AlphaGo Family

ATCI: Papers' Project

Bruno Sánchez Gómez

June 12, 2025

Abstract

The AlphaGo family of algorithms developed by DeepMind represents a series of landmark achievements in Artificial Intelligence (AI) and Reinforcement Learning (RL). This essay critically examines the evolution of these algorithms (AlphaGo, AlphaZero, and MuZero) by analyzing their core methodologies, mathematical foundations, and contributions to the field of RL. We trace their progression from leveraging human expertise to achieving superhuman performance via pure self-play, and ultimately to learning environment dynamics tabula rasa. This analysis highlights the key innovations, trade-offs, and enduring challenges in the quest for general-purpose learning agents capable of mastering complex sequential decision-making problems.

1 Introduction

The challenge of creating Artificial Intelligence capable of excelling in complex strategic games has long been a benchmark for progress in the field. Games like Go, with their vast search spaces and need for nuanced, long-term planning, were considered grand challenges for AI. The development of the AlphaGo family of algorithms by DeepMind marked a paradigm shift, demonstrating the power of deep Reinforcement Learning combined with sophisticated search techniques. This essay aims to critically analyze the evolution of three pivotal systems in this lineage: AlphaGo [1], AlphaZero [2], and MuZero [3]. We will delve into their respective approaches, discuss their mathematical principles, and evaluate their contributions, strengths, and limitations in the broader context of Reinforcement Learning. This journey from AlphaGo, which learned from human data, to AlphaZero, which mastered games purely through self-play, and finally to MuZero, which additionally learned a model of its environment, illustrates a compelling trajectory towards more general and autonomous AI.

2 AlphaGo: Mastering Go with Deep Neural Networks and Tree Search

AlphaGo [1] was the first computer program to defeat a human professional Go player, a feat that underscored the potential of combining deep neural networks with Monte Carlo Tree Search (MCTS). Its success relied on a multi-stage training pipeline and several interacting components.

2.1 Approach Description

AlphaGo's architecture comprised three main neural network components and a sophisticated MCTS algorithm:

1. **Supervised Learning (SL) Policy Network (p_σ):** This network was trained to predict human expert moves from a large dataset of Go games. It took the current board state s as input and outputted a probability distribution $p_\sigma(a|s)$ over possible actions a. This provided a strong initial policy capable of human-like play.

2. **Reinforcement Learning (RL) Policy Network (p_ρ):** The SL policy network was further refined through self-play. The RL policy network p_ρ was initialized with the weights of p_σ and then played games against previous versions of itself. Its parameters were updated using policy gradient methods to maximize the probability of winning.

3. **Value Network (v_θ):** This network was trained to predict the expected outcome (win/loss) from a given board position s. That is, $v_\theta(s) \approx \mathbb{E}[z|s, p_\rho]$, where z is the final game outcome when played by the RL policy p_ρ. It was trained on data generated from self-play games between RL policy networks.

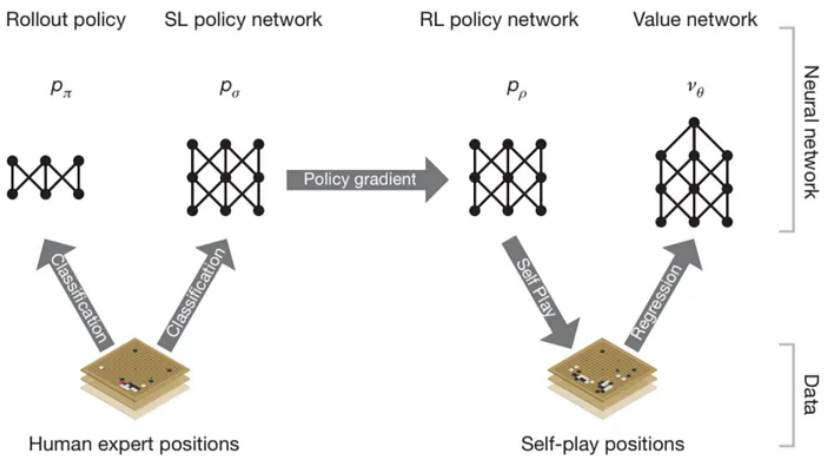

Figure 1: AlphaGo's architecture, showing the interaction between the rollout policy, SL policy network, RL policy network, and value network. The SL policy network is trained on human expert positions, while the RL policy network and value network are trained through self-play. MCTS uses these networks to guide its search (Image source: link).

