Project Report: MicroGo

Submitted By: Soem (22k-4629), Danish Raza (22k-4183), Muhammad Danish (22k-4381)

Course: Artificial Intelligence

Instructor: Sir Abdullah Yaqoob

1. Executive Summary

Project Overview:

The MicroGo project aimed to develop a simplified version of the traditional Go game, tailored for a 5x5 board with modified rules, including stone expiration and limited stone counts. The primary objective was to implement a Reinforcement Learning (RL)-based AI using a Convolutional Neural Network (CNN) combined with Monte Carlo Tree Search (MCTS) to play the game competitively. The AI was pretrained on random games and further refined through self-play, achieving strong performance against random and pretrained agents.

2. Introduction

Background:

Go is a strategic board game traditionally played on a 19x19 grid, where two players (Black and White) place stones to control territory. Its complexity makes it computationally challenging for AI. MicroGo simplifies Go to a 5x5 board, introducing novel mechanics like stone expiration after three turns and a limited stone supply (30 per player). This project was chosen to explore AI strategies in a constrained yet strategically rich environment, leveraging modern RL techniques inspired by AlphaGo.

Objectives of the Project:

1. Design and implement MicroGo with modified rules to enhance strategic depth.
2. Develop an AI using a CNN and MCTS to play MicroGo effectively.
3. Pretrain the AI on random game data and improve it through RL-based self-play.
4. Evaluate the AI's performance against random agents and its pretrained version.
5. Create an interactive in-notebook interface for human-AI gameplay.

3. Game Description

Original Game Rules:

In traditional Go, two players alternate placing black or white stones on a grid (typically 19x19) to surround territory. A stone or group is captured if surrounded by the opponent's stones. The game ends when both players pass, and the winner is determined by territory control (area scoring) plus captured stones.

Innovations and Modifications:

MicroGo introduces the following changes:

- **Board Size**: Reduced to 5x5 for computational efficiency.
- **Stone Expiration**: Stones expire after three turns, adding dynamic board management.
- **Limited Stones**: Each player has 30 stones, limiting placement options.
- **Scoring**: A territory heuristic awards 1 point per live stone and 0.5 points per empty intersection (shared by both players), preventing zero-score draws. If tied, the number of stones placed breaks the tie.
- **Game End**: The game ends after two consecutive passes or when both players exhaust their stones.

4. AI Approach and Methodology

AI Techniques Used:

The AI leverages a combination of:

- **Convolutional Neural Network (GoNNet)**: Predicts move probabilities (policy) and game outcome (value).
- **Monte Carlo Tree Search (MCTS)**: Guides move selection by simulating games, balancing exploration and exploitation.
- **Reinforcement Learning**: Refines the neural network through self-play, using game outcomes to update policy and value predictions.
- **Supervised Pretraining**: Initializes the network with data from 10,000 random games.
- **Data Augmentation**: Applies board rotations and flips to enhance training data.

Algorithm and Heuristic Design:

- **GoNNet Architecture**: Consists of an initial convolution layer, three residual blocks, and separate policy and value heads. The policy head outputs probabilities over 26 actions (25 board positions + pass), while the value head predicts win probability (-1 to 1).

- **MCTS**: Uses Upper Confidence Bound (UCB) to select actions, incorporating neural network predictions and a cpuct exploration parameter (1.5). Each simulation evaluates 40 game rollouts.
- **Training**: Pretraining minimizes policy loss (cross-entropy) on random game data. RL training optimizes policy and value losses using self-play data, with mixed-precision training (AMP) for efficiency.
- **Heuristics**: The territory score (live stones + 0.5 * empty spaces) guides game evaluation, with a tiebreaker based on stones placed.

AI Performance Evaluation:

- **Pretrained Model**: Achieved 8% policy accuracy after 12 epochs on 120,000 random game samples.
- **RL-Trained Model**: Evaluated via 50-game series against a random agent and 30 games against the pretrained model. Results (see Section 8) show significant improvement over random and pretrained baselines.
- **Metrics**: Win rate, decision time (negligible due to CUDA acceleration), and policy accuracy.

