CALPHAZERO: A REINFORCEMENT-LEARNING SUDOKU SOLVER WITH CURRICULUM SCHEDULING

ARUNIM AGARWAL, ZIRUN HAN, ALEXANDER KYIMPOPKIN, LUCAS KLEIN

ABSTRACT. We propose *cAlphaZero*, a reinforcement-learning and MCTS-based method inspired by AlphaZero to solve Sudoku as a step towards solving more general NP-complete problems with learning-based approaches. We evaluate *cAlphaZero* on 4×4 Sudoku to validate the performance of our method. We find that the use of curriculum scheduling is key to achieving excellent model performance and stability during the training process. We also propose several additional modifications to the AlphaZero algorithm to adapt it for puzzle solving. *cAlphaZero* can solve the 4×4 puzzles with a high degree of accuracy using only a few MCTS simulation steps.

1. INTRODUCTION

When generalized to an $n \times n$ grid, Sudoku is an NP-complete problem [24]. This class of problems is most classically associated with problems such as Boolean satisfiability (SAT) and the Travelling salesman problem. Prior work in Sudoku solvers has primarily relied on classical approaches, as well as a handful of modern learning-based approaches. [10]. Prior literature has shown the ability to reformulate any NP-complete problem as a decision problem and thus a Boolean satisfiability problem.

Being an NP-complete problem, algorithms for solving $n \times n$ Sudoku are exponential with respect to n and the run time quickly blows up with large Sudoku problems. Machine learning methods have been used in the past to approximately solve computationally intractable problems such as Go, molecule generation and data compression. [19][2][23] In this paper, we show that neural networks can be effective at solving small Sudoku problems. We attempt to train two seperate models inspired by AlphaZero and GFlowNets respectively to solve Sudoku problems.

1.1. **Contributions.**

We propose a neural network-accelerated Monte Carlo Tree Search (MCTS) method inspired by the AlphaZero algorithm to solve puzzles. We find that this algorithm can consistently solve 4×4 Sudoku games. We present several improvements to the canonical training regimen to adapt the algorithm for the single-player task of puzzle solving. We show that these modifications, including the use of entropy loss to optimize prediction confidence and scheduling the difficulty of training puzzles, enhance training stability and network performance. We additionally experimented with Generative Flow Network (GFlowNet) based approaches [3][7][25][26] but did not find them to be effective in solving puzzles.

2. BACKGROUND

2.1. **Monte Carlo Tree Search Algorithms.**

Monte Carlo Tree Search (MCTS) is an algorithm used for decision processes and is used in game-theoretic applications to solve the game tree. MCTS works by expanding a search tree one node at a time, where each node represents a state in the game and each branch represents an action by which the game transitions to a new state.

AlphaZero[19][18], an game-playing approach built on MCTS and trains solely through self-play without human knowledge, maintains

- $Q(s, a)$: the expected reward of a state-action pair (s, a)
- $N(s, a)$: the number of times action a has been searched from state s
- $P(s, a)$: the initial probability of taking action a from state s returned from the model's policy

$Q(s, a)$ and $P(s, a)$ are estimated by a neural network with a value head and policy head respectively. AlphaZero uses a variant of the PUCT algorithm[15] to determine the upper confidence bound (UCB) U

$$U(s, a) = Q(s, a) + c_{puct} P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{1}$$

$$a_t = \underset{a}{\operatorname{argmax}}\ U(s, a) \tag{2}$$

and thus select actions from each node, where c_{puct} is a hyperparameter that is used to balance exploration and exploitation.

Our approach slightly modifies this UCB score by introducing a small additive constant (in practice, 0.5) to the UCB score for any action that has not yet been explored. We have found that this is a simple but effective way to prevent an overconfident policy network from inducing poor MCTS searches and overfitting to these poor searches in the early stages of the curriculum-scheduled training process (e.g. 2-blank puzzles) when the value net is relatively unimportant.

