Learning to Play 3×3 Games: Neural Networks as Bounded-Rational Players
Technical Appendix

Daniel Sgroi[1]
Department of Economics
University of Warwick

Daniel J. Zizzo[2]
School of Economics,
University of East Anglia

December 2007

Abstract

This is a technical appendix designed to support "Learning to Play 3 × 3 Games: Neural Networks as Bounded-Rational Players" by the same authors. In particular, appendix A provides a rigorous definition and description of backpropagation as used in section 2.3, appendix B provides further detail on convergence from section 3.1, and appendix C examines the algorithms from section 3.2.

Appendix A

Before defining backpropagation, we begin with the error function. Following Rumelhart et al. (1986) we use a function of the form

$$\triangle w_{ij} = -\eta \frac{\partial \varepsilon}{\partial w_{ij}} = \eta k_{ip} o_{j\rho} \tag{1}$$

where w_{ij} is the weight of the connection between the sending node j and receiving node i. As ε is the neural network's error, $\partial\varepsilon/\partial w_{ij}$ measures 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 $\partial\varepsilon/\partial w_{ij} = -k_{i\rho} o_{j\rho}$ where $o_{j\rho}$ is the degree of activation (the actual output) of the sender node j for a given set of inputs ρ. The higher is $o_{j\rho}$, the more the sending node is at fault for the erroneous output, so it is this node we wish to correct more. $k_{i\rho}$ is the error on unit i for a given set of inputs ρ, multiplied by the derivative of the output node's activation function given

[1]Address: Faculty of Economics, Austin Robinson Building, Sidgwick Avenue, Cambridge CB3 9DD, UK. Email: daniel.sgroi@econ.cam.ac.uk. Telephone: +44 1223 335244. Fax: +44 1223 335299.
[2]Address: School of Economics, University of East Anglia, Norwich, NR4 7TJ, UK. Email: d.zizzo@uea.ac.uk.

its input. Calling $g_{i\rho}$ the goal activation level (the target "correct" output) of node i for a given set of inputs ρ, in the case of the output nodes $k_{i\rho}$ can be computed as

$$k_{i\rho} = (g_{i\rho} - o_{i\rho})f'(t_{i\rho}) = (g_{i\rho} - o_{i\rho})o_{i\rho}(1 - o_{i\rho}) \tag{2}$$

since the first derivative $f'(t_{i\rho})$ of the receiving node i in response to the set of inputs ρ is equal to $o_{i\rho}(1 - o_{i\rho})$ 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.

Using backpropagation, we first compute the error of the output layer (layer N) using equation 2, and update the weights of the connections between layer N and $N - 1$, using equation 1. 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, ρ the current set of inputs and β an index for each node of layer N (activated by i), we can use:

$$k_{i\rho} = f'(t_{i\rho}) \sum_{\beta} k_{\beta\rho} w_{\beta i} \tag{3}$$

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

$$\triangle w_{ij}^{t} = -\eta \frac{\partial \varepsilon^{t}}{\partial w_{ij}^{t}} + \mu \triangle w_{ij}^{t-1} \tag{4}$$

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).

Appendix B

The following table contains details on the convergence of C trained with different random seeds and achieving various levels of convergence for different learning and momentum rates. Only with a combination of high η and μ ($\eta = 0.5$ and $\mu = 0.9$) was convergence not achieved, even at $\gamma = 0.1$. A high momentum rate was particularly detrimental. A low learning rate ($\eta = 0.1$) slowed learning, particularly with $\mu \leq 0.3$, but did not prevent C from converging at $\gamma = 0.1$ or better. A maximum of 600000 rounds was used to try to achieve convergence. In practice, though, $\gamma = 0.1$ convergence was usually reached in about 100000-200000 rounds.

