AlphaZero

In our quest to push the boundaries of game playing AI, we embarked on a deep dive into three ConnectX agents: the AlphaZero based self-play agent (My implementation) and two prominent baselines: SergeyP's PROBS agent and the "KDB" workshop grandmaster. My analysis tries to leverage the provided source code and current literature in reinforcement learning (RL) to diagnose the AlphaZero agent's design and identify the factors contributing to its underperformance. A systematic comparison of architectures and design patterns across all three systems is conducted, contrasting approaches such as Monte Carlo Tree Search (MCTS) with PUCT, beam search, and heuristic search, as well as joint versus separate network configurations. Each contrast is substantiated with evidence from the code. The report addresses key questions concerning design choices, performance prospects, and overarching goals of RL based systems. All discussions are evidence backed and grounded in state of the art RL practices.

1 Diagnostic Findings on the AlphaZero Agent

An examination of the original agent logs and code reveals several key shortcomings in my AlphaZero implementation.

A primary concern is the persistence of high training losses. I noticed that even after 15 iterations, the policy loss remained in the range of 1.63-1.96, and the value loss hovered around 0.85-1.02. These values are far from converged; a well trained AlphaZero agent on a small board typically achieves losses below 0.5. Such consistently high losses strongly pointed me that the network is underfitting the data, likely due to an insufficient quantity of training data, too few training iterations, or issues with the learning rate and other hyperparameters. The neural network is trained using the following combined loss function:

$$L = (z - v)^2 - \pi^T \log p + c\|\theta\|^2$$

In our implementation, the L2 regularization term $c\|\theta\|^2$ is not explicitly added to the loss within the training loop but is instead enforced by the Adam optimizer's `weight_decay=1e-4` parameter. This setting applies a penalty proportional to the squared magnitude of each weight on every gradient update, which is mathematically equivalent to including $c\|\theta\|^2$ in the loss function. By relying on optimizer-level weight decay, we simplify our loss computation while still benefitting from the regularizing effects that prevent overfitting and constrain parameter growth. Empirically, this approach has been shown to stabilize training without altering the core optimization steps.

When I pit my agent's win rate against a random opponent also showed stagnation, oscillating between 0.80 and 1.00, with an average of approximately 0.95. Furthermore, it experienced losses even in self play games, evidenced by its around 150th rank range with around 300 points. This plateauing performance indicated me that the model reached a capacity limit given the available training data, struggling to learn more complex strategies or adapt to diverse game states.

The replay buffer dynamics also contribute to the agent's limitations. The code utilizes a sliding replay buffer with a maximum size of 1000 samples, which is significantly smaller than the 100,000+ samples typically used in full scale AlphaZero implementations. With only 30 new games generated per iteration and older data being replaced, my network predominantly encounters fresh, but often similar position, games. This limited and constantly refreshed data pool can lead to training oscillations or catastrophic forgetting of previously learned tactics. A larger or prioritized replay buffer would likely stabilize training and improve long term learning.

Regarding my network capacity and symmetry, the agent's network is moderately sized, featuring 3 blocks and 64 filters. While ConnectX is simpler than games like chess, this architecture may be adequate. However, a significant omission is the lack of data augmentation. ConnectX possesses horizontal symmetry, meaning that mirroring states could effectively double the training data. Without this, my network may fail to learn symmetry generalizations, thereby underutilizing its representational capacity.

The MCTS parameters also present issues. Only 50 simulations are performed per move. I quickly saw that was too few to build depth or stable statistics. The default PUCT `c` value of 1.25 might be too low for a wide, short game like ConnectX, potentially leading to an over exploitation of prior probabilities and insufficient exploration. Additionally, I also watched that the temperature parameter decays rapidly (from 1.0 to 0.49 over 15 iterations). This rapid decay means that early moves quickly become deterministic, which can reduce exploration and limit the variety of self play games generated, thereby hindering the discovery of new strategies.

The loss curves further indicated me a problem with policy collapse or rigid targets. The policy loss showed minimal change, hovering around 1.7. This suggested me that the improved MCTS targets (derived from visit counts) are not significantly diverging from the network's prior policy output. This can be a symptom of weak exploration or redundant self play, where the agent repeatedly plays similar opening moves, thereby limiting the scope for new learning and strategic improvement.

Finally, I realized I had skipped any explicit calibration analysis for the value head. It is plausible that the value network is poorly calibrated, consistently predicting outcomes near zero, which aligns with the observed value loss of approximately 0.9. Without proper calibration, the MCTS backup values, which rely on these predictions, may be biased, leading to suboptimal search decisions.

2 Responses to Key Questions

This section provides a structured response to the key questions posed regarding the comparative analysis of the ConnectX agents.

Q1. Parallel Assessment of My Work vs KDB vs SergeyP (PROBS)

My AlphaZero agent is a faithful, albeit stripped down, reproduction of the AlphaZero framework, constrained by available computational resources and data. It relies on pure self play MCTS with a compact, joint policy and value network. In contrast, SergeyP's PROBS agent introduces a fundamental architectural shift by employing beam search instead of MCTS and utilizing separate value and Q networks. This design allows PROBS to specialize each model for its task and learn effectively from limited deep searches. Empirically, SergeyP's agent achieved top ranks (around 1300 Elo) with relatively small beam sizes, which strongly suggests robust policy learning capabilities. The KDB (dott) baseline, on the other hand, appears to use heuristic or machine learning based move prediction, specifically LightGBM on engineered features, without any deep search component. This approach facilitates rapid learning of obvious tactical patterns but lacks the capacity for deep strategic planning.