2.2. **GFlowNets.** Flow networks consist of a directed graph with sources, sinks, intermediate nodes and edges carrying some amount of flow between them. Flow networks can be thought of like network of pipes with water flowing through them. Each sink, x represents a terminal state has an "out-flow", $R(x)$ which represents the reward of that terminal state. For our purposes there will be a singular source s_0 which is the initial state. We will denote $F(s,a) = f(s,s')$ as the flow between states s and s' where $T(s,a) = s'$ is the state reached by taking action a from state s. Using this formulation of actions and states one can obtain a Markov Decision Process (MDP) with the policy

$$P(s,a) = \frac{F(s,a)}{\sum_{a'} F(s,a')} \tag{3}$$

where following policy P from s_0 leads to terminal state x with probability $R(x)$. [3]

Compared to other methods GFlowNets generate more diverse trajectories because they generate trajectories with probability proportional to reward compared to methods which try to maximize reward.

3. Related Work

Sudoku is a game with a rich history [22] and myriad algorithmic approaches as well as many formulations. [12]

3.1. **Sudoku Solvers.** Numerous Sudoku solvers based on backtracking [4][13] exist, with recent [9] [14][20] literature showing ongoing work in this area towards establishing a baseline. Recent approaches in Sudoku have concentrated on classical approaches using a mixture of different approaches, ranging from graph theory[1] to alternating projections[5] and extensions to higher dimensions [20]. These approaches have shown significant promise in rapidly solving classical 9×9 Sudoku problems.

3.2. **Reinforcement Learning.** Prior literature in reinforcement learning has shown the efficacy of curriculum scheduling [11], suggesting its efficacy in solving challenging problems. More recent advancements in the field [19] have highlighted the efficacy of MCTS-based approaches, achieving superhuman performances in combinatorially challenging games such as Chess, Go, and Shogi. These advancements have been further expanded [16] to generalize to games with unknown rule structures. Work in MCTS-based puzzle solvers has made strides, using computational approaches [8], and showing substantial efficacy in solving a broad class of puzzles using probabilistic searches.

4. Approach

4.1. **Motivation for NN-Accelerated MCTS.** Two computationally expensive aspects of a traditional MCTS algorithm are that the game must be "rolled out" to an end state for a reward to be assigned and that a large number of nodes must be explored to ensure that the empirical selection policy is accurate. In our algorithm, at each game state s, we instead use a neural network to predict a policy π_s and an evaluation v_s. As such, when we conduct an MCTS search at s, π_s is used as the prior distribution to "hot start" the search process by focusing the search on the most promising actions. Additionally, instead of rolling out to an end state, we can assign $v_{s'}$, the expected reward, when we encounter an unexplored node s' in the game tree. Theoretically and empirically, this guided search process reduces the computation required to solve the puzzle.

4.2. **Learning Process.** The learning process we used can be broken down into two main components: game generation and network training. During game generation, a large number of games are played using the neural-MCTS algorithm. Each game generates a chain of data in the form of (s, p_s, r) where s is a board position, p_s is the empirical policy after conducting a number of MCTS simulations, and r is the reward at the end state of the chain of actions selected by the algorithm. For our algorithm, we chose to use a constraint satisfaction reward with the form

$$r(s^{\mathrm{T}}) = \frac{1}{N} \sum_{i=0}^{N} c_i(s^{\mathrm{T}}) \tag{4}$$

where $c_i \in \{0, 1\}$ is an indicator function for the ith constraint for a solved game. The game is completely solved when the reward for the terminal state, $r(s^{\mathrm{T}}) = 1$. For $n \times n$ Sudoku, there are $3n$ total constraints for the rows, columns, and subgrids.

After each game is played, we exploit the symmetries inherent to Sudoku to augment the data, including rotations of the board, permutations between rows within their row-group, permutations of entire row-groups. We also partially exploit the symmetry of permuting the numbers associated with each square, but as the board size n increases, using all such permutations becomes increasingly intractable by the combinatorial explosion of such transformations. The empirical policies are also transformed correspondingly, while the rewards are held constant, to generate our augmented (s, p_s, r) tuples. Through this process, we're able to massively increase the data available to train our policy and value network, without needing to play an intractable number of games.

