Mastering the Game of Go

Deep Neural Networks and Tree Search (AlphaGo)

Rushil Gupta, Dhruman Gupta

April 24, 2025

Table of Contents

Introduction

- **AlphaGo:** A computer program developed by Google DeepMind to play the board game Go.
- Uses deep neural networks combined with Monte Carlo Tree Search (MCTS).

Introduction

- First program to defeat a human professional Go player (Fan Hui) on a full-sized 19x19 board without handicap.
- Achieved a 5-0 victory in a formal match.
- Considered a grand challenge for Artificial Intelligence, previously thought to be decades away.

The Game: Go

- Go is a game of perfect information, like chess.
- **Challenge:** Extremely difficult for AI due to:
  - **Enormous Search Space:**
    - Branching factor $b \approx 250$.
    - Game depth $d \approx 150$.
    - Number of sequences $\approx b^d \approx 250^{150}$.
    - Exhaustive search is infeasible.
  - **Difficult Position Evaluation:** Hard to judge who is winning from a given board state.

Table of Contents

Methods Before AlphaGo: MCTS

- **Monte Carlo Tree Search (MCTS):** State-of-the-art before AlphaGo.
- **Core Idea:** Build a search tree, estimate state values using random simulations (rollouts).
- **Rollout Intuition:**
  - From a state s, play out many games randomly (or using a simple policy) to the end.
  - Average the win/loss outcomes from these rollouts to estimate the value of s.

Methods Before AlphaGo: MCTS

- MCTS balances exploration and exploitation, and used variants of UCB to select actions.
- **Limitations:** Often relied on shallow policies or simple value functions. Strong amateur level play was achieved.

AlphaGo's Methods: Overview

Key Idea: Use deep neural networks to guide MCTS.

- **Policy Network** $p(a|s)$**:** Predicts probability of choosing action a in state s. Reduces search *breadth*.
- **Value Network** $v(s)$**:** Estimates the probability of the current player winning from state s. Reduces search *depth*.

AlphaGo's Methods: Overview

Training Pipeline:

1. **Data Collection:** Collect games from human experts.
2. **SL Policy Network (p_σ):** Train on human expert games.
3. **RL Policy Network (p_ρ):** Improve SL network via self-play, optimizing for winning.
4. **Value Network (v_θ):** Train to predict game outcome from self-play games using the RL policy network.

Table of Contents

MDP Formulation

Go framed as a Markov Decision Process (MDP) / alternating Markov game:

- **States** $s \in S$**:** Board position + current player.
- **Actions** $a \in A(s)$**:** Legal moves.
- **Transition** $s' = f(s, a)$**:** Deterministic next state.

MDP Formulation

- **Reward Function $r(s)$:**
  - $r(s) = 0$ for non-terminal states $(t < T)$.
  - At terminal state s_T, reward is based on game outcome

$$
r(s) = \begin{cases} 0 & \text{if } t < T \\ 1 & \text{if win at } s_T \\ -1 & \text{if loss at } s_T \end{cases}
$$

Table of Contents

Supervised Learning (SL) Policy Network (p_σ)

- **Goal:** Imitate human expert moves.
- **Network:** 13-layer Convolutional Neural Network (CNN).
  - Input: Board state s.
  - Output: Probability distribution $p_\sigma(a|s)$ over legal moves a.

Supervised Learning (SL) Policy Network (p_σ)

- **Training Data:** 30 million positions from KGS Go Server.
- **Objective:** Maximize log likelihood of move a in state s:

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

- **Result:** 57.0% accuracy on test set, (3ms/move).

Supervised Learning (SL) Policy Network (p_σ)

- Another smaller, linear model was also trained for faster rollouts.
- **Fast Rollout Policy (p_π):** 24.2% accuracy, 2μs/move.

Reinforcement Learning (RL) Policy Network (p_ρ)

- **Goal:** Improve p_σ to maximize winning probability, not just accuracy.
- **Method:** Policy Gradient Reinforcement Learning.
- **Initialization:** Start with SL network weights ($\rho = \sigma$).

Reinforcement Learning (RL) Policy Network (p_ρ)

- **Training:**
  - Play games between current network p_ρ and random previous versions of p_{ρ^-} (random previous checkpoint).
  - Update weights using REINFORCE algorithm to maximize expected outcome z_t:

    $$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

    Note: they use baseline $= v(s_t)$ for variance reduction

What we have so far

- A policy network p_π that isn't very good, but is very *fast*. Trained using supervised learning on human games.
- A policy network p_ρ that can play at at the level of a strong amateur. Trained using reinforcement learning on self-play games.

Table of Contents

Value Network (v_θ)

- **Goal:** Estimate state value $v^{p_\rho}(s)$ = expected outcome from state s if both players use policy p_ρ.

$$v^p(s) = \mathbb{E}[z_t | s_t = s, a_{t \dots T} \sim p]$$

- **Network** $v_\theta(s)$**:** CNN similar to policy net, outputs single scalar value.

Value Network (v_θ)

- **Challenge:** Training on full games leads to overfitting due to correlated positions, and does not generalize well.
- **Solution:** Generate a new dataset of 30 million *unique* positions, each from a separate self-play game using p_ρ.
- **Objective:** Minimize Mean Squared Error (MSE) between prediction $v_\theta(s)$ and actual outcome z:

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))$$

Value Network (v_θ)

Results:

- Much more accurate than rollouts with p_π.
- Approached accuracy of rollouts with p_ρ but vastly faster.

Table of Contents

MCTS in AlphaGo: Notation

Key Notation:

- $N(s, a)$ - Visit count for state-action pair
- $Q(s, a)$ - Action value (expected outcome)
- $P(s, a)$ - Prior probability from policy network
- $v_\theta(s)$ - Value network prediction
- $p_\pi(a|s)$ - Fast rollout policy
- L - Maximum depth of tree search

MCTS in AlphaGo

Combines policy networks, value networks, and Monte Carlo rollouts within MCTS.

- **Tree Edges:** Store action value $Q(s, a)$, visit count $N(s, a)$, prior probability $P(s, a)$.

MCTS in AlphaGo: Algorithm Steps

Algorithm Steps (1-2):

1. **Select:** From root to leaf, choose actions by maximizing:

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t, a) + c_{\text{puct}} \cdot P(s_t, a) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)} \right)$$

2. **Expand:** Create new leaf node s_L. Initialize prior probabilities using the SL policy network:

$$P(s_L, a) = p_\sigma(a|s_L)$$

Algorithm Step (3):

3. **Evaluate:** Estimate node value using a combination of:
   - Value network: $v_\theta(s_L)$ - Deep strategic evaluation
   - Rollout: z_L - Fast simulation to end of game using p_π

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L \quad \text{where } \lambda = 0.5$$

Algorithm Step (4):

4. **Backup:** Update statistics for all visited nodes:
   - Increment visit counts: $N(s,a) \leftarrow N(s,a) + 1$
   - Update action values:

$$Q(s,a) \leftarrow \frac{N(s,a) \cdot Q(s,a) + V(s_L)}{N(s,a) + 1}$$

Table of Contents

Results

- **Against Programs:** Single machine AlphaGo won 99.8% (494/495) games vs strongest Go programs. Distributed version won 100%.
- **Against Human Professional:**
  - Defeated Fan Hui (3x European Champion, 2p) 5-0 in a formal match.
  - First time a computer beat a pro player without handicap.
- **Search Efficiency:** Evaluated thousands of times *fewer* positions than Deep Blue (chess), but selected/evaluated them more intelligently using the neural networks.