4. **Monte Carlo Tree Search (MCTS):** AlphaGo's MCTS algorithm integrated the policy and value networks to guide its search.

   - **Selection:** In each simulation, actions were selected within the tree to maximize an action-value $Q(s, a)$ plus an upper confidence bound (UCB) bonus $u(s, a)$. The action a_t at state s_t was chosen as:

     $$a_t = \arg\max_a (Q(s_t, a) + u(s_t, a))$$

     The bonus term $u(s, a)$ was proportional to the prior probability $P(s, a)$ from the SL policy network and inversely proportional to the visit count $N(s, a)$ for that action: $u(s, a) \propto \frac{P(s,a)}{1+N(s,a)}$. This encouraged exploration of promising moves that had not been visited often.

   - **Expansion:** When a simulation reached a leaf node s_L (a state not yet in the tree), the node was expanded. The SL policy network $p_\sigma(a|s_L)$ was used to provide prior probabilities $P(s_L, a)$ for newly expanded actions.

   - **Evaluation:** The leaf node s_L was evaluated using a combination of the value network's output $v_\theta(s_L)$ and the outcome z_L of a fast rollout played to the end of the game using a simpler, linear softmax policy p_π. The final evaluation $V(s_L)$ was a weighted average:

     $$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$$

     where λ is a mixing parameter.

   - **Backup:** After evaluation, the visit counts $N(s, a)$ for all traversed edges were incremented, and their action values $Q(s, a)$ were updated to be the mean of all evaluations $V(s'_L)$ in the subtree below (s, a).

2.2 Mathematical Details

The training of AlphaGo's networks involved distinct loss functions:

- **SL Policy Network (p_σ):** Trained by maximizing the log-likelihood of human expert moves a in state s:
$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

- **RL Policy Network (p_ρ):** Trained using a policy gradient method (REINFORCE) to maximize the expected outcome $z_t \in \{-1, +1\}$ (win/loss from perspective of current player at time t):
$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

- **Value Network (v_θ):** Trained by minimizing the mean squared error (MSE) between its prediction $v_\theta(s)$ and the actual game outcome z from self-play games:
$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s)) \quad \text{(minimizing } (z - v_\theta(s))^2 \text{ loss)}$$

2.3 Contextualization and Critique

AlphaGo's victory was a landmark, demonstrating that complex games like Go were solvable with a combination of deep learning and tree search. The use of MCTS, policy evaluation via self-play, and the ability to beat human masters are all core RL concepts evident in AlphaGo.

Strengths:

- Achieved superhuman performance in Go, a long-standing AI grand challenge.
- Innovatively combined supervised learning from human data, Reinforcement Learning from self-play, and Monte Carlo tree search.
- The policy network effectively pruned the search space for MCTS, and the value network provided a more accurate evaluation than traditional rollouts alone.

Limitations:

- **Reliance on Human Data:** The initial SL policy network required a large dataset of human expert games, limiting its applicability to domains without such data.
- **Domain-Specific Knowledge:** While the neural networks learned features, the MCTS component inherently relied on a perfect simulator (i.e., knowledge of Go rules) to transition between states and determine legal moves.
- **Multiple Networks and Complex Pipeline:** The training involved several distinct networks ($p_\sigma, p_\rho, v_\theta, p_\pi$) and a multi-stage pipeline, increasing complexity.
- **Fast Rollouts:** MCTS still used fast rollouts for leaf evaluation, which, while computationally cheaper than full value network evaluations at every step of a rollout, still represented a computational cost and a potential source of noise.

