AMOD 5310H Artificial Intelligence Term Project Proposal

Mastering the Game *No Thanks!* with Deep Neural Networks and Tree Search

Smars Hu, Trung Kien Ngo, Lya Wang

March 12, 2025

Abstract

We propose an AI gaming bot for *No Thanks!*, a multiplayer, stochastic card game. Our bot employs a model-based reinforcement learning approach inspired by AlphaZero, combining Monte Carlo Tree Search (MCTS) with deep neural networks. The neural network estimates both the policy and value function, while MCTS generates self-play data to refine training. The neural network is then iteratively updated based on MCTS playouts. To evaluate performance, we plan to test the bot against a (mixed) group of pure online MCTS agents, rule-based agents, and random agents.
Keywords: model-based reinforcement learning; MCTS; AlphaZero; gaming bot.

1 Introduction

The Game *No Thanks!*

No Thanks! is a multiplayer, turn-based card game centered around risk and resource management. The game consists of a deck of 33 numbered cards ranging from 3 to 35, along with a set of coins (each is worth -1) used as currency for passing on undesirable cards. The objective is to accumulate the lowest possible score by strategically taking or avoiding cards.

At the beginning of the game, each player is given a fixed number of coins, typically 11 in a standard 3- to 5-player game (and 9 for 6-player game, 7 for 7-player game). To avoid players from perfectly predicting the remaining cards, nine random cards are removed from the deck before play begins. The deck of 24 cards is then shuffled randomly.

In each turn, the first card in the deck is revealed, and the active player must decide whether to take the card or pass it with a coin. If they choose to take the card, they add its face value to their total score and collect any coins that have accumulated on it, after which the next card is revealed. If they prefer to avoid the card, they must place one of their coin on it, after which the turn passes to the next player. This process continues until a player eventually claims the card, either voluntarily or because she has run out of coins. The game continues until all cards in deck have been taken.

Scoring is determined at the end of the game, where each player's total score is calculated based on the values of the cards they have collected. Notably, consecutive numbered cards in a player's possession count only for the lowest-numbered card in the sequence, introducing more potential strategies by reducing expected penalty for taking large-valued cards. Additionally, each coin remaining at the end of the game reduces the player's score by one point. Since players must decide whether to accumulate coins at the cost of taking high-value cards, the game presents an interesting trade-off between immediate risk and long-run gain.

By design, *No Thanks!* is a stochastic game with perfect information, as the entire game state (i.e., the card in play, available coins, each player's collected cards, remaining coins, and the current player) is fully observable to all players. Stochasticity arises from the random removal of nine cards and the shuffling of the deck.

The game is Markovian under the optimal strategy, as the current player's decision depends solely on the present state. However, if opponents play sub-optimally, their past behavior (e.g., risk tolerance, bluffing tendencies, or willingness to take high-value cards) might provide useful information for decision-making. In such cases, a strong strategy could incorporate opponent modeling, which inherently depends on past observations, making the game non-Markovian.

The AlphaZero Algorithm

AlphaZero (cf. [1]) is a generalized reinforcement learning algorithm extending AlphaGo Zero (cf. [2]), originally designed for mastering combinatorial games such as Go, chess, and shogi, where it has demonstrated superhuman capabilities. Unlike its predecessor AlphaGo (cf. [3]), which relied on prior knowledge gathered from human expert data, AlphaZero is trained solely through self-play reinforcement learning (i.e., *tabula rasa* reinforcement learning).

A key improvement in AlphaZero is the use of a single deep network to estimate both the policy and value functions simultaneously, rather than maintaining separate networks. The algorithm employs Monte Carlo Tree Search (MCTS, cf. [4, pp. 207-210] and [5]) guided by the policy-value network to perform self-play. The network is then iteratively updated based on self-play data, with each iteration refining move selection and evaluation.

More recently, AlphaZero has been extended to multiplayer games (e.g., [6]) and stochastic games (also known as Stochastic MuZero; see [7]). AlphaZero can handle stochastic games when provided with a perfect simulator that follows fixed probability distributions. Since the probabilities in *No Thanks!* are known to all players, we adopt the AlphaZero approach for our bot. Further details on the methodology are provided in Section 2.