The generated and augmented data is then fed into the neural network which is trained on the loss function

$$\ell(p_s, \pi_s, r, v_s) = \lambda(r - v_s)^2 - p_s^\top \log(\pi_s) + \xi \pi_s^\top \log \pi_s \tag{5}$$

where π_s and v_s are the network-predicted policy and value respectively. Comparing this to the AlphaZero loss:

$$\ell_{AZ}(p_s, \pi_s, r, v_s) = (r - v_s)^2 - p_s^\top \log(\pi_s) \tag{6}$$

we have added a scaling factor $\lambda = 100$ to balance the magnitude of the value and policy loss. Additionally, we have added a third term that is equal to the entropy of the network policy scaled by a factor ξ. The motivation for adding this term is two-fold. First, this stabilizes the training process at the beginning of our runs, as we will discuss in the next section. Further, this can be used to control the network's confidence at different stages of the game. A high policy entropy corresponds to low confidence and more diverse searching, which is optimal for the early game where there are many correct moves. On the other hand, the network should be confident near the endgame for efficiency.

4.3. **Curriculum Scheduling.** A major component of our training process is the scheduling of puzzle difficulties. Training directly on full-difficulty Sudoku puzzles resulted in the network failing to learn a meaningful policy.

Drawing on existing work in curriculum learning and reinforcement learning curriculum schedules, we introduced a curriculum schedule detailed in Training Algorithm. Specifically in the case of Sudoku, intuitively, models $\{\mathcal{M}_{n+1}\}$ which can solve $n+1$-blank Sudoku puzzles are a subset of models $\{\mathcal{M}_n\}$ able to solve n-blank Sudoku puzzles. This is visualized in 1.

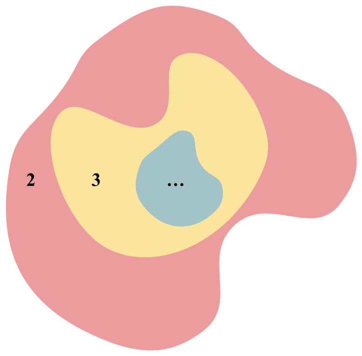


FIGURE 1. Regions of model space which are able to solve puzzles of increasing difficulty, in our case, increasing numbers of blanks.

Our curriculum gradually ramps the difficulty, beginning with two-blank puzzles and increasing the number of blank squares by one once the model has attained a specific accuracy or trained for a maximum number of iterations.

MCTS lends itself well to this curriculum schedule, since the process of MCTS naturally provides a gradual transition when we increase the number of blanks. We can see this by considering the n-blank game to be a child node of many $(n+1)$-blank games which are one square removed, and noting that when MCTS reaches the next layer, there are only n blanks. As such, the covariate shift the network experiences is gradual, improving the stability of the training process. We suggest that this behavior is caused by the size of the Rashomon Set[17] shrinking substantially for each increase in the number of blanks.

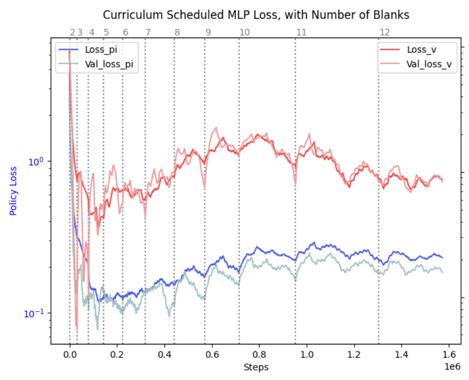
5. Experimental Results

In the section below, we plot and detail the results of our training process using *cAlphaZero* on 4×4 Sudoku games on multiple architectures, and we present a negative result of poor training convergence without the use of curriculum scheduling.

