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

AMOD 5310H Artificial Intelligence Term Project

Smars Hu, Trung Kien Ngo, Lya Wang

March 26, 2025

Table of Contents

Motivation

- *No Thanks!*: popular, multiplayer, turn-based, stochastic card game, centered around risk and resource management.
- *Goal*: Build an AI gaming bot for *No Thanks!*
- *Algorithm*:
  — Inspired by *AlphaGo*, the well-known breakthrough in AI that mastering Go game.
  — Model-based RL: Combining Monte Carlo Tree Search (MCTS) and deep neural network.
  — Upgraded version: *AlphaZero*, no prior knowledge RL.

The Game *No Thanks!*

1 Introduction

- 3-7 players, 33 cards (3 to 35 points), 11 coins (each is worth -1) for each player.
- To win: get *lowest* score.
- Each turn: take the card revealed or spend a coin to pass it; if take the card, you can also take the coins accumulated on it.
- 9 cards are removed from the deck randomly.
- Consecutive cards count only for the lowest-numbered card in the sequence.

AlphaZero Algorithm

- Generalized reinforcement learning algorithm extending AlphaGo Zero and AlphaGo.
- Trained solely through self-play reinforcement learning.
- A single deep network to estimate both the policy and value functions simultaneously.
- MCTS guided by the policy-value network to perform self-play (PUCT).
- The network is then iteratively updated based on self-play data, with each iteration refining move selection and evaluation.

Contribution

- To the best of our knowledge, there has been no such algorithm applied to this specific game.
- Explore and extend the applicability of AlphaZero-like algorithms to multiplayer, stochastic games.

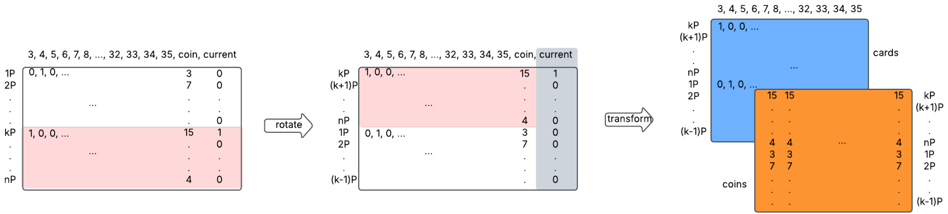
Table of Contents

Modeling the Game
2 Methodology

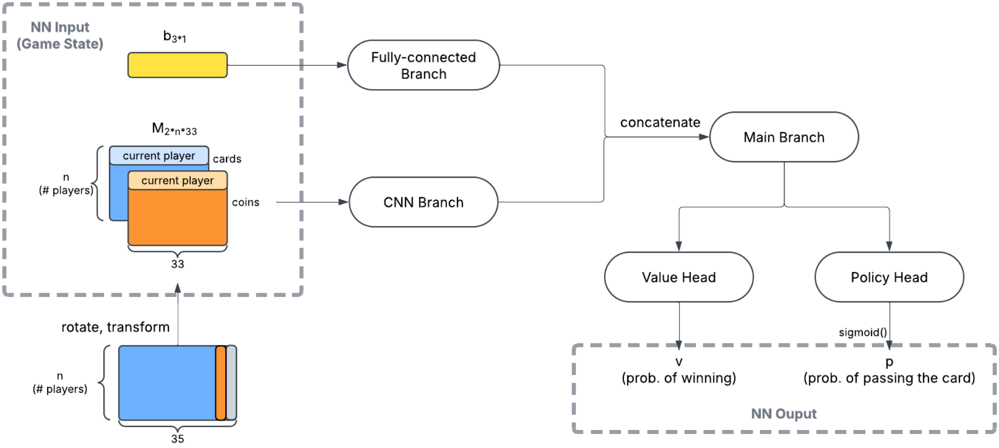
- The game is Markovian if all players play with the optimal strategy.

Game State: (M, b)

- M: Matrix representing cards and coins holding by each player;
- b: vector: (card in play, coins in play, remaining cards in deck).



Deep Network Architecture
2 Methodology



Monte Carlo Tree Search (MCTS) Self-play

- Keep track of *Edge*: $(s, a) \to [N(s, a), W(s, a), Q(s, a), P(s, a)]$.
- *Step 1. Selection:* At each state,

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

where

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

- *Step 2. Expansion:* Expand from the root to a leaf (terminal state), where the terminal rewards can be found; we define the terminal rewards to be 1 for winning and 0 for losing.

MCTS Self-play (Ct.d)

- *Step 3. Back-propagation:* Update $\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$ for each edge in the playout such that

$$N(s,a) = N(s,a) + 1; \tag{3}$$
$$W(s,a) = W(s,a) + z; \tag{4}$$
$$Q(s,a) = W(s,a)/N(s,a). \tag{5}$$

- *Step 4. Choose Action at Root:* After searching, at the root, choose the edge with highest visits; i.e.,

$$a_0 = \arg\max_a N(s_0, a). \tag{6}$$

MCTS Self-play (Ct.d)

- *Step 5. Update Network and Prior:* Gather self-play data (s, a, q), where $a = 1$ for PASS, $a = 0$ for TAKE, and $q = Q(s, a)$; Update NN weights θ :

$$\begin{aligned} \theta^* &= \arg\min_{\theta} l(\theta) \\ &= \arg\min_{\theta} \left\{ \|q - v_\theta\|^2 - [a \log \mu_\theta + (1-a) \log(1-\mu_\theta)] + c_{L2} \|\theta\|^2 \right\}. \end{aligned} \tag{7}$$

And finally update prior:

$$P(s, a) = \mu_{\theta^*}, \tag{8}$$

where

$$v_\theta, \mu_\theta = \mathrm{NN}_\theta(s).$$

Performance Evaluation

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.

Table of Contents

Current Progress

- The code ran properly locally and has been tested for small data (10 rounds training, each round 4 games played);
- The bot is not smart because of insufficient training:
  — NN cannot capture the stochastic nature if it's not in data.
- Challenge: Self-play is very slow.
  — Paralleled;
  — Cut the game after some moves - states closing to the end are not as helpful.

To-Dos

- Bring the code to *Kaggle* and train the model with LARGE amount of self-play data;
- Tune hyperparameters to make the bot smarter;
- More experiments to evaluate the performance;
- Optimize the code.

Q&A

Thank you for listening!
Your feedback will be highly appreciated!