Can Deep Reinforcement Learning Solve Misère Combinatorial Games?

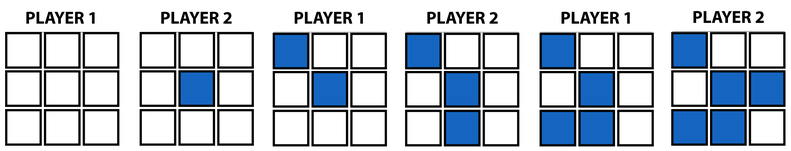
Abraham Oliver, Brown County High School

1 Introduction

A central goal of machine learning has been to train models *tabula rasa*, i.e. from a blank slate with no human guidance. Progress towards this goal has accelerated with DeepMind's AlphaGo Zero's win over the previous version of AlphaGo which defeated the Go world champion in 2016 and their models' wins over the strongest computer opponents in Chess and Shogi[1]. The training algorithm proposed requires no human-crafted features or human-generated game data and is general enough to be applied to games other than those tested. Many combinatorial games (discrete zero-sum perfect information games) can be "solved" mathematically[2]. However, the misère versions of these games — in which the winner by the regular game's rules loses — are much more difficult to analyze and thus possibly harder for computers to play well. Better understandings of these games could lead to advances in combinatorics, optimization, complexity theory, and even gene folding.

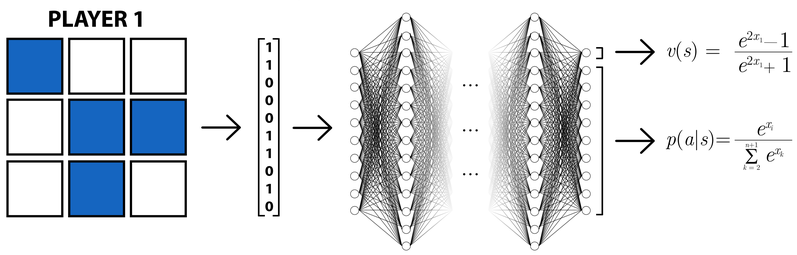
2 Misère Tic-Tac-Toe

Misère tic-tac-toe ("Notakto") is like tic-tac-toe played on an n x n board, except that both players place the same symbol, and the player that completes a row, column, or diagonal loses. For certain n, the game is "weakly solved" -- it is known which player can win no matter the actions of the opponent. For a 3x3 board, player one can always force a win[3], while for 4x4, player two can always win. By computer serach, we know which player can win (called the game theoretic value) for certain n, but not the winning strategy: player one wins for 5x5, player one wins for 6x6, and player two wins all boards with side lengths divisible by four[4]. For example, in the sample below, player one uses a winning strategy for the 3x3 game: playing in the middle on the first turn, and subsequently choosing empty spaces either over one and up/down two or over two and up/down one from the opponent's move (like a knight in chess). Player one wins because any player two move results in a three-in-a-row.



3 Deep-Q Model

To model the decisions of a player (the "policy", denoted π) and the likelihood that the player will win (the "value"), we use a deep neural network, a system of addition and multiplication nodes ("neurons") that abstractly model the functioning of a human brain. Like Silver et al[5], we use a neural network with parameters θ to produce a policy and value for a given state s such that $(p,\ v)=f_\theta(s)$. v is the value of the game at the given state, represented by a scalar in $[-1,\ 1]$ calculated by applying the hyperbolic tangent function to first output neuron. p is the policy, represented by a discrete probability distribution where $\pi(a|s)=p_a$ calculated by applying the softmax normalization function to the remaining outputs. The network consists of randomly-initialized fully-connected layers each followed by the Relu activation function. Training data for the model is stored as a set of tuples $\{s,\ \pi_{target},\ z\}$ where π_{target} is a desired policy generated by the process in Section 6 and $z=\pm1$ is the eventual winner from the perspective of the current player. Old data points are discarded with a memory replay queue as new ones are generated. During training, random batches of the memory replay are passed to the network which is updated with gradient descent minimizing the loss function defined by Silver at al: $l = \Sigma[(z\ -\ v)^2\ -\ \log p^{\mathrm{T}}\pi_{target}]$ over all data points in the batch.



