Reproducing and Enhancing DeepMind's MuZero

Madhumitha Sivaraj and David Tian
Rutgers University, Department of Computer Science
110 Frelinghuysen Rd, Piscataway, NJ 08854
{madhu.sivaraj,david.tian}@rutgers.edu

Abstract

Games have become one of the most efficient vehicles for evaluating artificial intelligence (AI) algorithms. Agents with tree-based path planning approaches have achieved remarkable successes in artificial intelligence. However, these planning algorithms all rely on knowledge of the environment's dynamics, such as the rules of the game or an accurate simulator. A team at DeepMind released its MuZero agent which predicts the quantities most relevant to game planning to achieve high performance in complex domains. Intuitively, MuZero internally invents game rules or dynamics that lead to accurate planning. In this paper, we reproduce and enhance the MuZero algorithm and attempt to improve the current training rates and final results of the MuZero algorithm. A common problem in reinforcement learning is the balance between exploration and exploitation. Our goal was to build on the current algorithm in an intuitive manner that would allow it to make more informed choices when choosing which actions to explore.

1 Introduction

Constructing agents with planning capabilities has long been one of the main challenges in the pursuit of artificial intelligence. Tree-based planning methods have enjoyed huge success in challenging domains, such as chess and Go, where a perfect simulator is available. In recent years, there has been a great deal of research in the field of reinforcement learning regarding agents with planning capabilities. Many such planning algorithms have already achieved remarkable feats such as defeating the world's best human players in games like Chess, Go, Shogi, and more. They are even already being put into use in various industrial applications. However, the limiting factor behind most planning agents is their ability to precisely understand the underlying dynamics of a scenario, for example, the rules and valid moves of a game. Therefore, they are still not widespread in many industries.

The development of improved AI agents with planning capabilities is important in expanding its use in various real-world applications and industries. Model-based approaches to planning algorithms attempt to learn the rules of the environment in addition to the optimal action policy. These algorithms deduce a model to replicate the environment's state through their interactions with the environment. Current agents have shown that it is possible to achieve superhuman performance in strategic tasks without having prior knowledge of the environment, and future models can be applied beyond games to various industries.

However, this task is quite difficult. While being efficient for some applications, these models struggle to perform well when the environment is visually complex and features many inputs and stimuli, such as with Atari 2600 games like Breakout, Space Invaders, and Tetris. They require immense amounts of training data to perform well, and even then it may not be possible to provide an entirely accurate model for complex real-world scenarios. On the other hand, model-free approaches leave out the environment model and determine the optimal policy directly from the environment. While better than model-based algorithms for problems that are difficult to model, their performance was still not

satisfactory. These algorithms have lackluster performance in games that require significant future planning, such as chess and Go.

In 2019, the team at Google's DeepMind unveiled MuZero [1], a state of the art model-based reinforcement learning algorithm which is notable for being capable of achieving high performance in both visually complex domains, namely Atari 2600 video games, and games that require a high degree of planning, such as chess and Go. MuZero utilizes a set of three neural networks to implement its planning capabilities. The input is a history of observations of the game, such as the last couple states of a chessboard or the last couple actions and RGB frames on the screen for an Atari 2600 game. This is transformed into a hidden state which is treated as the root of a Markov Decision Process. The hidden state is expanded by predicting the policy and the expected reward for each action, and the algorithm recursively explores the best action. There is no restriction on the algorithm for the hidden state to be able to reconstruct the observed game state. In this sense, MuZero comes up with its understanding of the game's dynamics. The authors hope that it can serve as the groundwork for applying reinforcement learning tools to extend to solving real-world problems that are not easily modeled.

The researchers who designed the MuZero algorithm indicated that there is room for improvements to be made to it that could make it more efficient or allow it to be generalized to other problems more easily. Our team decided to attempt to create our basic implementation of MuZero using Python. We brainstormed some other potential adjustments that we could make to the algorithm to improve its performance. We added some of these adjustments to an open-source implementation to test their effects.