Contribution

We develop an AI gaming bot for *No Thanks!* using an AlphaZero-like reinforcement learning approach that combines Monte Carlo Tree Search (MCTS) with a deep neural network that simultaneously estimates the policy and value functions. We evaluate the bot's performance against a group of pure online MCTS agents, rule-based agents, and random agents. Particularly, to the best of our knowledge, there has been no such algorithm applied to the *No Thanks!* game. This work extends the applicability of AlphaZero-like algorithms to multiplayer, stochastic card games.

2 Methodology

2.1 Modeling the Game

We represent the state of a n-player game can by $s = (M, b)$ where M is a $35 \times n$ matrix containing information on each player's cards in hand, coins in hand, and who's the current player; b is a two-dimensional vector containing the card in play, and coins in play. More specifically, the first 33 columns of M contains binary values (0 or 1), if $M_{k,j} = 1$ for $j = 1, 2, ..., 33$, then card $j + 2$ is in Player k's hand. Column 34 of M contains the number of coins for each player; namely, $M_{k,34} =$ the number of coins Player k has. The last column of M is binary, representing who is the current player; i.e., $M_{k,34} = 1$ if it is Player k's turn, and $M_{k,34} = 0$ otherwise. All other information, e.g., players' scores, the number of remaining cards in deck, can be revealed given (M, b). Note that we will transform M into a three-dimensional array of $33 \times n \times 3$, and add a redundant term (the number of remaining cards) to b when they are treated as the input of the neural network (Figure 1).

The state is fully observable to all players. Suppose all players share the same policy $\pi = (p, 1 - p)$ where p denotes the probability of taking the card in play, then each player faces a Markov decision problem given the current state s.

Denote $(q, \vec{v}) = f_\theta(s)$ as the policy-value neural network parameterized by θ. The vector $\vec{v}$ is n-dimensional, representing the value for each player. There is no intermediate rewards, and at a terminal state s_T, players gain their final rewards $\vec{z}$ that is determined by their ranking. Foe example, in a five-player game, $z_k = 1, 0.5, 0, -0.5, -1$ for Player k being rank one to five, respectively.

2.2 MCTS Process

In our algorithm, a node in the searching tree is an alias of a state, as the game is Markovian. Indeed, we keep track of *edges* instead of nodes as in classic tree search algorithms. Each edge (s, a), where $a \in \mathcal{A}(s)$, stores a set of statistics

$$\{N(s, a), W(s, a), Q(s, a), P(s, a)\};$$

where $N(s, a)$ is the visit count, $W(s, a)$ is the total value, $Q(s, a)$ is the average value, $P(s, a)$ is the prior probability (policy) of selecting that edge.

Given a random initialization of θ, in each iteration, the guided Monte Carlo tree generates a playout with state trajectories $(s_0, s_1, ..., s_T)$ where s_T is a terminal state (i.e., leaf node).

Selection At each state, an action is chosen using a variant of the PUCT algorithm (cf. [8], [3]) such that

$$a_t^* = \arg\max_a \{Q(s_t, a) + U(s_t, a)\}; \tag{1}$$

where

$$U(s, a) = c_{puct} P(s, a) \frac{\sqrt{\sum_\alpha N(s, \alpha)}}{1 + N(s, a)}; \tag{2}$$

where $P(s, a) = p$ if $a = 0$ (take) and $P(s, a) = 1 - p$ if $a = 1$ (pass) for $(p, v) = f_\theta(s)$, c_{puct} is a hyperparameter determining the level of exploration;

Expansion and Evaluation We expand each node through edges until a leaf node is reached. The leaf node s_T corresponds to a final reward $\vec{z}$, which is then back-propagated along the trajectory.

Back-propagation Update $\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$ for each edge in the playout such that

$$N(s,a) = N(s,a) + 1; \tag{3}$$

$$W(s,a) = W(s,a) + \vec{z}; \tag{4}$$

$$Q(s,a) = W(s,a)/N(s,a); \tag{5}$$

Self-play Data and Network Update After a batch of simulations, we gather data in the form of (s_t, π_t, z_t), where π_t represent the posterior distribution of actions a_t in edges that stems from s_t, and $\vec{z}_t = \pi_t' Q(s_t, a_t)$. The neural network weights θ are updated so as to minimize the error between the predicted outcome $\vec{v}$ and the game outcome $\vec{z}$, and to maximize the similarity of the policy vector $\mu = (q, 1-q)$ to the search probabilities π. More precisely,

$$\theta^* = \arg\min_\theta l(\theta) = \arg\min_\theta \left\{ ||\vec{z} - \vec{v}||^2 - \pi' \log \mu + c||\theta||^2 \right\}. \tag{6}$$

2.3 Network Architecture

Recall that the game state is represented by two parts: a matrix M concerning each player's state and a vector b concerning the card and coins on the table. We process these two components differently:

(i) The matrix M is first transformed into a three-dimensional array of size $33 \times n \times 3$, corresponding to three channels of $33 \times n$ grids. The array is then fed into a convolutional neural network (CNN) branch consisting of several convolution, pooling, flatten, linear, ReLU, and batch normalization layers.

