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

CSM6420 Report by Morgan Jones (mwj7@aber.ac.uk)

# Overview of MuZero

The algorithm presented in this paper is MuZero [1]. It is an extension and generalisation of AlphaZero [2] a past state-of-the-art work by the same authors. MuZero is a model-based reinforcement learning (RL) [3] algorithm that uses deep neural networks [4] [5] to estimate transition reward, action-selection and state value quantities. The learning of these quantities enables the planning capabilities of the algorithm to achieve superhuman performance in Go<REF>, Chess<REF>, Shogi<REF> and 57 different Atari games implemented in the Arcade Learning Environment (ALE) [6].

MuZero is a model-based RL algorithm because it plans with respect to a learned model of the environment’s dynamics; that is actions, rewards, state transitions and (if stochastic) transition probabilities. Previous model-based algorithms have struggled in visually rich domains, such as Atari 2600, with the most successful methods based on model-free RL [3]. This is a significance of MuZero it is a model-based algorithm that achieves state-of-the-art performance in visually complex domains (Atari 2600) as well as maintaining superhuman performance in two player zero sum games.

The domain model has to be learned by MuZero from scratch. This is a generalisation of AlphaZero because AlphaZero was given knowledge of the game rules in the form of implementation in a simulator. AlphaZero made use of the provided model when performing the search required for its search-based policy iteration. The given model provided all legal actions, deterministic state transitions and terminal states to be used by the Monte-Carlo Tree Search (MCTS) [7] to traverse game simulations in the search tree. How MuZero does search without model?

MuZero also extends AlphaZero to work in more environments including single agent domains and environments where actions yield non-zero immediate rewards. The AlphaZero algorithm was built for Chess, Go and Shogi where all state transitions were set to zero immediate reward except for terminal states where a Win, Loss or Draw was 1, -1 and 0 respectively.

# Algorithm Components

## Model

MuZero’s RL model is made up of three components: a representation function , dynamics function and prediction function . All three functions are neural networks with denoting the network parameters (weights).

The **representation function**  is responsible for transforming observations of the environment into an initial hidden state used to initialise the root of the search tree for planning. This action is one-way because “There is no direct constraint or requirement for the hidden state to capture all information necessary to reconstruct the original observation”.

The **dynamics function** is responsible for thinking ahead at each hypothetical step k by transitioning between hidden states. The dynamics function takes an action from a hidden state and computes the next hidden state and the immediate reward for the transition.

The **prediction function** evaluates a hidden state by predicting the policy and value from it. This is the same as the joint policy & value predicting network of AlphaZero.

A **policy** is a mapping of states to actions. Here the policy is specifically a mapping of a sequence of observations (initial hidden state ) and a sequence of actions to the next action. Effectively saying what action to take given the currently considered hidden state .

The **value function** in reinforcement learning estimates the expected value of being in a state, that is the expected sum of discounted future rewards given we are in a state and continue to follow the current policy . This is similar in MuZero except the discounted future rewards are those generated by the environment not the immediate rewards used for internal planning and the total reward is conditioned is on the hidden state that is past observations and future actions. The discount factor is denoted by

The **immediate reward** is approximating the true observed reward of the environment.

The model predicts at each time step the three quantities described above: policy, value function and immediate reward. This mirrors the structure of the typical Markov-Decision Process (MDP) [8] model of the RL problem. A key difference being a hidden state has no semantics of the environment associated with it because hidden states exist solely to predict the quantities mentioned above.

## Search

Given the model above MuZero can use a planning algorithm to search over hypothetical future trajectories. Monte Carlo Tree Search (MCTS) is used by the authors but they do state that any MDP planning algorithm may be used. The MCTS is used to output a recommended policy and a value for the current environment observation at time step . The agent then acts on the environment according to the policy’s recommended next action . The values of the actions in the search policy are proportional to their visit count during the MCTS. MCTS is briefly described below.

MCTS HERE

## Networks

### Network Input

### Network Architecture

### Learning?

### Training?

# Proposed Alterations

# Results

ASK CHUEN IF THIS SHOULD BE IN HERE

# Limitations & Future Directions

* The dynamics function is deterministic which means all state transistions succedd with a probability of 1. The authors mention that “extension to stochastic transitions is left for future work”. Expand this to show RL equation and how it could be done with their algorithm.
* Future real world domains with unknown environment dynamics.

# References

|  |  |
| --- | --- |
| [1] | J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Gues, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap and D. Silver, “Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model,” *arXiv preprint arXiv: 1911.08265,* 2019. |
| [2] | D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis, “A general reinforcement learning algorithm that masters chess, shogi and go through self-play,” *Science,* pp. 1140-1144, 2018. |
| [3] | R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 2018. |
| [4] | J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” *Neural Networks,* vol. 61, pp. 85-117, 2015. |
| [5] | A. Krizhevskym, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *In Advances in Neural Information Processing Systems,* no. 25, pp. 1106-1114, 2012. |
| [6] | M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research,* vol. 47, pp. 253-279, 2013. |
| [7] | C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in games,* vol. 4, pp. 1-43, 2012. |