AlphaGo Family

ATCI: Papers' Project

Bruno Sánchez Gómez

June 12, 2025

Abstract

The success of DeepMind's AlphaGo [1] and its successors, AlphaZero [2] and MuZero [3], marked a pivotal moment in Artificial Intelligence, particularly in Reinforcement Learning (RL). This essay critically examines these three landmark systems. We will delve into their core methodologies, tracing the evolution from AlphaGo's reliance on human expert data and distinct network components to AlphaZero's unified, tabula rasa learning, and finally to MuZero's ability to master games without prior knowledge of their rules by learning its own model of the environment. The mathematical principles of their learning algorithms and search techniques will be explored. Furthermore, these systems will be contextualized within broader RL paradigms, and their strengths, limitations, and profound impact on the field will be evaluated.

1 Introduction

The game of Go, with its vast search space and complex positional evaluation, long stood as a grand challenge for Artificial Intelligence. Traditional AI approaches, successful in games like chess, struggled to reach professional human levels in Go. The advent of the AlphaGo family of algorithms, developed by DeepMind, represented a paradigm shift, not only conquering Go but also providing a general framework for tackling complex decision-making problems. This essay aims to:

1. Explain the core technical contributions of AlphaGo [1], AlphaZero [2], and MuZero [3].
2. Discuss the mathematical principles of their learning and search processes.
3. Contextualize these methods within the broader landscape of Reinforcement Learning.
4. Offer a critical evaluation of their strengths, weaknesses, and overall impact.

2 The AlphaGo Family: An Evolutionary Journey

The AlphaGo family represents a clear progression in reducing reliance on domain-specific knowledge and human data, moving towards more general and autonomous learning agents.

2.1 AlphaGo (Silver et al., 2016) [1]

AlphaGo was the first program to defeat a human professional Go player and later a world champion (Lee Sedol). Its architecture was a sophisticated blend of deep neural networks and Monte Carlo Tree Search (MCTS).

Core Components:

- **Policy Network (p_σ):** Trained via supervised learning (SL) on a large dataset of human expert games to predict expert moves. Input: board state s. Output: probability distribution $p_\sigma(a|s)$ over actions a. The update rule for SL is a standard gradient ascent to maximize log-likelihood:
$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma} \tag{1}$$
- **Fast Rollout Policy (p_π):** A simpler, faster linear softmax policy also trained on human expert data, used for quick playouts during MCTS simulations.

- **Reinforcement Learning (RL) Policy Network (p_ρ):** Initialized from the SL policy network p_σ, this network was further improved through self-play. Games were played between current and previous versions of the policy network. The update rule is a policy gradient method (akin to REINFORCE):

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t \tag{2}$$

  where $z_t = \pm 1$ is the terminal reward (win/loss) from the perspective of the current player at time t.

- **Value Network (v_θ):** Trained to predict the outcome (probability of winning) from a given board position s, when games are played by the RL policy network p_ρ against itself. It is trained by regression to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$ and the actual game outcome z:

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s)) \tag{3}$$

- **Monte Carlo Tree Search (MCTS):** AlphaGo's MCTS algorithm integrated these components.

  - **Selection:** Traversed the tree by selecting actions a_t at each state s_t to maximize an action-value $Q(s_t, a)$ plus a bonus $u(s_t, a)$ based on the SL policy network's prior probability $P(s, a) = p_\sigma(a|s)$:

$$a_t = \arg\max_a \left(Q(s_t, a) + u(s_t, a)\right), \quad \text{where } u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)} \tag{4}$$

    $N(s, a)$ is the visit count of edge (s, a).

  - **Expansion:** When a leaf node s_L was reached, it was expanded. The SL policy network p_σ provided prior probabilities for new actions.

  - **Evaluation:** The leaf node s_L was evaluated in two ways: by the value network $v_\theta(s_L)$ and by a Monte Carlo rollout to the end of the game using the fast rollout policy p_π to get an outcome z_L. These were combined: $V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$.

  - **Backup:** Action-values $Q(s, a)$ and visit counts $N(s, a)$ were updated based on the evaluations.

AlphaGo's success was remarkable, but it relied on supervised learning from human data for initial policy learning and featured distinct, separately trained network components.