In summary, my AlphaZero agent prioritizes a principled search approach over domain specific knowledge, but its performance is limited by under training. PROBS sacrifices the uniformity of a single network for depth limited search learning, offering a powerful yet more complex engineering overhead. KDB completely foregoes deep planning in favor of speed and simplicity. Each agent embodies distinct trade offs: AlphaZero is theoretically powerful but currently undertrained; PROBS is powerful but demands more intricate engineering; and KDB is lightweight and fast but strategically less profound.

Q2. Evaluation of My Own Work

The current implementation of my AlphaZero agent is structurally correct, accurately implementing MCTS+PUCT and a residual network. However, its empirical performance is insufficient due to shallow training. The loss curves and win rates clearly indicate underfitting and a plateauing performance. Key issues include an inadequate number of self play games, the absence of data augmentation, and a lack of strategic heuristics. In its present form, the agent struggles to reliably defeat even simple baselines. Qualitatively, it successfully tries to demonstrate the core AlphaZero concept of learning from self play but critically lacks the computational budget necessary to achieve competitive strength. While substantial enhancements, as proposed, hold the potential for significant improvement, the agent currently underperforms both the PROBS agent and, likely, even the KDB baseline in competitive play.

Q3. Assessment of SergeyP's PROBS

SergeyP's PROBS agent represents an innovative and effective approach. Its use of beam search combined with dual networks is novel for ConnectX and has reportedly achieved top 5 rankings. PROBS constructs a predictive search mechanism, which can be conceptually likened to a limited depth minimax search guided by learned functions. Its strength lies in the observation that even a small beam, significantly shorter than the full game length, can still yield non trivial lookahead capabilities. Furthermore, training the policy network on the outcomes of these beam searches allows the agent to generalize this depth without the need for costly repeated searches during inference. According to the associated paper, PROBS consistently improved its win rate with increased computational resources. This suggests that SergeyP's implementation is robust and well tuned. In essence, PROBS is a powerful yet effective system, requiring considerable offline training and two distinct models. However, it leverages search more effectively than a raw AlphaZero agent of comparable scale in this specific setting, as evidenced by its strong leaderboard dominance.

Q4. Performance Prospects (500-700 points)

Achieving an Elo rating of 500-700 points with only marginal increases in resources for the current AlphaZero agent may be highly improbable. The agent currently scores around -300 points at the 150th rank. A jump to 700+ Elo, which represents the near top tier in such competitions, would typically necessitate significantly deeper search capabilities or massively more extensive training. Even a 5-10x increase in the simulation budget or GPU training time (e.g., training 5x more games) would likely leave the agent far from a truly well trained state. In general practice(not always fixed), top Kaggle scores are often achieved through the integration of strong heuristics or extensive supervised pretraining using expert games, going beyond pure self play. Without fundamentally altering the agent's approach, such as incorporating the proposed heuristics or adopting a PROBS like hybrid strategy, simply scaling up the current design will be insufficient. Reaching 500-700 Elo points I think would likely demand a more sample efficient algorithm or the integration of engineered knowledge, which the current design fundamentally lacks.

These proposed changes align with contemporary literature, I have tried present below in my references section, such as MuZero's emphasis on learned environment models and EfficientZero's focus on efficient data usage. The inclusion of heuristic rules, while not a feature of classic AlphaZero, is a pragmatic adaptation for competitive environments like Kaggle, where several top submissions incorporate such pruning strategies.

Q5. Main Goal of RL based Works ("Cluster vs Memorize")

The primary objective of the reinforcement learning (RL) submissions discussed, including AlphaZero and PROBS, is to learn strategic policies through self play, rather than merely memorizing fixed sequences of moves. AlphaZero style methods inherently cluster states based on similarities in their learned representations, but their ultimate goal is to optimize a policy that generalizes well to unseen states. The phrasing "cluster of games" suggests the construction of a search tree or graph to identify winning paths, a process implicitly tried to performed by both AlphaZero's MCTS and PROBS's beam search.

In contrast, KDB's learned baseline, utilizing LightGBM, is closer to memorizing patterns observed in its training data. Therefore, the overarching goal of these RL submissions is interpreted as generalization through learned planning that is, discovering the underlying principles and optimal strategies of ConnectX rather than rote memorization of specific sequences. In practice, all these agents implicitly build extensive "clusters" of connected game states during their learning phases, facilitated by their neural features. However, PROBS and AlphaZero explicitly explore these game trees. Consequently, their objective is to achieve strategic mastery through search driven training, extending beyond simply "learning the openings" or relying on fixed lookup tables.

References

- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815.

- Pastukhov, S. (2024). Playing Board Games with the Predict Results of Beam Search Algorithm. arXiv:2404.16072.

- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Graepel, T., Lillicrap, T., & Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. arXiv:1911.08265.

- DeepMind. (2020). MuZero: Mastering Go, chess, shogi and Atari without rules. https://www.deepmind.com/blog/muzero-mastering-go-chess-shogi-and-atari-without-rules.

- Plaat, A., Wang, H., & Emmerich, M. (2019). Policy or Value? Loss Function and Playing Strength in AlphaZero. In *Proceedings of the 24th Conference on Games* (CoG 2019).

  https://liacs.leidenuniv.nl/ plaata1/papers/CoG2019.pdf.

These sources tries to describe the core methods (PUCT-MCTS, beam search with dual networks, and learned planning) that underpin the agents compared in my report. All conclusions tried to drawn are based on these fundamental principles.