5. Game Mechanics and Rules

Modified Game Rules:

- Players alternate placing stones on a 5x5 grid or passing.
- Each stone has an age (0 to 2); stones age every two turns and expire at age 3.
- Each player starts with 30 stones; placement is forbidden when stones are depleted.
- Passing increments the turn count and ages stones if it completes a round.

Turn-Based Mechanics:

- White (player 1) moves first, followed by Black (player -1).
- A move is either placing a stone on an empty intersection or passing.
- After each move, the board updates stone ages (every two turns) and checks for expiration.

Winning Conditions:

- The game ends when both players pass consecutively or both exhaust their stones.
- The winner is determined by the territory score (live stones + 0.5 * empty intersections).
- If scores are tied, the player with more stones placed wins. Absolute ties (rare) yield a small positive score (1e-4).

6. Implementation and Development

Development Process:

1. **Game Logic**: Implemented the Board and GoGame classes to manage board state, moves, and scoring.
2. **Neural Network**: Designed GoNNet with PyTorch, incorporating residual blocks for robust feature extraction.
3. **Pretraining**: Generated 600,000 samples from 10,000 random games, training the network on a 120,000-sample subset.
4. **Self-Play and RL**: Conducted 8 RL iterations, each with 60 self-play games, refining the network with MCTS-guided policies.
5. **Evaluation**: Tested AI performance in 50-game series against random agents and 30 games against the pretrained model.
6. **Interface**: Developed an interactive HTML/JS-based interface in Colab for human-AI gameplay.

Programming Languages and Tools:

- **Programming Language**: Python
- **Libraries**: PyTorch, NumPy, Matplotlib, TQDM, IPython
- **Tools**: Google Colab (GPU support), GitHub (version control), Pickle for data caching

Challenges Encountered:

- **Training Stability**: Initial RL training was unstable due to high variance in self-play outcomes. Addressed by reducing learning rate (5e-editor: **Data Augmentation**: Rotations and flips increased dataset diversity but required careful memory management.
- **Performance**: MCTS simulations were computationally expensive. Using CUDA and limiting simulations to 40 per move balanced speed and strength.
- **Interface**: Integrating JavaScript with Colab's Python kernel required precise callback handling for real-time updates.

7. Team Contributions

- **Soem (22k-4629)**: Developed the neural network (GoNNet) and implemented MCTS for AI move selection.
- **Danish Raza (22k-4183)**: Designed and implemented MicroGo game rules, board mechanics, and scoring heuristics.
- **Muhammad Danish (22k-4381)**: Built the interactive game interface, handled data augmentation, and conducted AI performance evaluations.

8. Results and Discussion

AI Performance:

- **Pretrained Model vs. Random Agent**: In a 50-game series, the pretrained AI won 45 games, lost 3, and drew 2 (90% win rate).
- **RL-Trained Model vs. Random Agent**: The RL-trained AI won 48 games, lost 1, and drew 1 (96% win rate).
- **RL-Trained vs. Pretrained**: In a 30-game series, the RL-trained AI won 22 games, lost 5, and drew 3 (73.3% win rate).
- **Decision Time**: Average decision time was ~0.1 seconds per move on CUDA-enabled GPUs, ensuring responsive gameplay.
- **Effectiveness**: The RL-trained AI demonstrated superior strategic play, effectively managing stone expiration and optimizing territory control. The interactive interface allowed seamless human-AI gameplay, with the AI winning most matches against human testers.

---

9. References

1. Silver, D., et al. (2016). "Mastering the game of Go with deep neural networks and tree search." *Nature*, 529(7587), 484-489.
2. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
4. PyTorch Documentation. https://pytorch.org/docs/stable/index.html
5. Google Colab Documentation. https://colab.research.google.com/
6. AlphaGo Resources. https://deepmind.com/research/case-studies/alphago