2 Related Works

In 2019, DeepMind published their algorithm MuZero [1], highlighted by superhuman ability in 57 different Atari games. MuZero is the fourth in a line of reinforcement learning algorithms developed by engineers at DeepMind, which paves the way for learning methods in a host of real-world domains.

The first iteration was AlphaGo [2], which was provided with a model of the game Go before training. It used two neural networks to evaluate the best next action to take and to predict the expected value of that move. It was trained on data from thousands of matches between professional players. In 2016, AlphaGo became the first computer agent to beat a human professional player and was able to do so reliably. Just a year later, DeepMind released a second algorithm called AlphaGo Zero [3]. Unlike its predecessor, AlphaGo Zero was trained from scratch without any need for human strategy inputs. It started from completely random policies and played games against itself, eventually surpassing AlphaGo by far. This algorithm learns solely through self-play, starting *tabula rasa*, and gradually finding strategies that would beat previous incarnations of itself. The creation of AlphaGo Zero rendered the need for a database of human expert games to build a superhuman AI useless. The third iteration of the series, AlphaZero [4], combined the insights from the two preceding models to create a single generalized algorithm that is capable of learning superhuman strategies in many other complex games, including Chess and Shogi. It was trained against itself in the same vein as AlphaGo Zero, but instead of using a handcrafted evaluation function and move-ordering heuristics, it used a neural network to learn reward estimates directly.

All three previous algorithms are model-based and require knowledge of the game's rules before being trained. Although this is not a problem for well-defined games like Go, Chess, and Shogi, in which AlphaZero excels, it is a major obstacle to planning algorithms being able to perform on par with or better than humans in more complex domains. The closest precursor to MuZero is the value prediction network. Like most other model-based approaches, it represents the game state with a Markov Decision Process (MDP), which contains a model for the transitioning of states, such as the actions and the value associated with each action. However, value prediction focuses on achieving value equivalence, or in other words, the reward of moving through a chain of actions in the MDP is equal to the reward of taking the same actions in the environment. It is not concerned with policy prediction.

Unlike AlphaZero which had the advantage of knowing the rules of games it was tasked with playing, MuZero intuitively invents game rules or dynamics that lead to accurate planning, teaching itself the rules by combining a tree-based search with a learned model. It predicts the quantities most relevant to game planning to achieve high performance in complex domains.

3 Main Contributions

Our goal was to reproduce DeepMind's MuZero Algorithm, which predicts the quantities most relevant to game planning to achieve high performance in complex domains. We also intended to improve the current training rates or final results of the MuZero algorithm. A common problem in reinforcement learning is the balance between exploration and exploitation. The challenge here is to build upon the current algorithm in an intuitive way that allows it to make more informed decisions when choosing which actions to explore.

3.1 MuZero Architecture

As mentioned before, MuZero is the fourth and most advanced planning algorithm developed by DeepMind. Unlike previous iterations, MuZero has no understanding of the dynamics of the environment; instead, it trains three neural networks to gather only the information necessary to the operation of the algorithm. The first, representation, takes the observed state of the game, such as a chessboard or Atari screenshot, as input and outputs a hidden state for the overall model. This hidden state does not correlate with the actual environment state, thus making the model far more flexible. The second, dynamics, takes as input a hidden state and a possible action from that state and computes the resulting new hidden state and the immediate reward for doing so. The third, prediction, generates the policy, or the set of possible actions that can be taken from a hidden state, as well as the value for each action.