AlphaGo laid the groundwork for future systems by proving the viability of its core ideas, but its limitations paved the way for the development of more general and streamlined algorithms.

3 AlphaZero: Mastering Games by Self-Play with a General RL Algorithm

AlphaZero [2] represented a significant step towards more general game-playing AI, building upon AlphaGo's success while addressing some of its key limitations. It demonstrated the ability to achieve superhuman performance in Go, Chess, and Shogi, starting entirely from random play (tabula rasa) without any human data or domain-specific heuristics beyond the game rules.

3.1 Approach Description

AlphaZero introduced several key simplifications and generalizations compared to AlphaGo:

1. **Tabula Rasa Learning:** AlphaZero learned entirely through self-play Reinforcement Learning, starting with random weights and no prior human game data. This was a crucial departure from AlphaGo's reliance on supervised pre-training.

2. **Single Neural Network (f_θ):** Instead of separate policy and value networks, AlphaZero used a single, deeper neural network f_θ that took the board state s as input and outputted both a policy vector $\mathbf{p}_\theta$ and a scalar value v_θ: $(\mathbf{p}_\theta, v_\theta) = f_\theta(s)$. The policy vector $\mathbf{p}_\theta$ represented the probabilities $P_\theta(a|s)$ for each action, and v_θ estimated the expected outcome from state s.

3. **No Fast Rollouts in MCTS:** The MCTS algorithm in AlphaZero relied solely on the value v_θ predicted by the neural network for evaluating leaf nodes. This eliminated the need for the fast rollout policy p_π used in AlphaGo, simplifying the search and making evaluations more consistent.

4. **Generality:** The same algorithm, network architecture (with minor game-specific input/output adaptations), and hyperparameters were used across Go, Chess, and Shogi, demonstrating its generality.

MCTS in AlphaZero: The MCTS in AlphaZero was also refined:

- **Selection:** Actions within the tree were selected using a variant of the PUCT (Polynomial Upper Confidence trees) algorithm. At state s_t, the action a was chosen to maximize:

$$Q(s_t, a) + U(s_t, a)$$

  where $U(s_t, a) = c_{\text{puct}} P_\theta(a|s_t) \frac{\sqrt{\sum_b N(s_t,b)}}{1+N(s_t,a)}$. Here, $P_\theta(a|s_t)$ is the prior probability from the policy head of the network f_θ, $N(s_t, a)$ is the visit count of edge (s_t, a), $\sum_b N(s_t, b)$ is the visit count of the parent state s_t, and c_{puct} is a constant controlling exploration.

- **Expansion and Evaluation:** When a leaf node s_L was reached, it was expanded. The neural network f_θ was called once to evaluate s_L, yielding $(\mathbf{p}_\theta(s_L), v_\theta(s_L))$. The value $v_\theta(s_L)$ was then used directly for the backup phase.

- **Backup:** The action values $Q(s, a)$ and visit counts $N(s, a)$ along the traversed path were updated based on the evaluation $v_\theta(s_L)$.

- **Action Selection during Self-Play:** For generating self-play data, after MCTS was completed for the current state s_t, an action a_t was sampled from the improved policy $\pi_t(a|s_t) \propto N(s_t, a)^{1/\tau}$, where τ is a temperature parameter that controls exploration (initially $\tau = 1$, then $\tau \to 0$ for greedy play). The training targets π_t are these MCTS visit count distributions.