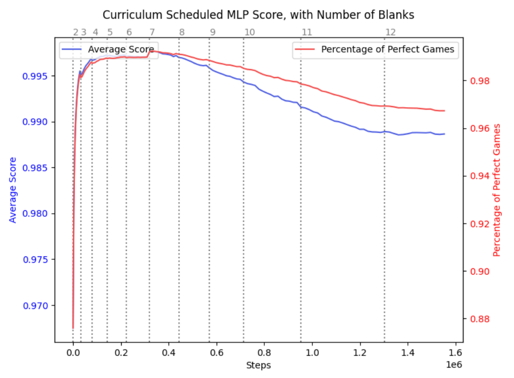
We can now see the effect of our previously mentioned change in the size of the Rashomon set associated with our problem over the course of the curriculum. This set, defined to be the set of almost-equally-accurate models, is largest for the simplest problem of filling in two blanks and much smaller for each subsequent model. Beginning the training process with two-blank games therefore presents a large target that is easy to find. We see this in the sharp drop in loss and corresponding increase in accuracy in the first steps of Figure 2b. Then, as blanks are added in, we see the losses hold steady on the order of 0.1 and 0.001, while the accuracy stays clear of 95%. The training run without any curriculum scheduling, on the other hand, provides a stark contrast and a strong indication of a negative result. This test was run on a modest difficulty of 8 blanks, and as seen in Figure 2c and 2d, the loss initially drops, but hovers about an order of magnitude above the losses with curriculum scheduling. Even the initial drop in loss is deceiving, as the limiting policy loss of 4.1 roughly corresponds to predicting $P(s,a) = \frac{1}{\text{\# actions}}$, i.e. random policy. Following the analogy of decreasing size of solvers in model space, a training process without curriculum scheduling is unable to locate the small space of models able to make progress on 8-blank puzzles, let alone solve them fully. Iteratively guiding the process towards the correct area in model space, however, empirically is shown to work.

The validation losses plotted in Figures 2a, 2c are evaluated for the neural network with respect to a holdout set of generated game data after each epoch. We do not provide overall results on "test" data, as the augmentation process can transform boards in the training dataset to boards in the validation set, making the construction of a holdout set a bit more intensive. Despite these constraints, we argue that our model is generalizing by observing a few key points. First, the validation metrics on the holdout data after each epoch closely track the training loss, indicating that during each epoch, we are not overfitting to the provided data. Second, we can observe the drop in accuracy at each increase in number of blanks. By decreasing the number of blanks, we induce a shift in the overall distribution, and we find that the percentage of perfect games stays above 90%, indicating generalization.

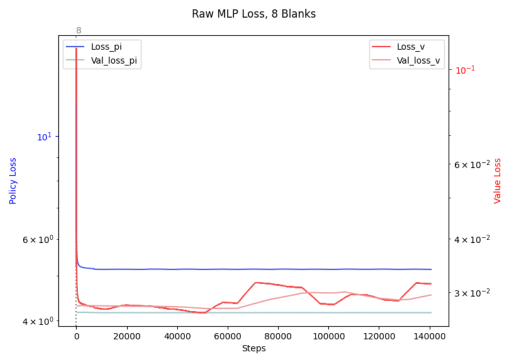
We provide two experimental training runs with self-attention-based architectures with learnable position encoding. This is motivated by the ability of self-attention layers to capture long-range dependencies while using significant weight sharing compared to MLPs. Empirically, we find that the smaller of these models (80% smaller than the MLP) achieves similar losses but struggles towards the end of training, while the larger model (22% smaller) has no trouble maintaining low loss throughout training. See corresponding loss and score curves in Figure 4 in the Appendix.



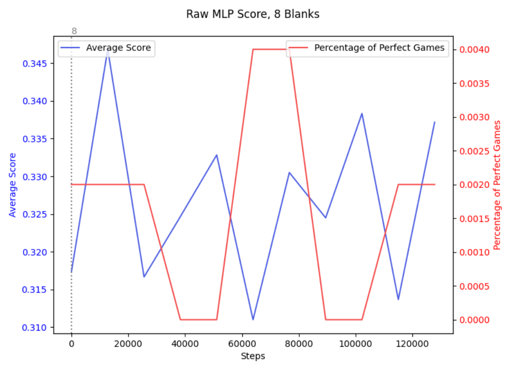
(A) Policy and value training loss curves for an MLP architecture using *cAlphaZero*



(B) Associated score metrics during the *cAlphaZero* training run



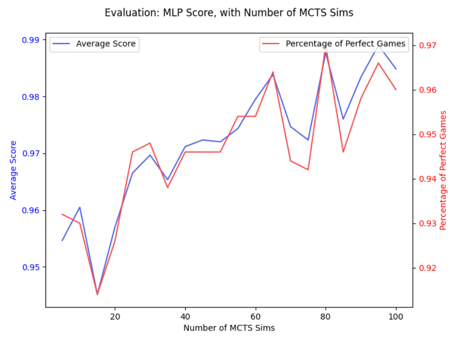
(C) Policy and value training loss curves without curriculum scheduling