(ii) We append the number of remaining cards in the deck as a third element to the vector b. The extended vector is then processed by a fully connected branch consisting of several linear, ReLU, and batch normalization layers.

The outputs of the two branches are concatenated and fed into the main branch, which is a classic feedforward network. The main branch then splits into two heads: the value head applies tanh() to the final layer to ensure that the output $\vec{v}$ is restricted to the range $[-1, 1]$, while the policy head applies sigmoid() to produce valid output probabilities.

Figure 1 summarizes the general neural network architecture.

2.4 Measure of Performance

We evaluate the performance of our gaming bot by testing it against a mixed group of pure online MCTS agents, rule-based agents, and random agents. The online MCTS agent follows the classic UCT algorithm. The rule-based agent follows predefined rules, such as "take the card if it is consecutive to my cards in hand"; if no rule applies, the agent makes a random move. The random

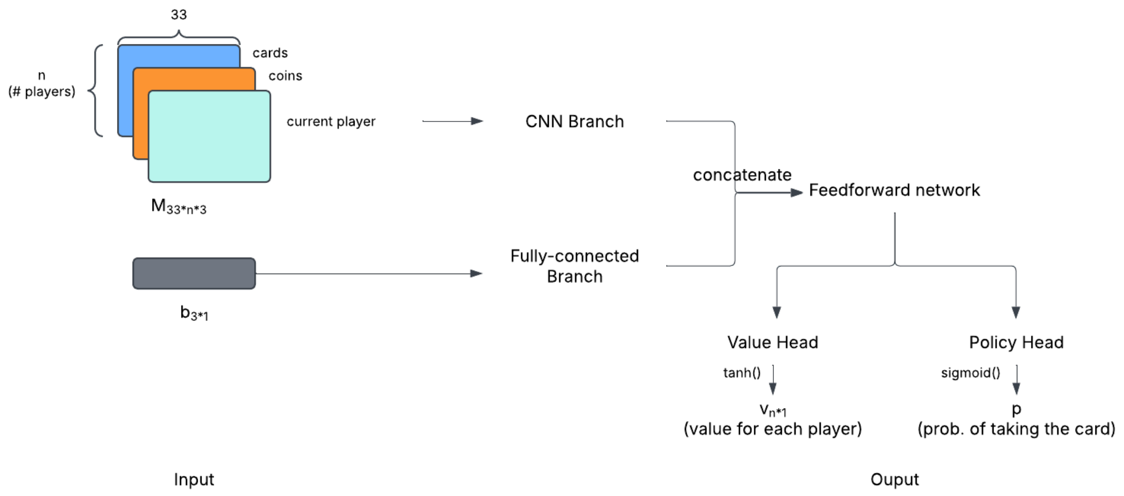

Figure 1: Design of the Policy-Value Network

agent makes random decisions, with a 50% chance of taking or passing unless it runs out of coins. We simulate multiple rounds and evaluate the overall rank of our bot.

3 Expected Results

We expect our AlphaZero-based bot to outperform rule-based and random agents by learning optimal strategies through self-play. Against pure online MCTS agents, the bot should demonstrate comparable or superior performance due to its ability to refine policy and value estimations through deep reinforcement learning. We also anticipate that the bot's performance will improve over successive training iterations, as the neural network becomes more effective at guiding Monte Carlo Tree Search.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[2] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] S. Russell and P. Norvig, "Artificial intelligence: A modern approach, 4th, global edition pearson," *Harlow, UK*, 2021.

[5] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. DOI: 10.1109/TCIAIG.2012.2186810.

[6] N. Petosa and T. Balch, *Multiplayer alphazero*, 2019. arXiv: 1910.13012 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1910.13012.

[7] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver, “Planning in stochastic environments with a learned model,” in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=X6D9bAHhBQ1.

[8] C. D. Rosin, “Multi-armed bandits with episode context,” *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, 2011.