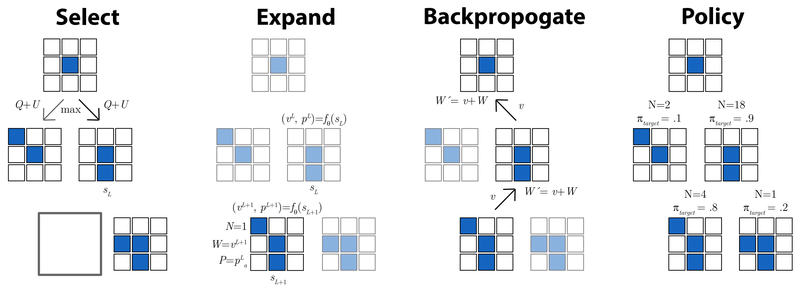
4 Hypothesis

With limited computational resources, we will not be able to train a perfect model for board sizes five and greater but we will be able to train perfect models such that player one can win 100% of games on 3x3 boards and player two can win 100% of games on 4x4 boards.

5 Training Algorithm

Designed by Silver et al, the algorithm generates target policies for a given move by playing games against itself guided by a modified Monte-Carlo Tree Search[5]. Each state s is the root node of a tree with child nodes representing the state after a given action and where each node contains a set of statistics $\{N,\ W,\ P\}$. $N(s,\ a)$ is the visit count for a given node, $W(s,\ a)$ is the total value of all runs that traversed the node, and $P(s,\ a)$ is the probability of selecting an action according to the model. The algorithm proceeds by: **(1) Selection** Starting from a given root node s_t , choose an action $a_t = \mathrm{argmax}_a(Q(s_t,\ a)+U(s_t,\ a,\ \varepsilon))$ (where ε is an hyperparameter) until a node is reached whose possible actions have not all been evaluated. Call this node s_L. **(2) Expand and Evaluate** Randomly choose an un-explored action, play the action to progress to s_{L+1}, and evaluate the value and action probabilities with $(v,\ p)=f_\theta(s_{L+1})$. Initialize a new child with $\{N = 1,\ W = v,\ P = P(s_L|a)\}$. **(3) Backpropogate** Pass v up the tree. For each node traversed, update the node's statistics by $\{N'=1+N,\ W'=v+W,\ P'=P\}$. **(4) Repeat** Repeat the process for a set number of simulations. **(5) Calculate Policy** Starting with a blank board s_0 , run steps (1) to (4). Compute the target policy $\pi_{target}(s_0,\ \tau)$ (where τ is a hyperparameter) and choose an action a_1 from this policy . Play this action and begin a new tree with root node s_1. Repeat this process until a terminal state. Save the data from this game as $\{s_t,\ \pi_{target}(s_t),\ z\}$ where z is the eventual winner. Train the neural network over these data points and run further self-play games with the newly trained network.

$$Q(s,a) = \frac{W(s,a)}{N(s,a)} \qquad U(s_t,a,\varepsilon) = \varepsilon P(s_t,a)\ \frac{\sqrt{\sum_{b\in A(s_t)} N(s_t,b)}}{1+N(s_t,a)} \qquad \pi_{target}(a|s_t,\tau) = \frac{N(s_t,a)^{1/\tau}}{\sum_{b\in A(s_t)} N(s_t,b)^{1/\tau}}$$



Future Work

Performing tree searches to determine the game theoretic values for games with large search spaces, like large misère tic-tac-toe games or Go, would take thousands of years with current technology. The models in this work could reduce this time by orders of magnitude by "tree pruning" in which the search can prioritize certain branches of the tree from the model's suggestions. We hope to use this method to find the guaranteed winners and possibly the winning strategies of large misère tic-tac-toe games. Definitively solving these games could lead to understanding in other areas in which combinatorial games naturally occur as complexity theory and optimization. In terms of Silver et al's algorithm's effectiveness against misère games, nothing can be definitively concluded without conducting experiments with much greater computational power. Also, it would be worth testing different neural network architectures such as convolutional neural networks, residual networks, and capsule networks to check the effectiveness of the algorthim versus the effectiveness of the model being trained.