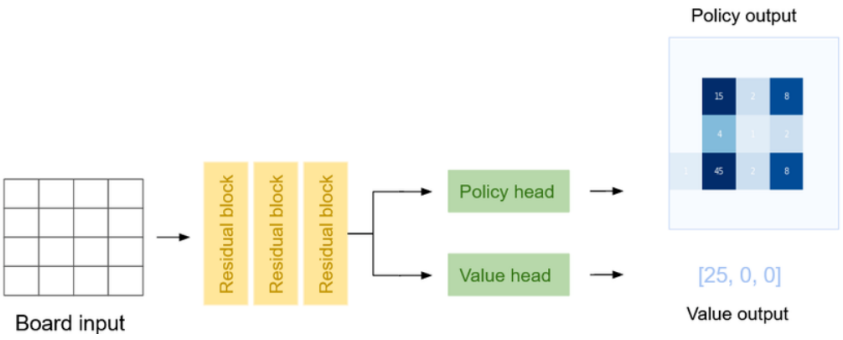

Figure 2: AlphaZero's architecture, showing the single neural network f_θ (inspired by ResNet, with residual blocks) that outputs both policy and value predictions. MCTS uses this network to guide its search without fast rollouts (Image source: link).

3.2 Mathematical Details

AlphaZero's single neural network f_θ was trained using a combined loss function that aimed to simultaneously improve both policy and value predictions. For each state s_t encountered during self-play, with MCTS-derived policy π_t and eventual game outcome z_t, the loss l was:

$$l = (v_\theta(s_t) - z_t)^2 - \boldsymbol{\pi}_t^T \log \mathbf{p}_\theta(s_t) + c||\theta||^2$$

This loss function consists of:

- An MSE term $(v_\theta(s_t) - z_t)^2$ for the value head, encouraging $v_\theta(s_t)$ to predict the actual game outcome z_t.
- A cross-entropy term $-\boldsymbol{\pi}_t^T \log \mathbf{p}_\theta(s_t)$ for the policy head, encouraging the network's policy $\mathbf{p}_\theta(s_t)$ to match the MCTS search policy $\boldsymbol{\pi}_t$.
- An L2 weight regularization term $c||\theta||^2$ to prevent overfitting.

3.3 Contextualization and Critique

AlphaZero marked a significant advance by demonstrating that superhuman performance could be achieved without human data, relying purely on self-play and a general Reinforcement Learning algorithm. This addressed a major limitation of AlphaGo. The unification into a single network and removal of fast rollouts simplified the system.

Strengths:

- **Tabula Rasa Learning:** Achieved superior performance without any human data, demonstrating true self-learning.
- **Generality:** The same algorithm successfully mastered Go, Chess, and Shogi, highlighting its broad applicability to different perfect information games.
- **Improved Performance and Simplicity:** Outperformed AlphaGo and had a more elegant, streamlined architecture and training process.
- **State-of-the-Art MCTS:** The integration of the policy prior directly into the UCB calculation (PUCT) became a standard in subsequent MCTS-based agents.

Limitations:

- **Requires Perfect Simulator:** Like AlphaGo, AlphaZero's MCTS still required a perfect model of the environment (i.e., the game rules) to provide legal moves and transition between states within the search tree. This limited its direct application to domains where such a simulator is unavailable.

- **Computational Resources:** Training AlphaZero required substantial computational resources for the extensive self-play and MCTS computations. For example, the original AlphaZero run used 5,000 first-generation TPUs for generating self-play games and 64 second-generation TPUs for training the neural networks.

- **Zero-Sum Games:** The algorithm was primarily demonstrated on two-player, zero-sum games with perfect information.

AlphaZero's success fueled further research into general game playing and self-learning systems, setting the stage for algorithms that could operate even without explicit knowledge of environment dynamics.

4 MuZero: Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

MuZero [3] took the principles of AlphaZero a step further by removing the requirement of knowing the environment's rules or having access to a perfect simulator. It achieved this by learning its own model of the environment dynamics, enabling it to plan in a learned latent space. This allowed MuZero to master not only board games like Go, Chess, and Shogi, but also visually complex Atari games, all without prior knowledge of their rules.

4.1 Approach Description

The core innovation of MuZero is its ability to learn a model that predicts only the aspects of the environment relevant for planning: the policy, value, and reward. It consists of three main neural network components, typically part of a single, larger network:

1. **Representation Function (h_θ):** This function takes a sequence of past observations $o_1, \ldots, o_t$ (e.g., screen pixels in Atari, board configurations in Go) and maps them to an initial hidden state $s^0 = h_\theta(o_1, \ldots, o_t)$. This hidden state serves as the starting point for planning.