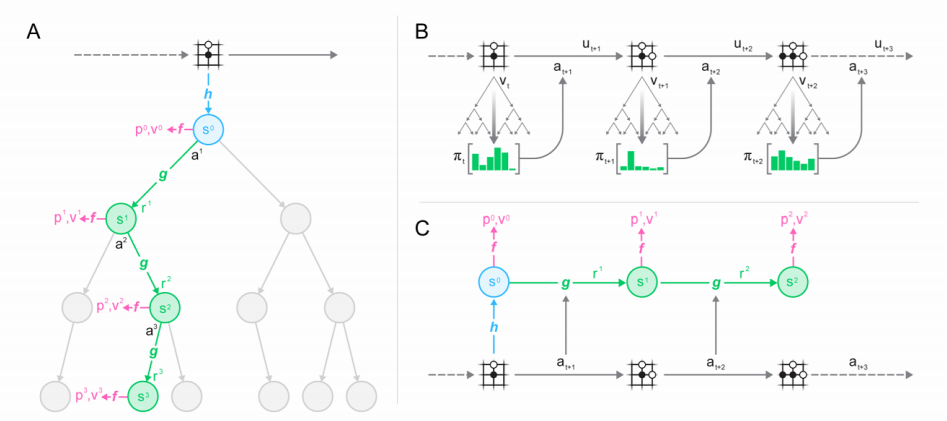

Figure 1: MuZero (A) uses its model to plan, (B) acts in the environment, and (C) trains its model.

At its core, MuZero uses a Monte Carlo Tree Search (MCTS) algorithm similar to that of AlphaZero in tandem with its three neural networks. At every turn of the game, the observation of the game state is processed into a hidden state by the representation network, denoted h in Figure 1. This root node is expanded into a number of actions and their values by the prediction network, denoted *f*. The MCTS then traverses down the tree, selecting the node with the highest upper confidence bound (UCB), which increases with the mean value of the action and the number of times it has been visited. Once it reaches a leaf, it is expanded with the dynamics function, denoted as *g*. The value is finally backpropagated up the tree, updating visited counts as it goes. This process is repeated a number of times, then the action is chosen based on the number of times each child of the root has been visited. In this way, the model inherently selects the next action that it believes is the best since it would have elected to visit nodes with better value estimates more frequently.

3.2 Reproduction

One of our main contributions to this project was creating our implementation of the MuZero algorithm. Our implementation is based on the pseudocode [5] made available by DeepMind. Our model involves four components or classes that run simultaneously in a dedicated thread. The shared storage holds the latest neural network weights. The self-play uses those weights to generate self-play

games and store them in the replay buffer. Finally, the played games are used to train a network and store the weights in the shared storage. This structure is illustrated in Figure 2.

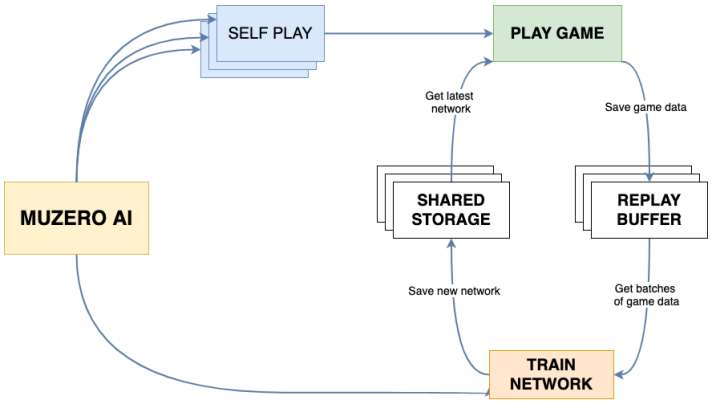

Figure 2: Training schematic.

3.3 Modification

In addition to creating our implementation, we also made modifications to the MuZero algorithm to see if they could improve its performance. To do so we began with the MuZero General implementation [6], available on GitHub. We tried applying a technique known as simulated annealing, in which the algorithm is allowed to make "mistakes" to discover potentially better strategies. Instead of always traversing the action tree by selecting the node with the highest UCB, we modified the algorithm to select a node based on a probability distribution computed from a softmax of all the UBS scores. We reasoned that this would give the algorithm more opportunity to explore moves that it would not have if it was following the original greedy approach. This would allow it to discover potentially better moves that might not have immediately better rewards. Finally, we also increased the maximum number of moves that the algorithm would simulate to counteract the effect of exploring a wider area of the tree.