2.2 AlphaZero (Silver et al., 2017) [2]

AlphaZero marked a significant step towards generality. It learned to play Go, chess, and shogi from scratch (tabula rasa), without any human data, domain-specific heuristics (beyond game rules), or handcrafted features.

Core Innovations:

- **Single Neural Network (f_θ):** A single, deep neural network $f_\theta(s) = (\mathbf{p}, v)$ takes the board state s as input and outputs both a policy vector $\mathbf{p}$ (probabilities $P_\theta(a|s)$ for all actions) and a scalar value v (predicted outcome $v_\theta(s)$).

- **Pure Self-Play Reinforcement Learning:** The network is trained entirely from games of self-play.

- **MCTS Modification:**

  - **Selection:** The MCTS selection step uses an Upper Confidence Bound for Trees (UCT) variant similar to AlphaGo, but the prior probabilities $P_\theta(a_i|s_t)$ now come directly from the policy head of the single network f_θ. The action a_t^{UCB1} is selected at state s_t according to:

$$a_t^{UCB1} = \arg\max_{a_i \in \mathcal{A}} \left(Q(s_t, a_i) + cP_\theta(a_i|s_t) \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a_i)} \right) \tag{5}$$

    where c is an exploration constant. (Note: The slides use $t(S_t)$ for $\sum_b N(s_t, b)$).

  - **Expansion and Evaluation:** When a leaf node s_L is expanded, its value v and policy priors $\mathbf{p}$ are obtained directly from the single network $f_\theta(s_L)$. There are *no Monte Carlo rollouts* to the end of the game; the value head $v_\theta(s_L)$ provides the sole evaluation for backpropagation.

  - **Action Execution during Self-Play:** The policy $\pi(s, a)$ for executing moves during self-play is based on the visit counts $N(s_t, a_i)$ from the MCTS at the root state s_t:

$$\pi(s_t, a_i) = \frac{N(s_t, a_i)}{\sum_b N(s_t, b)} \tag{6}$$

    Often, temperature is used to encourage exploration in early game stages.

- **Learning:** The network f_θ is trained on data $(s_t, \pi(s_t), z_t)$ collected from self-play games, where $\pi(s_t)$ is the MCTS search policy and z_t is the final game outcome. The loss function l minimizes both the error in the value prediction and the mismatch between the network's policy $\mathbf{p}_\theta(s_t)$ and the MCTS search policy $\pi(s_t)$:

$$l = \sum_t \left((v_\theta(s_t) - z_t)^2 - \pi(s_t)^T \log(\mathbf{p}_\theta(s_t)) \right) + c'||\theta||^2 \tag{7}$$

  The first term is the MSE for the value head, the second is the cross-entropy loss for the policy head, and c' is an L2 regularization parameter.

AlphaZero achieved superhuman performance in Go, chess, and shogi, surpassing specialized programs and AlphaGo itself, demonstrating remarkable generality. However, it still required a perfect simulator of the game (the rules) to perform MCTS expansions.

2.3 MuZero (Schrittwieser et al., 2019) [3]

MuZero took generality a step further by learning to master games *without knowing their rules.* It combines tree-based search with a learned model of the environment's dynamics.

Core Innovations: MuZero introduces three core neural network components, all learned end-to-end:

- **Representation Function (h_θ):** Maps a history of recent observations $o_1, ..., o_t$ to an initial hidden state $s^0 = h_\theta(o_1, ..., o_t)$. This s^0 forms the root of the MCTS.

- **Dynamics Function (g_θ):** Given a previous hidden state s^{k-1} and a candidate action a^k (hypothetically taken within the MCTS), it predicts the next hidden state s^k and the

immediate reward r^k:

$$r^k, s^k = g_\theta(s^{k-1}, a^k) \tag{8}$$

This function allows MuZero to "simulate" future states and rewards internally.