2. **Dynamics Function (g_θ):** This function models the environment's transitions in the learned latent space. Given a previous hidden state s^{k-1} and a hypothetical action a^k to take from that state, it predicts the next hidden state s^k and the immediate reward r^k: $(r^k, s^k) = g_\theta(s^{k-1}, a^k)$.

3. **Prediction Function (f_θ):** Similar to AlphaZero's network, this function takes a hidden state s^k as input and outputs a policy $\mathbf{p}^k$ (probabilities for actions) and a value v^k (predicted future cumulative discounted reward): $(\mathbf{p}^k, v^k) = f_\theta(s^k)$.

MCTS with the Learned Model: MuZero performs MCTS by "unrolling" its learned model.

- Starting from an initial hidden state s^0 (derived from real observations via h_θ), the MCTS explores sequences of hypothetical future actions $a^1, a^2, \ldots$.

- At each step k in a simulation path within the MCTS tree, if the agent considers taking action a^k from state s^{k-1}:

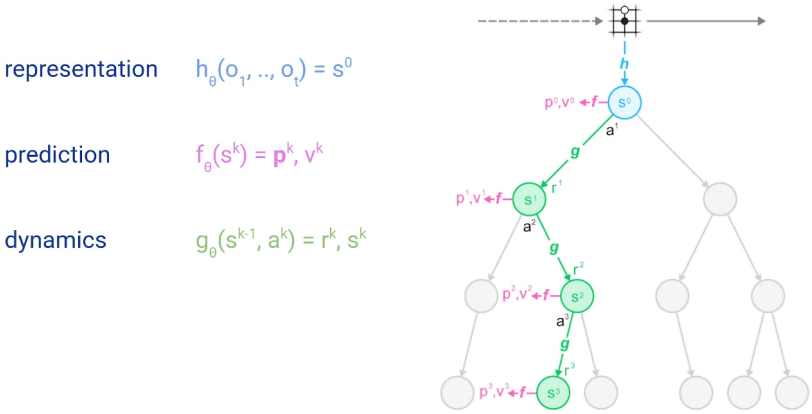

Figure 3: MuZero's architecture, showing the representation function h_θ, dynamics function g_θ, and prediction function f_θ. The MCTS operates in the learned latent space, using these functions to simulate future states and rewards (Image source: link).

  - The dynamics function g_θ is used to predict the next hidden state s^k and reward r^k.
  - The prediction function f_θ is then used on s^k to get the policy $\mathbf{p}^k$ (used as priors for expanding from s^k) and the value v^k (used for evaluation if s^k is a leaf node in this simulation path).

- The MCTS selection rule, expansion, and backup are similar to AlphaZero's PUCT-based search, but operate entirely within the space of learned hidden states and use the predicted rewards r^k, policies $\mathbf{p}^k$, and values v^k.

Training: The three functions $(h_\theta, g_\theta, f_\theta)$ are trained jointly and end-to-end. The goal is to make the predictions of the learned model consistent with observed data and MCTS-improved targets. For a trajectory segment, the overall loss $l_t(\theta)$ at time t considers K steps of unrolling the model:

$$l_t(\theta) = \sum_{k=0}^{K} \left[l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\boldsymbol{\pi}_{t+k}, \mathbf{p}_t^k) \right] + c||\theta||^2$$

where:

- u_{t+k} is the true observed reward from the environment at step $t+k$. r_t^k is the reward predicted by g_θ for the k-th step in the imagined trajectory starting from observations up to t. l^r is the reward prediction loss.