6 Results

All models trained had a memory queue size of 300 states and trained over 10 epochs of 100 state replays separated into batches of size 10 with a learning rate of .001. For the tests recorded, model hyperparameters were chosen randomly from a set of candidates and run for a set number of training iterations. The "score" of a model is the number of games out of 100 that the best version of the model won in a tournament against a randomly playing opponent. The "iteration" is the length of the training period over which the score was choosen. If a model achieved a perfect score, the iteration at which it achieved this is presented. The "simulations" is the number of simulations per move used in step (4) of the training algorithm.

Game	Hidden Layers	Simulations	Score	Iterations
3x3	3 x 10 neurons	300	25	100
	4 x 10 neurons	200	28	100
	4 x 10 neurons	500	32	100
	None	20	73	100
	3 x 100 neurons	50	81	100
	1 x 1000 neurons	500	85	100
	None	50	100	25
	None	500	100	25
	1 x 100 neurons	100	100	20
	3 x 100 neurons	2000	100	14
4x4	[1000, 1000, 200]	2000	31	500
	[1000, 500, 200]	2500	38	500
	[1000, 200, 200]	3000	41	500
	[1000, 500, 200]	3500	44	500
	[1000, 500, 200]	2500	46	500

Figure 1 Best 3x3 Models

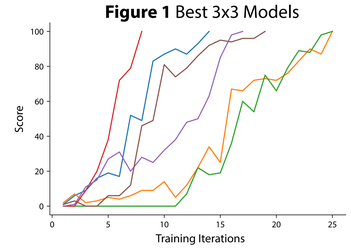


Figure 2 Best 4x4 Model

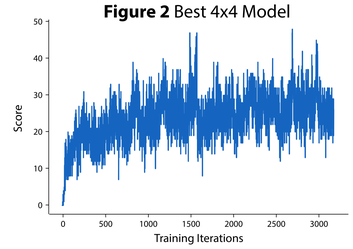


Figure 3 Imperfect 3x3 Models

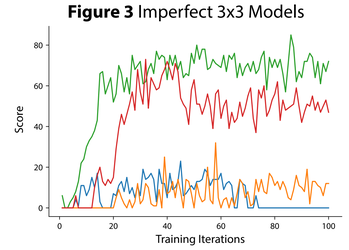


Figure 1 The learning curves of a selection of the best 3x3 models above.

Figure 2 The best 4x4 model we trained, even using hyperparameter tuning, only achieved a score of 48. The model trained over 3183 iterations with 2000 simulations per move.

Figure 3 The learning curves of 3x3 models that did not achieve a score of 100 within 100 iterations.

7 Analysis

The number of neurons and layers seems not to significantly impact score or required number of training iterations, which is surprising because higher capacity models should have greater capacity to model data. Some models trained within seconds, suggesting that the algorithm would be able to easily scale to larger boards, but this was not the case. Although a 4x4 board has only 128 times as many possible positions as 3x3, our best model only won 46% of games even after training 30,000 times longer, and the trend did not suggest that it would win more games given more time (Figure 2). All other 4x4 models were unable even to achieve a 40% win rate. The training performed by Silver et al with massive computing resources involved over 4.9 million generated games each using 16,000 simulations per game[5]; our difficulty with the 4x4 board may be due to insufficient training period with too weak computing power.

All figures, diagrams, and charts were created by the student.

[1] arXiv:1712.01815 [cs.AI]

[2] Heule, M. J., & Rothkrantz, L. J. (2007). Solving games: Dependence of applicable solving procedures. Science of Computer Programming, 67(1), 105-124.

[3] Timothy Y. Chow, Online mathoverflow.net question, "Neutral tic tac toe", 16 May 2010. http://mathoverflow.net/questions/24693/neutral-tic-tac-toe/

[4] Crandall, David, personal communication, August 23, 2017.

[5] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Chen, Y. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354.