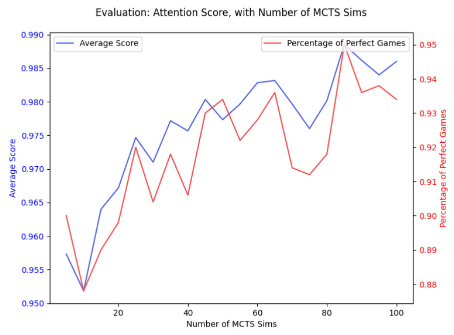
(D) Associated score metrics during the same training run without curriculum scheduling

FIGURE 2. Comparative analysis of MLP architecture in *cAlphaZero* training with and without curriculum scheduling.

We find that GFlowNet-based methods do not learn to solve Sudoku puzzles within a reasonable time. We experimented with both detailed balance and trajectory balance losses in order to address this challenge. [21]. The



(A) Evaluation of the MLP architecture using *cAlphaZero* after the training shown in Figure 2a



(B) Evaluation of the attention architecture with hidden layer size 512 using *cAlphaZero* corresponding to Figure 4c

FIGURE 3. Post-training evaluation of MLP and attention-based architectures using *cAlphaZero*

model learned some aspects of the 9x9 puzzles, reducing the trajectory balance loss from 55.76 to 19.029 after 5000 episodes, but failed to materially improve on those results. Given our observed successes with curriculum scheduling in *cAlphaZero*, a curriculum scheduled GFlowNet may be able to substanially improve on these results.

6. DISCUSSION

We find that our AlphaZero [19] based approach to learning solutions to Sudoku problems is able to consistently solve small problems when presented with a scheduled curriculum, drawing on prior literature in curriculum scheduling [21]. We additionally attempted to solve Sudoku puzzles with gflownets [3], but consistently encountered challenges around sample efficiency as well as small models inability to converge to a meaningfully low error.

Although we only experimented with single-try runs (i.e. no explicit backtracking), the algorithm can be easily adapted to deterministically obtain the answer using many runs. To do this, we perform a single run, and if the end state does not solve the puzzle, we set the associated UCB score to zero and return to the root node of the search tree.

7. FUTURE WORK

For this work, we focused on smaller 4×4 puzzles in our reinforcement learning-based approach due to computational challenges in our reinforcement-learning-based regime. Nevertheless, the ability to generalize this algorithm to a broader context hinges on its ability to generalize to larger puzzles. We aim to extend this work to perform game generation in parallel and therefore hope to more closely focus on self-attention-based models. This is because fully connected MLPs will become computationally challenging to train as the state and action spaces scale rapidly with puzzle size. Here, we hope to test novel sub-quadratic attention approaches based on alternative paradigms such as Monarch Matrices [6] to evaluate their relative efficacy for this probabilistic puzzle-solving paradigm. While GFlowNets were not meaningfully able to solve 9×9 Sudoku problems, switching them to a curriculum-scheduled version following our promising results with *cAlphaZero* presents a direction for further exploration. We are excited about the performance of probabilistic solvers for Sudoku, and hope to extend our approach to Boolean Satisfiability (SAT) and the Traveling Salesman Problem.

References

[1] M. BATAR. "An efficient sudoku solver application based on graph theory". In: *International Journal of Innovative Engineering Applications* 5 (2 2021), pp. 218–224. DOI: 10.46460/ijiea.982908.

[2] Emmanuel Bengio et al. *Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation*. 2021. arXiv: 2106.04399 [cs.LG].

[3] Yoshua Bengio et al. *GFlowNet Foundations*. 2023. arXiv: 2111.09266 [cs.LG].

[4] Eric C. Chi and Kenneth Lange. *Techniques for Solving Sudoku Puzzles*. 2013. arXiv: 1203.2295 [math.OC].

[5] O. Eva and B. Dunka. "A comparison of three sudoku solving methods". In: *International Journal of Computer Applications* 181 (40 2019), pp. 46–52. DOI: 10.5120/ijca2019918439.

[6] Daniel Y. Fu et al. *Monarch Mixer: A Simple Sub-Quadratic GEMM-Based Architecture*. 2023. arXiv: 2310.12109 [cs.LG].