- z_{t+k} is the target value for the state reached after k imagined steps. For board games, this is the final game outcome. For Atari, this is typically an N-step bootstrapped return: $z_{t+k} = \sum_{i=0}^{N-1} \gamma^i u_{t+k+i} + \gamma^N v_{\text{MCTS}}(s_{t+k+N})$, where v_{MCTS} is the MCTS-derived value of the state N steps into the future. v_t^k is the value predicted by f_θ. l^v is the value prediction loss.

- $\boldsymbol{\pi}_{t+k}$ is the MCTS-improved policy target for the state reached after k imagined steps. $\mathbf{p}_t^k$ is the policy predicted by f_θ. l^p is the policy prediction loss.

- $c||\theta||^2$ is an L2 regularization term.

This loss trains the representation function h_θ to produce a useful initial state, the dynamics function g_θ to accurately predict future states (in terms of their value/policy/reward conse-

quences) and rewards, and the prediction function f_θ to predict policies and values consistent with search and actual outcomes.

4.2 Contextualization and Critique

MuZero's primary contribution is its ability to perform effective planning without knowing the rules of the environment, a significant step towards general AI.

Strengths:

- **No Game Rules Required:** MuZero learns its own model of dynamics, making it applicable to environments where rules are unknown or too complex to specify, including visually rich Atari games.
- **State-of-the-Art Performance Across Diverse Domains:** Demonstrated superhuman performance in Go, Chess, Shogi, and achieved new SOTA in Atari, showcasing its versatility.
- **End-to-End Learning of Model and Policy/Value:** The joint training of representation, dynamics, and prediction functions allows the system to learn a model optimized for planning.

Limitations:

- **Complexity of Model Learning:** Learning an accurate and useful model of environment dynamics, even one focused only on planning-relevant quantities, is a challenging task, especially in complex or stochastic environments.
- **Potential for Model Exploitation and Compounding Errors:** Planning with a learned model can suffer if the model has inaccuracies. Errors can compound over longer planning horizons, leading to suboptimal plans. MuZero's focus on predicting value/policy/reward rather than full state reconstruction might mitigate this but doesn't eliminate it.
- **Sample Efficiency and Computational Cost:** While MuZero can learn without rules, learning an effective model and then planning with it can still be sample-intensive and computationally expensive, especially compared to highly optimized model-free methods in some domains (though MuZero Reanalyze improved sample efficiency significantly in Atari).
- **Focus on Deterministic Dynamics in Original Paper:** The original MuZero paper focused on environments with deterministic dynamics or where the stochasticity could be implicitly captured. Extending learned models to explicitly handle significant stochasticity remains a challenge.

MuZero represents a powerful fusion of model-based and model-free RL ideas, pushing the boundaries of what can be achieved with learned world models and tree search.

5 Comparative Analysis and Evolution

The journey from AlphaGo to AlphaZero, and then to MuZero, showcases a clear evolutionary path towards more general and autonomous Reinforcement Learning agents.

- **AlphaGo to AlphaZero:** The primary shift was the elimination of reliance on human expert data and the move to pure self-play. Architecturally, this involved unifying the

separate policy and value networks into a single network and removing the fast rollout policy from MCTS. This simplification led to a more general algorithm applicable to multiple games (Chess, Shogi, Go) with minimal changes, at the cost of requiring more computation for initial learning from scratch.

- **AlphaZero to MuZero:** The most significant evolution was the removal of the requirement for a perfect simulator or knowledge of game rules. MuZero achieved this by learning its own internal model of the environment's dynamics, specifically tailored for planning. This involved introducing explicit representation, dynamics, and prediction functions. While adding complexity in terms of what needs to be learned (the model itself), it granted the system unprecedented generality, allowing it to tackle environments like Atari where rules are not easily provided.

Common Threads: Despite their differences, all three algorithms share core principles:

1. **Monte Carlo Tree Search (MCTS):** MCTS remains the central planning algorithm, providing a powerful way to explore the state space and improve policy estimates.

2. **Deep Neural Networks:** Deep learning is used for powerful function approximation, learning policy distributions, value functions, and, in MuZero's case, environment models.