- **Prediction Function (f_θ):** Similar to AlphaZero's network, it takes a hidden state s^k (either s^0 from h_θ or s^k from g_θ) and predicts a policy $\mathbf{p}^k$ and a value v^k:

$$\mathbf{p}^k, v^k = f_\theta(s^k) \tag{9}$$

- **MCTS within the Learned Model:** MCTS is performed using these learned functions.
  - Starting from $s^0 = h_\theta$(observations).
  - Transitions within the tree use $s^k = g_\theta(s^{k-1}, a^k)$.
  - Policy priors and node evaluations use $\mathbf{p}^k, v^k = f_\theta(s^k)$.
  - The MCTS accumulates the predicted rewards r^k along a trajectory.
- **Learning:** The model (θ) is trained to predict three quantities simultaneously for K unrolled steps:
  1. The policy $\mathbf{p}_t^k$ should match the MCTS search policy π_{t+k}.
  2. The value v_t^k should match the actual (target) value z_{t+k} (e.g., n-step return or game outcome).
  3. The reward r_t^k should match the observed reward u_{t+k} from the environment.

  The overall loss function is a sum of these individual losses:

$$L_t(\theta) = \sum_{k=0}^{K} \Big(L^r(u_{t+k}, r_t^k) + L^v(z_{t+k}, v_t^k) + L^p(\pi_{t+k}, \mathbf{p}_t^k) \Big) + c''||\theta||^2 \tag{10}$$

  where L^r, L^v, L^p are specific loss functions (e.g., cross-entropy for categorical rewards/values, MSE for continuous, cross-entropy for policy).

MuZero matched AlphaZero's performance in Go, chess, and shogi, and achieved state-of-the-art results in the Atari benchmark, all without being given the rules of the games. This demonstrates a significant step towards general-purpose, model-based RL.

3 Contextualization with Broader RL Concepts

The AlphaGo family integrates and advances several key RL concepts:

- **Model-Free vs. Model-Based RL:**
  - **AlphaGo and AlphaZero** are primarily model-free in their learning approach for the policy and value networks. They learn directly from experience (self-play outcomes z_t). However, their MCTS component relies on a perfect model of the environment (the game simulator) for state transitions during search.
  - **MuZero** is explicitly a model-based RL algorithm. It learns a model of the environment's dynamics (g_θ), rewards (g_θ), and relevant state representations (h_θ, s^k).

Planning (MCTS) then occurs entirely within this learned model. This is a significant departure and aligns with a long-standing goal in RL to combine the data efficiency of model-based methods with the performance of model-free methods.

- **Monte Carlo Tree Search (MCTS):** MCTS is the backbone of the planning component in all three systems. It's a powerful heuristic search algorithm for decision processes, balancing exploration and exploitation. The AlphaGo family showcases how deep learning can guide MCTS (providing priors and evaluations) far more effectively than traditional heuristics or random rollouts. AlphaZero's elimination of explicit rollouts in favor of learned value estimates was a key simplification. MuZero further evolves this by performing MCTS within a learned latent space.

- **Deep Learning for Function Approximation:** All three systems leverage deep convolutional neural networks (CNNs) to represent the policy and value functions (and the model in MuZero). This allows them to handle high-dimensional state spaces (like a Go board) and learn complex patterns without manual feature engineering. The use of residual networks (ResNets) in AlphaZero and MuZero was crucial for training very deep networks.

- **Policy Gradients and Value-Based Learning:**

  - AlphaGo's RL policy network (p_ρ) was trained using a policy gradient method similar to REINFORCE, learning from the final game outcome.

  - All three systems employ value networks (v_θ) trained to predict outcomes, which is a form of value-based learning. The loss function for AlphaZero and MuZero directly incorporates targets derived from MCTS-improved policies (π) and game outcomes (z_t), effectively performing policy iteration where MCTS is the policy improvement operator and network training is the policy evaluation and fitting step.

- **Self-Play:** This is a crucial data generation mechanism, especially for AlphaZero and MuZero which learn tabula rasa. By playing against continually improving versions of itself, the agent explores the state space and generates high-quality training data.

- **Planning and Learning Integration:** A core theme is the tight integration of lookahead search (planning via MCTS) and function approximation (learning via NNs). MCTS provides strong training targets for the NNs, and the NNs, in turn, make MCTS much more efficient. MuZero exemplifies this by having the learned model itself be the environment for planning.

4 Evaluation and Critique

Strengths:

- **Superhuman Performance:** Achieved and surpassed human world-champion levels in highly complex games.