[7] Moksh Jain et al. *Biological Sequence Design with GFlowNets*. 2023. arXiv: 2203.04115 [q-bio.BM].

[8] Mohammad Sina Kiarostami et al. "On Using Monte-Carlo Tree Search to Solve Puzzles". In: *2021 7th International Conference on Computer Technology Applications*. ICCTA 2021. ACM, July 2021. DOI: 10.1145/3477911.3477915. URL: http://dx.doi.org/10.1145/3477911.3477915.

[9] A. Maji and R. Pal. "sudoku solver using minigrid based backtracking". In: (2014). DOI: 10.1109/iadcc.2014.6779291.

[10] Anav Mehta. *Reinforcement Learning For Constraint Satisfaction Game Agents (15-Puzzle, Minesweeper, 2048, and Sudoku)*. 2021. arXiv: 2102.06019 [cs.LG].

[11] S. Narvekar. "curriculum learning in reinforcement learning". In: (2017). DOI: 10.24963/ijcai.2017/757.

[12] M. Nishiara and R. Hidai. "Decoding error of sudoku for erasure channels". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E100.A (12 2017), pp. 2641–2646. DOI: 10.1587/transfun.e100.a.2641.

[13] U. Pfeiffer, T. Karnagel, and G. Scheffler. "A sudoku-solver for large puzzles using sat". In: (). DOI: 10.29007/79mc.

[14] P. Roach et al. "A knowledge-rich approach to the rapid enumeration of quasi-magic sudoku search spaces". In: (2009). DOI: 10.5220/0001659502460254.

[15] Christopher D. Rosin. "Multi-armed bandits with episode context". English. In: *Ann. Math. Artif. Intell.* 61.3 (2011), pp. 203–230. ISSN: 1012-2443. DOI: 10.1007/s10472-011-9258-6.

[16] Julian Schrittwieser et al. "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (Dec. 2020), pp. 604–609. ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4. URL: http://dx.doi.org/10.1038/s41586-020-03051-4.

[17] Lesia Semenova, Cynthia Rudin, and Ronald Parr. "On the Existence of Simpler Machine Learning Models". In: *2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '22. ACM, June 2022. DOI: 10.1145/3531146.3533232. URL: http://dx.doi.org/10.1145/3531146.3533232.

[18] D. Silver et al. "Mastering the game of go with deep neural networks and tree search". In: *Nature* 529 (7587 2016), pp. 484–489. DOI: 10.1038/nature16961.

[19] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (Dec. 2018), pp. 1140–1144. ISSN: 1095-9203. DOI: 10.1126/science.aar6404. URL: http://dx.doi.org/10.1126/science.aar6404.

[20] P. Utomo and R. Makarim. "Solving a binary puzzle". In: *Mathematics in Computer Science* 11 (3-4 2017), pp. 515–526. DOI: 10.1007/s11786-017-0322-4.

[21] Constantin Waubert De Puiseau, Hasan Tercan, and Tobias Meisen. "Curriculum Learning in Job Shop Scheduling using Reinforcement Learning". en. In: (2023). DOI: 10.15488/13422. URL: https://www.repo.uni-hannover.de/handle/123456789/13532.

[22] *Wayback Machine — web.archive.org*. https://web.archive.org/web/20061210103525/http://cboyer.club.fr/multimagie/SupplAncetresSudoku.pdf. [Accessed 19-12-2023].

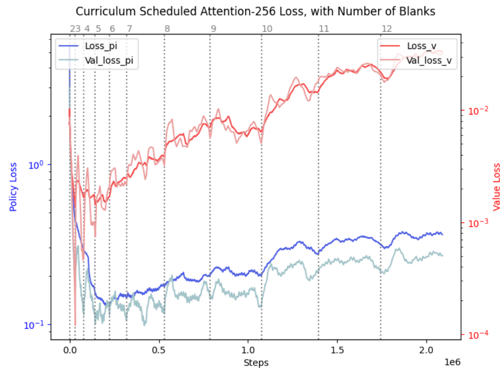
[23] Yibo Yang, Stephan Mandt, and Lucas Theis. *An Introduction to Neural Data Compression*. 2023. arXiv: 2202.06533 [cs.LG].