4 Results

The MuZero general implementation that we used as a baseline included utilities to plot training statistics, including loss and total reward, in real-time. We ran the training in Google Colab and collected information in TensorBoard. Since the implementation did not provide an environment for Chess, Shogi, or Go, we ran our training on Connect 4.

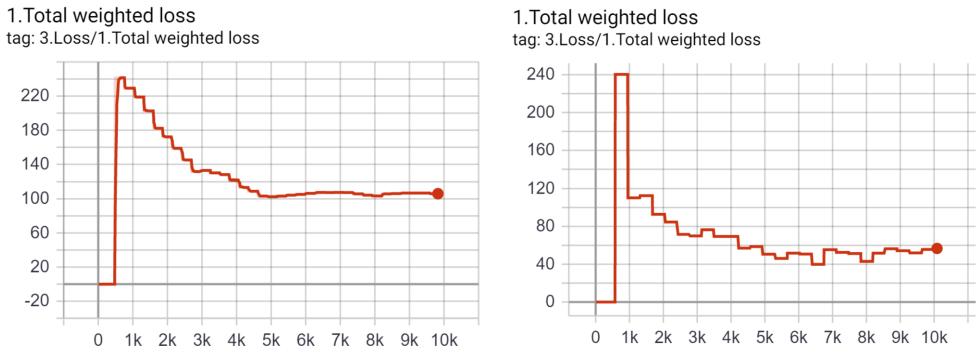

Figure 3: Total loss of the original algorithm (left) compared to that of our modified algorithm (right).

Figure 3 shows the total weighted loss of the original MuZero General on the left and that of the algorithm with our modification on the right. There is a sharp drop early on in the modified algorithm's loss, we presume because it was able to explore and gather far more information than the original. However, the modified loss also does not converge as well because of its tendency to open up many branches of the action tree that may not be helpful. Overall, the modified algorithm still achieves a lower loss in the same amount of training time. We also observed a more stable and higher total reward for the modified algorithm.

5 Conclusion

We reproduced and enhanced DeepMind's MuZero agent which intuitively invents game rules or dynamics that lead to accurate planning and predicts the quantities most relevant to game planning to achieve high performance in complex domains. We recreated a performant machine learning model capable of teaching itself the rules by combining a tree-based search with a learned model. We experimented with strategies including hyperparameter tuning and continuous action space to help improve our agent's performance without much success. Our modified Monte Carlo Tree Search shows somewhat promising results, but as always there is still room for additional improvement on our model.

6 Future Work

While DeepMind's MuZero agent is a step ahead of its predecessors, it is still not quite suitable for real-life scenarios. However, we believe further improvements can be made to our model to enhance its performance within the scope of games (ie. Chess, Go Shogi, and other Atari games). We can scale the hidden state from -1 and 1 using the tanh function rather than between 0 and 1 using min-max normalization. We could also scale the loss of each head by $\frac{1}{X}$ with X being the number of unrolled steps. We could also explore using a simple invertible transform for the value prediction by removing the linear term. Lastly, during training, we can investigate utilizing samples that are drawn from a uniform distribution instead of using prioritized replay.

Contribution

Madhumitha Sivaraj and David Tian contributed equally to all aspects of the project including reproducing the MuZero agent, designing and developing modifications to existing implementations, running performance experiments of various games, and putting together the final presentation and report.

Acknowledgments

We thank Dr. Sungjin Ahn of the Department of Computer Science, Rutgers University for his support and instruction.

References

[1] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., & Lillicrap, T. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. https://arxiv.org/pdf/1911.08265.pdf

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489. https://doi.org/10.1038/nature16961

[3] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354–359. https://doi.org/10.1038/nature24270

[4] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140–1144. https://doi.org/10.1126/science.aar6404

[5] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., & Lillicrap, T. (2020). MuZero Pseudocode (Version 2). https://arxiv.org/src/1911.08265/anc/pseudocode.py

[6] Duvaud, W., Hainaut, A., & Lenoir, P. (2020). MuZero General [Source code]. https://github.com/werner-duvaud/muzero-general