Machine Learning and Mixed Strategy Games

Edmund Lam, Avi Patni

1 Introduction

1.1 Mixed Strategy Games

A Mixed Strategy Game within this paper refers to a subset of strategic games within game theory. A game is defined to have P players and M moves per player. Each player can play any one of their M moves every round, and scores for each player are calculated by accessing the payoff matrix with the results of all their opponents. The payoff matrix is represented by a P-dimensional hypercube matrix with side length M. Players can be conceptualized as being arranged in a circle, each player is the first in their perspective, which increments for player to their right. Thus, player P_i's' ($1 \leq i \leq P$) score is calculated by accessing the payoff matrix with the coordinates $(P_i, P_{i+1}, \cdots, P_P, P_1, P_2, \cdots, P_{i-1})$. Figure 1 shows an example of how payoffs could be calculated for each player. Players 1 and 2 make moves P_1 and P_2, respectively. Thus, the players will recieve scores $M_{<1,2>} = 3$ and $M_{<2,1>} = 3$, respectively.

Figure 1: Example 2×2 payoff matrix with sample moves and resulting scores.

Payoff Matrix	(P_1, P_2)	Payoffs
$M = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix}$	$(1, 2)$	$(1, 3)$

1.2 Neural Networks

An artificial Neural Network (NN) is a computing system loosely inspired by the biological neural networks found within animal brains. A neural network is composed of a number of interconnected groups of nodes, such that each node can transmit signals to other nodes within the Neural Network. This can be most simply represented by layers, each with a number of nodes, where two adjacent layers of nodes can be represented as a complete bipartite graph. Each connection between two nodes represents a weight $w_{<i,j>}$

Given layers x (length X) and y (length Y), node $x_i (1 \leq i \leq X)$ will be connected to node $y_j (1 \leq j \leq Y)$ with a weight $w_{<i,j>}$. This represents the weight held by x_i on the resulting value of y_j, which is calculated by summing each node x_i by the weight of its connection, $w_{<i,j>}$, which can be represented using the following formula, which will be run for every node y_j in layer y:

$$y_j = \sum_{i=1}^{X} x_i \cdot w_{<i,j>}$$

A set of weights w can thus be represented as an $X \times Y$ matrix, where $w_{<i,j>}$ still represents the weight between x_i and y_j. w can thus be multiplied by the first layer $x(1 \times X)$ to calculate the next layer, $y(1 \times Y)$. Furthermore, an added vector b of dimensions $1 \times Y$ is added, which will be added to y after it is calculated. This allows the vector y to be skewed, and results in the the simple formula $y = x \cdot w + b$, where w represents the matrix of weights between x and y. For neural networks with more than two layers, this is done sequentially to calculate each layer in the network. The first and last layers are vectors that determine the input information, and the output, respectively.

Our project utilizes artificial neural networks and natural selection to optimize strategies for mixed strategy games and maximize individual returns. By providing neural networks solely with information on the moves of each player for the last N rounds, and using natural selection based on the resultant score after a number of rounds with different players, we can observe the strategies emerging in the best neural networks, and test how they react to certain situations. We were particularly keen on observing more complicated payoff matrices, such as when $P > 2$, $M > 2$, as well as games similar to the Prisoner's Dilemma, where nash equilibrium exists, but is not necessarily optimal in the long term.

2 Method

2.1 Design

As defined above, a Mixed Strategy Game consists of three main parameters: the number of players P, number of possible moves M, and G, a P-dimensional hypercube payout matrix with sidelength M. One round of this game consists of each player independently making their move, and having payouts for all players calculated as defined in section 1.1. The program is given an input consisting of P, M, and the payoff matrix G, which is read along with parameters defining the total population of Neural Networks, the number of Generations (as defined in section 2.1.1), and a number of simple agents, which simply choose a random move each round, defined by a random Dirichlet distribution, where vector α (of length M) can be defined as $\alpha_i = \frac{1}{r_i}$ for all $1 \leq i \leq M$, where r is a set of uniformly random numbers such that $r_i \in [0, 1)$. These simple agents can be chosen in place of real neural networks to play as opponents of networks, which introduces a more diverse distribution of actions. This diversity forces the neural networks to learn to adapt to different situations, helping prevent situations where all remaining neural networks essentially fall into the same move loop, and only experience a small number of the possible memory situations, which leads to knowledge gaps.

2.1.1 Generations

Within one generation, each neural net plays a number of games N. Each game N is created by choosing P random players from the combined set of neural nets and simple agents, with replacement (this means that it is possible for a neural net to play itself, and that some neural nets will participate in more games than others). Each game will last a number of rounds R, where each round all agents will make a decision and scores will be calculated for each. Finally, the total scores for each neural network will be divided by the number of rounds they played, resulting in a final scoring representing their average score per round.

After average scores for a full generation is calculated, these scores are cubed, and used in a weighted random choice, where the weight for each neural network is the cubed score. Half of the neural networks are chosen using this weighted random choice (without replacement) and the surviving neural networks will reproduce one time, whereas the remaining neural networks are deleted. In this experiment, Neural Nets reproduce asexually, such that each weight is duplicated, but with a chance of mutation. The mutation parameters are as follows:

$$\text{Description} \quad \text{Probability} \tag{1}$$
$$\text{Create completely random agent} = 0.01 \tag{2}$$
$$\text{Completely randomize weight or bias} = 0.0005 \tag{3}$$
$$\text{Multiply weight/bias by normal distribution } (\mu = 1) = 0.001 \tag{4}$$
$$\sigma \text{ of the normal distribution used in (4)} = 0.001 \tag{5}$$

Afterwards, the next generation is run on these new neural networks, consisting of the survivors from the last generation and the new cloned networks. This process repeats for a specified number of generations. Thus, as survival of the fittest is employed, the neural networks are expected to improve their strategies at the game over time.

2.1.2 Neural Networks

Each neural network used will consist of three layers, an input, a hidden, and an output layer. The hidden layer is usually of length 10, though is sometimes 5 for earlier and simpler input games. The output layer is of length M, where each output value represents the probability that the network wants that move chosen. Thus, a random weighted choice is run with a probability distribution using this output vector, and the chosen move is selected in this manner. While another valid method is to choose the highest value as the chosen move, the use of probabilities allows neural networks to specify a distribution of moves and thus a mixed strategy for each memory state, rather than a fixed one.

The input layer for each Neural Network is of size $M \cdot P \cdot l$, where the neural networks remembers all moves in the last l games (length of its memory). Each value is either a 1 or 0, where 1 indicates that the move was chosen. For example, a single game with 2 players and 2 moves can be represented with a vector of length 4, where

$$[1, 0, 0, 1]$$

indicates that player 1 (the neural network itself) played move 1, whereas the opponent played move 2. This is duplicated l times, where the rightmost chunk of length $P \cdot M$ is the most recent game. If memory is empty before a certain point, which occurs when l is greater than the total number of rounds played so far, all nodes will be a 0, as no moves have been chosen.

2.2 Procedure

A number of random neural networks are generated, which is specified by the total population of neural netowrks. This will be held constant across all generations. All weights and biases within this neural network are created using a normally distributed random number generator, with $\mu = 0$ and $\sigma = 1$. As defined in section **??**, these neural networks will compete with one another (and the simple agents) for a number of generations (5000 in most runs), and thus improve through the natural selection process. All networks are saved, along with statistics on their average point totals, and how often they played each move. Input datasets are specifically designed to challenge neural networks and look for human-like behavior.

3 Results

4 Discussion