standard errors. Numbers are approximated to the third decimal value. ****, ***, ** and * stand for significance at the 0.001, 0.01, 0.05 and 0.1 levels, respectively.