Accelerating Neural MCTS Algorithms using Neural Sub-Net Structures

Extended Abstract

Prashank Kadam
Vesta Corporation
Portland, Oregon, United States
prashank.kadam@vesta.io

Ruiyang Xu
Northeastern University
Boston, United States
xu.r@northeastern.edu

Karl Lieberherr
Northeastern University
Boston, United States
k.lieberherr@northeastern.edu

ABSTRACT

Neural MCTS algorithms are a combination of Deep Neural Networks and Monte Carlo Tree Search (MCTS) and have successfully trained Reinforcement Learning agents in a tabula-rasa way. These algorithms have been able to find near-optimal strategies through self-play for different problems. However, these algorithms have significant drawbacks; they take a long time to converge, which requires high computational power and electrical energy. It also becomes difficult for researchers without cutting-edge hardware to pursue Neural MCTS research. We propose Step-MCTS, a novel algorithm that uses subnet structures, each of which simulates a tree that provides a lookahead for exploration. A Step function is used to switch between the subnet structures. We show how state-of-the-art Neural MCTS algorithms can be extended to Step-MCTS and evaluate their performances. Algorithms extended to Step-MCTS show up to 2.1x decrease in the training times and achieve a faster convergence rate compared to the other widely used algorithms in the Neural MCTS domain.

KEYWORDS

Competitive MARL; Adaptive Learning; Growing Neural Networks

ACM Reference Format:
Prashank Kadam, Ruiyang Xu, and Karl Lieberherr. 2023. Accelerating Neural MCTS Algorithms using Neural Sub-Net Structures: Extended Abstract. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION

In recent years, there has been considerable progress in beating human benchmarks and achieving superhuman capabilities in games like Chess, Go, Shogi, etc. The main reason for this breakthrough is the accessibility of a larger amount of computational resources and the development of machine learning in these domains. The invention of algorithms like AlphaZero [11], and MuZero [10] has led to computers beating humans in games like Go which was a far-fetched idea some years ago. These algorithms are called Neural Monte Carlo Tree Search (MCTS) algorithms, as they combine Deep Neural Networks (DNNs) and MCTS [1], [5] to find near-optimal strategies to play these games.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom. © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Although these algorithms perform exceptionally well on such complex games, they have some significant drawbacks. These algorithms take incredibly long periods to converge. They need state-of-the-art graphics processors to run in parallel for training. There have been attempts made to accelerate these algorithms in the past [13], [4], [7], [9], [12], [6], [14], [3], but none of them have shown a substantial improvement in training times.

In general, it has been observed that larger networks perform better at state estimation, but training these networks is computationally very expensive. In contrast, smaller networks can be trained much faster, but the evaluations are not very accurate.

As a part of this paper, we have developed Step MCTS. This algorithm can be extended to other Neural MCTS algorithms to converge much faster on the same hardware configuration in considerably less training time. We achieve this by incrementally increasing the size of the network (subnet) starting from the lowest possible configuration (ex. one hidden layer). During an iteration of training, we train the network based on the self-play trajectories generated during the previous iteration. Then we roll out a tree using the current network configuration. We keep track of all the states visited during each of the self-plays and prioritize them. A 'switch' action (adding an additional layer) is made when performance dips, thus we transition to a new subnet. The new subnet will now also consider the previously prioritized states while simulating a tree.

We extend our Step-MCTS to one of the most widely used Neural MCTS algorithms, AlphaZero, and evaluate our improvements over the vanilla and advanced versions of this algorithm for games like Connect-4, Othello, and Chess. Our experiments show that Step-MCTS shows up to 2.1x improvements in the training times while having a better rate of improvement during training compared to vanilla versions of these algorithms and some of their special configurations.

2 STEP MCTS

2.1 Subnet

In this section, we discuss the Step MCTS algorithm in detail. Let us call our complete network f, which has N hidden layers. Note that a complete network, in this case, means the upper bound to the number of computational resources available. Thus, N is the highest number of hidden layers which we could add to the network given computational hardware. In theory, N could be as large as possible, but we fix it to some finite value in practice. We define a subnet as a configuration of f that has an input layer, output layer, and n hidden layers such that $n \in [1, N]$. Thus, a subnet with one hidden layer would be the most basic configuration available to the algorithm. Let's call it f_1. We start our training with f_1 and keep on increasing the number of hidden layers as required by the network up to f_{N-1} at which point we would have used up all our resources.

Each of these subnet configurations simulates its tree. Let us call the tree simulated by f_1 as T_1 and so on until $T_{(N-1)}$. Starting from f_1, we train in mini-batches, and after each iteration, the step function tells us whether we move to f_2. If the switch is made, f_2 will help roll out T_2, which will now use all the important states/observations explored by T_1, thus saving a considerable time on exploration.

When training the subnet, at the end of each training iteration, the subnet is provided training examples of the form (s_t, π_t, z_t). π_t is an estimate of the policy from state s_t, $z_t \in [-1, 1]$ is the final outcome of the game from the perspective of the player at s_t. All the subnets are then trained to minimize the following loss function (excluding regularisation terms):

$$l = \sum_t (v_\theta(s_t) - z_t)^2 - \pi_t . \log(p_\theta(s_t))$$

here p_θ and v_θ are the policy and values functions respectively.

2.2 Step Function (Threshold based)

The step function is applied after each training iteration to decide whether to switch to the following subnet configuration. Depending on the requirement, the step function could have any underlying configuration (Threshold based, MLP, RNN, etc.). Our threshold-based step function outperformed the RNN-based one. In a threshold-based step function, we set up a threshold value $\lambda_{threshold}$ based on the ratio of the number of games won and lost during a given iteration of training. Since the agent is pitted against itself, if this ratio is lower, the policy is not showing notable improvements.