- **Generality:** AlphaZero and especially MuZero demonstrated applicability across different games and even different types of environments (board games and Atari), with MuZero removing the need for a game simulator.

- **Learning from Scratch:** AlphaZero and MuZero's tabula rasa learning is a significant achievement, reducing reliance on human expertise or data.

- **Discovering Novel Strategies:** The systems often discovered strategies and patterns not previously known or favored by human experts, enriching human understanding of these games.

- **Architectural Evolution:** The progression from separate, specialized networks in AlphaGo to a single, unified network in AlphaZero, and then to a learned model in MuZero, shows a clear path towards more elegant and powerful architectures.

Limitations and Criticisms:

- **Computational Cost and Sample Inefficiency:** Training these systems requires enormous computational resources (e.g., hundreds or thousands of TPUs/GPUs for days or weeks) and vast amounts of experience (e.g., AlphaZero played 4.9 million games of Go for its initial training run for the 3-day experiment, and 29 million for the 40-day run). While impressive, this is far beyond human learning capabilities in terms of raw experience.

- **Perfect Information and Deterministic Environments (Initially):** AlphaGo and AlphaZero were designed for and excelled in fully observable, deterministic, two-player zero-sum games. MuZero extends this to single-agent and stochastic environments (like Atari), but challenges remain for partially observable environments or games with very long horizons and sparse rewards without carefully designed n-step returns or other auxiliary signals.

- **Reliance on MCTS:** While MCTS is powerful, it has its own computational demands (e.g., AlphaZero used 1600 simulations per move in some Go experiments). The performance heavily relies on the quality of the MCTS search, which is guided by the learned components. It's unclear how these approaches would fare in domains where MCTS is less suitable (e.g., continuous action spaces without effective discretization, or extremely large branching factors where even guided search is intractable).

- **Learned Model Complexity (MuZero):** While MuZero's learned model is a breakthrough, training such a model accurately for complex, stochastic, and partially observable environments is extremely challenging. The model only needs to predict quantities relevant for planning (value, policy, reward), not reconstruct observations, which helps, but it can still suffer from compounding errors over long rollouts within the model.

- **Interpretability:** The strategies learned by these deep neural networks are often opaque, making it difficult to understand *why* a particular move is chosen, beyond the network's output probabilities and values.

- **Requirement of Rules (AlphaZero):** As noted on slide 7, AlphaZero still requires knowledge of game rules to run MCTS. MuZero addresses this critical limitation by learning the rules implicitly through its dynamics model g_θ.

- **Still Generates a Tree (AlphaZero/MuZero in production):** Slide 7 raises the question for AlphaZero (and it applies to MuZero) of why, when playing in production, the system still generates a search tree rather than just using the learned policy directly. The answer is that MCTS acts as a policy improvement operator; even a very strong learned policy can be further improved by lookahead search, leading to better performance, especially in critical game situations. The raw policy network in AlphaZero (and MuZero) is strong, but MCTS consistently makes it stronger.

5 Conclusion

The AlphaGo family represents a monumental achievement in AI and Reinforcement Learning. The journey from AlphaGo's expert-data-primed system to AlphaZero's tabula rasa mastery of several board games, and culminating in MuZero's ability to learn game dynamics from scratch, showcases a relentless pursuit of generality and autonomy. These systems have not only solved long-standing AI challenges but have also provided powerful new frameworks that blend deep learning with sophisticated search techniques.

Key mathematical contributions include the effective use of MCTS guided by learned policy and value functions, the end-to-end training of these components through self-play, and in MuZero, the pioneering approach of learning a model of the environment tailored for planning.

While significant challenges related to computational cost, sample efficiency, and applicability to more complex real-world scenarios remain, the AlphaGo lineage has undeniably pushed the boundaries of what is possible in AI. It has laid a strong foundation for future research into developing more general, efficient, and capable intelligent agents. The principles demonstrated—self-play, learned models for planning, and the synergy of search and learning—will likely be central to continued progress in the field.

References

[1] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[2] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *arXiv preprint arXiv:1712.01815* (2017).

[3] Julian Schrittwieser et al. “Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model”. In: *Nature* 588.7839 (2020), pp. 604–609.