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

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

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

Table 5: Number of C Converged

Appendix C

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

Consider the game G and the trained neural network player C^*. Index the neural network by i and the other player by j. The neural network's minmax value (or reservation utility) is defined as $r_i = \min_{a_j} [\max_{a_i} \pi_i (a_i, a_j)]$. The payoff r_i is literally the lowest payoff player j can hold the network to by any choice of $a \in A$, provided that the network correctly foresees a_j and plays a best response to it. Minmax therefore requires a particular brand of pessimism to have been developed during the network's training on Nash equilibria.

Rationalizability is widely considered a weaker solution concept compared to Nash equilibrium, in the sense that every Nash equilibrium is rationalizable, though every rationalizable equilibrium need not be a Nash equilibrium. Rationalizable sets and the set which survives the iterated deletion of strictly dominated strategies are equivalent in two player games: call this set S_i^n for player i after n stages of deletion. To give a simple intuitive definition, S_i^n is the set of player i's strategies that are not strictly dominated when players $j \neq i$ are constrained to play strategies in S_j^{n-1}. So the network will delete strictly dominated strategies and will assume other players will do the same, and this may reduce the available set of strategies to be less than the total set of actions for i, resulting in a subset $S_i^n \subseteq A_i$. Since we are dealing with only three possible strategies in our game G, the subset can be adequately described as $S_i^2 \subseteq A_i$ with player j restricted to $S_j^1 \subseteq A_i$.

The algorithm 0SD checks whether all payoffs for the neural network (the row player) from playing an action are strictly higher than those of the other actions, so no restriction is applied to the action of player $j \neq i$, and player i's actions are chosen from $S_i^0 \subseteq A_i$. 1SD allows a single

level of iteration in the deletion of strictly dominated strategies: the row player thinks that the column player follows 0SD, so chooses from $S_j^0 \subseteq A_j$, and player i's action set is restricted to $S_i^1 \subseteq A_i$. Both of these algorithms can be viewed as weakened, less computationally demanding versions of iterated deletion.

L1 and MPD are different ways of formalizing the idea that the agent might try to go for the largest payoffs, virtually independently of strategic considerations. If following MPD, C^* simply learns to spot the highest conceivable payoff for itself, and picks the corresponding row, hoping the other player will pick the corresponding column. In the case of L1, an action $a_i = a_{L1}$ is chosen by the row player according to $a_{L1} = \arg\max_{a_i \in A_i} \{a_i \mid a_j = \triangle_j\}$, where $\triangle_j$ is defined as a perfect mix over all available strategies in A_j. Put simply, a_{L1} is the strategy which picks a row by calculating the payoff from each row, based on the assumption that player j will randomly select each column with probability $\frac{1}{3}$, and then chooses the row with the highest payoff calculated in this way. It corresponds to the definition of 'level 1 thinking' in Stahl and Wilson (1995).

A player that assumes that his opponent is a level 1 thinker will best-respond to L1 play, and hence engage in what, following Stahl and Wilson (1994) and Costa Gomes et al. (2001), we can label L2 (for 'level 2') play.

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

Since, by construction, all games in the training set have a unique pure strategy Nash equilibrium, we are virtually guaranteed to find a NNG solution for all games in the testing set. The only potential (but unlikely) exception is if the nearest neighbor is not unique, because of two (or more) games having exactly the same sum of squares. The exception never held with our game sample. Clearly, a unique solution, or indeed any solution, may not exist with 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.

References

Costa Gomes M, Crawford VP, Broseta B, 2001. Cognition and behavior in normal-form games: an experimental study. Econometrica 69, 1193-1235.

Rumelhart DE, Hinton RJ, Williams RJ, 1986. Learning representations by backpropagating error. Nature 323, 533-536.

Stahl, DO, Wilson PW, 1994. Experimental evidence on players' models of other players. Journal of Economic Behavior and Organization 25, 309-327.

Stahl, DO, Wilson PW, 1995. On players' models of other players: theory and experimental evidence. Games and Economic Behavior 10, 218-254.