[24] Takayuki YATO and Takahiro SETA. "Complexity and Completeness of Finding Another Solution and Its Application to Puzzles". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E86-A (May 2003).

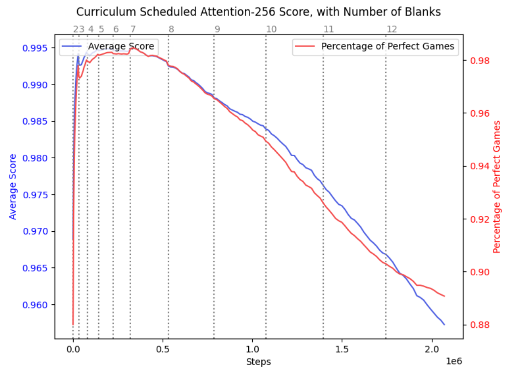
[25] Dinghuai Zhang et al. *Generative Flow Networks for Discrete Probabilistic Modeling*. 2022. arXiv: 2202.01361 [cs.LG].

[26] Dinghuai Zhang et al. *Unifying Generative Models with GFlowNets and Beyond*. 2023. arXiv: 2209.02606 [cs.LG].

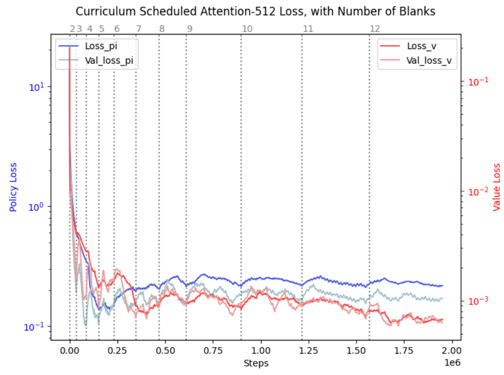
Appendix A. Attention-based Architecture Results



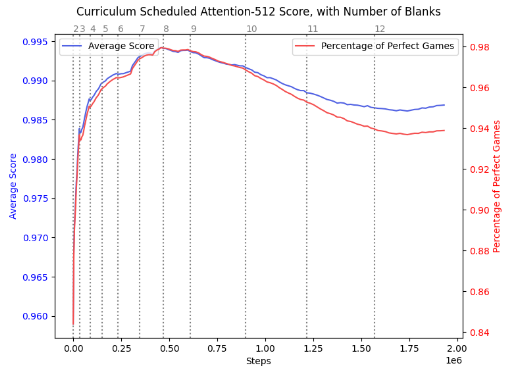
(A) Policy and value training loss curves for attention architecture with hidden layers of size 256 using *cAlphaZero*



(B) Associated score metrics during the cAlphaZero training run with hidden layers of size 256



(C) Policy and value training loss curves for attention architecture with hidden layers of size 512 using cAlphaZero



(D) Associated score metrics during the cAlphaZero training run with hidden layers of size 512

Figure 4. Comparative analysis of attention-based architecture in cAlphaZero training with hidden layers of size 256 and 512.

APPENDIX B. ALGORITHMIC APPROACH TO TRAINING

Algorithm 1: *cAlphaZero* Training Algorithm with Curriculum Scheduling

```
Inputs :
    Neural Network $\theta$
    $\mathcal{X} \leftarrow Set\ of\ puzzles$
    $u \leftarrow Upper\ bound\ of\ number\ of\ squares\ to\ solve\ for\ curriculum\ schedule$
    f(n, $\theta$) $\leftarrow Function\ to\ generate\ data/trajectories\ given\ model$
    $\gamma \leftarrow Quality\ metric\ to\ increase\ number\ of\ squares\ to\ solve$

    while $number\ of\ blanks\ n \leq u$:

        for x$_i \in \mathcal{X}$
            d$_i = f(n \mid \theta, x_i)$          ▷ Generated trajctories of board state, empirical policy and reward
        end for

        $\alpha =$ TrainNet$(\theta, d)$          ▷ Accuracy of training policy network on empirical value of moves

        if $\alpha > \gamma$ :          ▷ Curriculum scheduling
            n' $\leftarrow n + 1$
```

FIGURE 5. The *cAlphaZero* training algorithm with curriculum scheduling and the assumed notation.