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 22, 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.

- 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; 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.

- 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 card games.

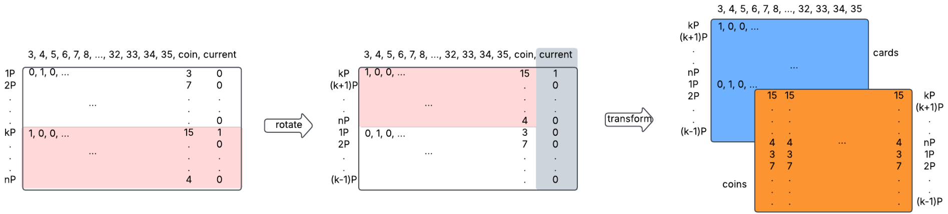
Table of Contents

Modeling the Game

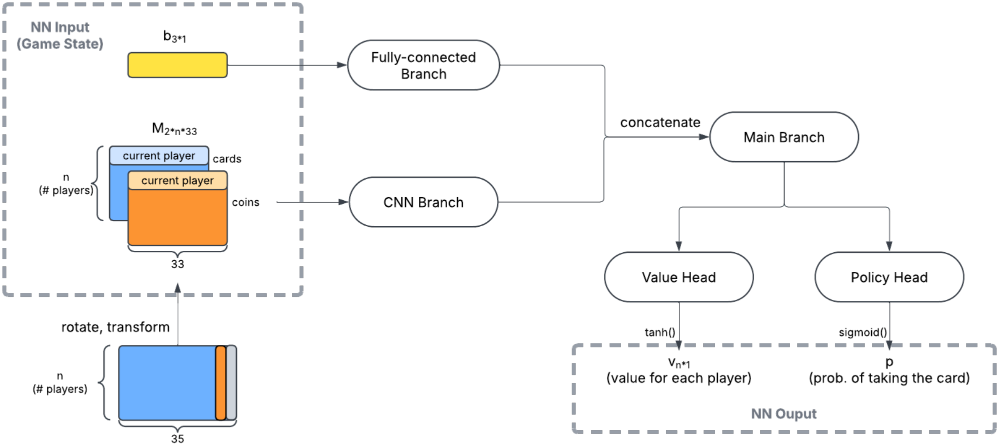
- The game is Markovian if all players play with the optimal strategy.

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



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; e.g., $\vec{z}$ such that $z_k = 1, 0.5, 0, -0.5, -1$ for Player k being rank one to five.

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

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

- *Step 4. Update network and prior:* Gather self-play data (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 $\vec{z}_t = \pi_t' Q(s_t, a_t)$; Update NN weights θ :

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

  And finally update prior:

$$P(s,a) = \text{(the policy output of) } \mathrm{NN}_\theta(s) \tag{7}$$

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

- tba

To-Dos

- 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!