3. **Self-Play:** Self-play (or interaction with the environment in MuZero's Atari case) is the primary mechanism for generating data and driving the learning process, allowing the agents to continuously improve beyond fixed datasets.

The evolution demonstrates a clear trend: reducing external dependencies (human data, explicit rules) and increasing the scope of what the agent learns internally. Each step traded some form of prior knowledge or architectural simplicity for greater autonomy and generality.

6 Critical Evaluation and Broader Context

The AlphaGo family has profoundly impacted the field of Reinforcement Learning, demonstrating capabilities previously thought to be decades away. Their success has spurred immense interest and research in deep RL and planning.

Strengths of the Research Line:

- **Achieving Superhuman Performance:** Consistently reached or surpassed human expert levels in highly complex strategic games.

- **Pushing Algorithmic Boundaries:** Introduced novel combinations of search, deep learning, and RL principles, leading to powerful, generalizable frameworks.

- **Demonstrating End-to-End Learning:** Showcased the potential of learning complex behaviors and even environment models directly from raw inputs and reward signals.

Enduring Limitations and Challenges:

- **Computational Expense:** Training these systems requires massive computational resources (e.g., thousands of TPUs/GPUs, extensive self-play), as highlighted for AlphaZero. This makes them inaccessible for many researchers and impractical for many real-world applications without significant hardware investment.

- **Sample Efficiency:** While impressive, the sample efficiency (amount of experience needed

to learn) can still be a concern, particularly for MuZero when learning complex models from scratch. Model-free methods can sometimes be more sample-efficient in domains where a good model is very hard to learn.

- **Applicability Beyond Constrained Domains:**
  - **Perfect Information & Observability:** These algorithms were primarily developed for perfect information games where the full state is known. Extending them robustly to partially observable environments or games with hidden information (beyond what MuZero's history-based representation handles) is an ongoing research area.
  - **Beyond Zero-Sum Games:** While the MCTS framework can be adapted, their demonstration in multi-agent settings has largely been in two-player zero-sum contexts. Generalizing to complex multi-agent cooperation or non-zero-sum competition presents further challenges.
  - **Real-World Interaction:** Applying these methods to real-world robotics or other physical systems involves tackling issues like continuous state/action spaces, noisy sensors/actuators, and safety constraints, which are not primary concerns in simulated game environments.
- **Interpretability:** The learned representations within the neural networks, especially MuZero's hidden states, are often "black boxes," making it difficult to understand precisely what the agent has learned or why it makes certain decisions.
- **Exploration in Vast State Spaces:** While MCTS inherently performs exploration, ensuring effective and efficient exploration in exceptionally large or sparse-reward environments remains a fundamental RL challenge that these systems grapple with, often relying on the sheer scale of search.

These systems, while groundbreaking, operate within relatively well-defined, albeit complex, environments. The transition to messier, less structured real-world problems remains a significant hurdle for RL, one that future iterations or new paradigms will need to address.

7 Conclusion

The AlphaGo, AlphaZero, and MuZero algorithms represent a remarkable progression in the field of Artificial Intelligence and Reinforcement Learning. From AlphaGo's initial reliance on human data to defeat a Go champion, to AlphaZero's mastery of multiple games through pure self-play, and culminating in MuZero's ability to learn environment models from scratch, this lineage has systematically reduced reliance on prior human knowledge and domain-specific engineering.

These systems have powerfully demonstrated the efficacy of combining deep neural networks for rich function approximation with Monte Carlo Tree Search for robust planning, all driven by Reinforcement Learning from self-generated experience. While their computational demands and current applicability largely to perfect-information, discrete-action domains highlight ongoing challenges, their success has irrevocably advanced the state of the art. The AlphaGo family not only solved grand challenges but also provided a conceptual framework and a set of powerful tools that continue to inspire and inform the development of more general, autonomous, and capable AI systems. Future work will likely focus on enhancing sample efficiency, scaling to more complex and partially observable environments, and bridging the gap between superhuman game play and effective real-world agency.

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.