2.3 Lookahead

Lookahead is a buffer that stores the count of the number of times a state has been visited, the higher the count, the higher the priority. States are assigned priors in the value function based on the priorities calculated in the lookahead buffer. Algorithm 1 gives a detailed description of Step-MCTS.

3 EVALUATIONS

We evaluate our Step-MCTS algorithm for different games. To evaluate our new algorithm, we will be using the 6x6 Connect-4, 8x8 Othello, and Chess. We choose these games because they are sufficiently complex to concretely show that our algorithm performs much better than other Neural-MCTS configurations (AlphaZero [11], and MPV-MCTS [6]). Our evaluations are based on the Elo score [2], Alpharank [8], and the training times. Table 1 shows the details of the experiments and improvements over the AlphaZero algorithm.

Algorithm 1 Step MCTS

```
$s \leftarrow$ Initial State/Observation
$f_i \leftarrow f_1, T_i \leftarrow T_1$
$m \leftarrow$ Self-plays per iteration
for Each Iteration do
    if $S(n) \leftarrow$ Threshold based then
        if $\lambda_{threshold} < n(z_{won})$ then
            $s_{leaf} \leftarrow$ Select leaf from $T_i$ using lookahead
            if No lookahead is available then
                $s_{leaf} \leftarrow$ Select unevaluated leaf from $T_i$
            end if
            $(p, v) \leftarrow f_i(s_{leaf})$
            Update $(T_i, s_{leaf}, (p, v), p_k)$
        else if $\lambda_{threshold} > n(z_{won})$ then
            $f_i \leftarrow f_{i+1}, T_i \leftarrow T_{i+1}$
            $s_{leaf} \leftarrow$ Select leaf from $T_i$ using lookahead
            if No lookahead is available then
                $s_{leaf} \leftarrow$ Select unevaluated leaf from $T_i$
            end if
            $(p, v) \leftarrow f_i(s_{leaf})$
            Update $(T_i, s_{leaf}, (p, v), p_k)$
        end if
    end if
end for
```

4 CONCLUSION

In this paper, we presented Step-MCTS, a novel algorithm that uses subnet structures within the complete network to generate lookahead that helps in faster training of Neural MCTS algorithms on modest hardware. We extended our algorithm to one of the most commonly used Neural MCTS algorithms, AlphaZero, and showed that using Step-MCTS trains ~2x faster on average over all the games that we have evaluated. Note that along with the improvements in training time, we also saw a faster rate of improvement in the Elo and the AlphaRank scores compared to the Vanilla forms of those algorithms. As a future scope of this research, we would like the step function also to predict what kind of layer should be added to the network after the switch event (convolutional, residual, dropout, fully connected, etc.). For this paper, we only considered convolutional layers.

Step MCTS (Threshold)								
Game	Iterations	Self-plays/Iter	Simulations	Switching Iterations	Total Time	Improvement over AZ	Elo Score	AR Score
Connect-4	60	100	25	[4,7,26]	38 hrs	1.89x	1462	0.987
Othello	60	100	50	[3,12,24]	77 hrs	1.92x	1848	0.994
Chess	300	100	50	[7,74,148]	236 hrs	2.06x	2482	0.991

Table 1: Step MCTS training metrics

REFERENCES

[1] Rémi Coulom. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, Vol. 4630. https://doi.org/10.1007/978-3-540-75538-8_7

[2] Arpad E Elo. 1978. The rating of chessplayers, past and present. (1978).

[3] Prashank Kadam, Ruiyang Xu, and Karl J. Lieberherr. 2021. Dual Monte Carlo Tree Search. *CoRR* abs/2103.11517 (2021). arXiv:2103.11517 https://arxiv.org/abs/2103.11517

[4] Julien Kloetzer. 2010. Monte-Carlo techniques : applications to the game of the Amazons.

[5] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning* (Berlin, Germany) *(ECML'06)*. Springer-Verlag, Berlin, Heidelberg, 282–293. https://doi.org/10.1007/11871842_29

[6] Li-Cheng Lan, Wei Li, Ting-Han Wei, and I-Chen Wu. 2019. Multiple Policy Value Monte Carlo Tree Search. In *IJCAI*.

[7] Richard J. Lorentz. 2008. Amazons Discover Monte-Carlo. In *Proceedings of the 6th International Conference on Computers and Games* (Beijing, China) *(CG '08)*. Springer-Verlag, Berlin, Heidelberg, 13–24. https://doi.org/10.1007/978-3-540-87608-3_2

[8] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. 2019. α-Rank: Multi-Agent Evaluation by Evolution. *Nature* 1038 (Jul 2019).

[9] Raghuram Ramanujan and Bart Selman. 2011. Trade-Offs in Sampling-Based Adversarial Planning.

[10] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. 2019. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. http://arxiv.org/abs/1911.08265 cite arxiv:1911.08265.

[11] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144. https://doi.org/10.1126/science.aar6404 arXiv:https://www.science.org/doi/pdf/10.1126/science.aar6404

[12] Mark Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. 2008. Monte-Carlo Tree Search Solver. 25–36. https://doi.org/10.1007/978-3-540-87608-3_3

[13] Mark Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. 2010. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games* 2 (12 2010), 239 – 250. https://doi.org/10.1109/TCIAIG.2010.2061050

[14] Ruiyang Xu, Prashank Kadam, and Karl Lieberherr. 2021. First-Order Problem Solving through Neural MCTS based Reinforcement Learning. https://doi.org/10.48550/ARXIV.2101.04167