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 given 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 runs simulations from the root state to leaf nodes and stores a set of statistics for each edge along the search tree that it visits. In our algorithm the set of statistics stored is

* is the visit count of an edge.
* is the average Q-value (expected total future return) of the edge, averaged over all simulations.
* is the prior probability of taking that actions from that state according to the current policy.
* is the immediate transition reward of that edge.
* is the resulting state after traversing the edge.

There are 3 stages to MCTS:

1. **Selection**
2. **Expansion**
3. **Backup**

MuZero can be used for games or MDPs because MCTS approach to planning converges asymptotically to the optimal policy in single agent domains and to the minimax <REF> value function in zero sum games <REF>. Note that authors used 800 simulations per search for board games and 50 for Atari because of the smaller branching factor in Atari.

## Networks

### Learning

MCTS is a relevant to the machine learning components of the solution because it results in targets that guide the learning of the neural networks. The three objectives of this learning are as follows:

* To minimise the error between predicted policy and search policy .
* To minimise the error between the predicted value and the value target .
* To minimise the error between the predicted reward and the observed reward

These objectives along with an L2 regularisation term lead to the overall loss function for the model.



The function refers to the linear combination of a real number through a combination of its adjacent integers. This is because the authors normalise values and rewards using an invertible transform, thereafter using a transformation to obtain equivalent categorical representations of the target values from a discrete set of 601 integers between 300 and -300. With each real value being a linear combination of its two adjacent integers in the set. The value and reward are predicted as a Softmax distribution over the set with a real value being obtained by computing the expected value from over the distribution and then inverting the transformation.

### Input

#### Encoding game state

For the board games the input to the representation function in MuZero is similar to the representation AlphaZero used for board states (shown in table below). With the exception that for chess the history has been increased to the last 100 states to better predict draws. Meaning total input depth for chess below would now be 8x8x1407 instead of 8x8x119.

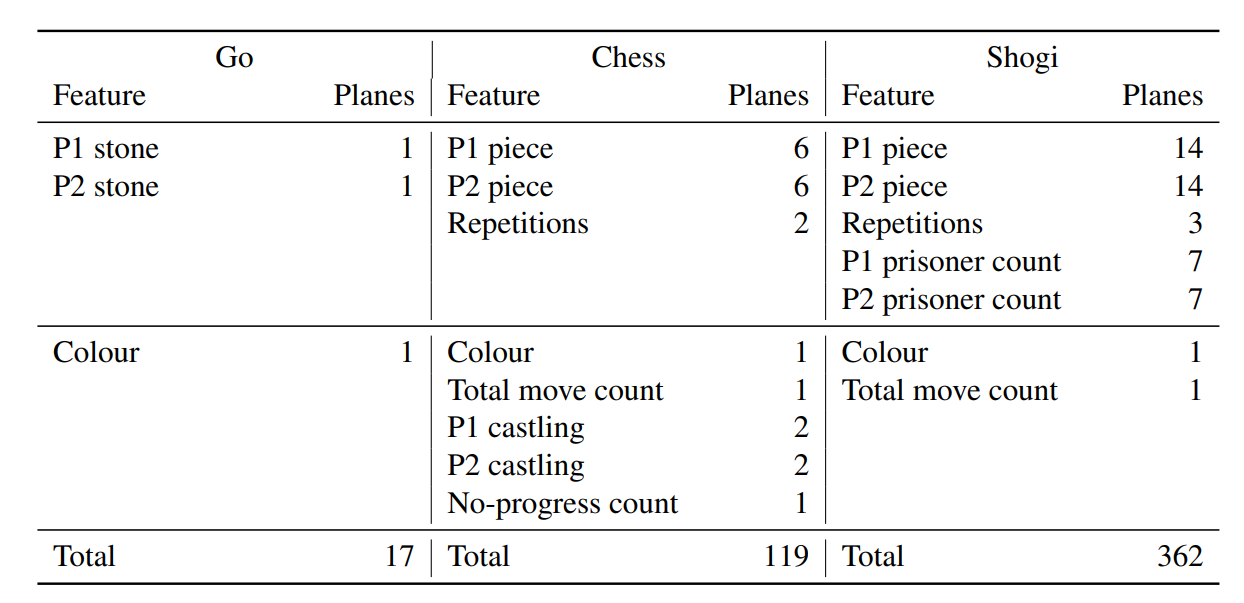


Figure 1 - Input features used by AlphaZero in Go, Chess and Shogi. First row repeated for a T = 8-step history.

The representation input for Atari is the last 32 RGB frames at resolution 96x96 and the last 32 actions taken at each frame. Historical actions are needed because not all actions have immediate visible effect. RGB planes are encoded as 3 binary planes leaving the final input dimensions at 96x96x128.

#### Encoding transition action

The dynamics function takes the output of either the representation function or a previous application of the dynamics function as input along with a representation of an action. Actions are encoded as planes of the same resolution as the hidden state (6x6 for Atari & board size for board games).

A Go action is a single plane representing the position of stone placement or a pass. An action in chess is 8 planes: the start position, end position, a binary plane describing if the move was legal along with 5 binary planes representing the type of promotion {queen, knight, bishop, rook, none}. Shogi actions are encoded in 11 planes: 8 planes for where the piece was moved from, 2 planes for the target and legality of the move and a single binary plane denoting promotion.

Atari actions are one hot vectors tiled appropriately into planes. UNDERSTAND THIS AND EXPAND?

### Architecture

### Training

At the end of an episode of gameplay trajectories are stored in a replay buffer. These sequences are sampled by selecting a state from any game stored in the replay buffer. The initial step receives the past observations up to that state and applies the representation function to attain a hidden state. Thereafter the sequence is unrolled until the end of the game (Chess/Go/Shogi) or 10 steps into the future (Atari) by passing each hidden state and the actual past action taken into the dynamics function.

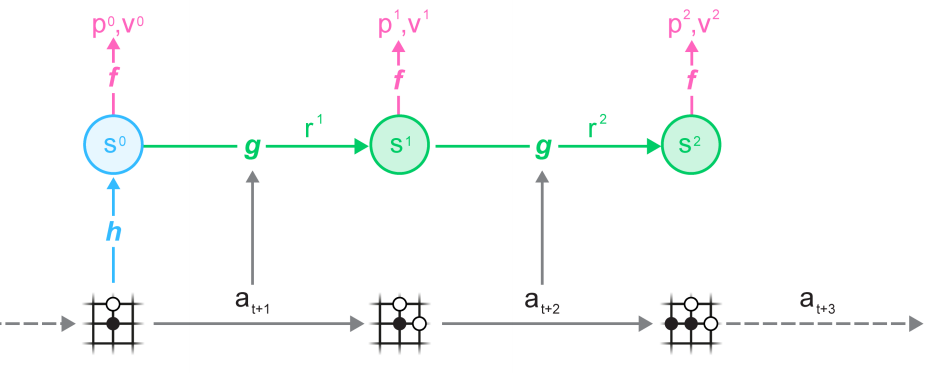


Figure - MuZero training by unrolling from the replay buffer.

Prioritised replay <REF> is used to draw samples from the replay buffer when learning Atari. With states being sampled uniformly for zero sum games. Prioritised replay focuses the algorithm to train on those past observations from which there is the most to learn. This is more useful in Atari than in zero sum games because in zero sum games all that decides a win or loss is the terminal state of the game. PR important because Atari has huge depth can’t just play to the end every time so must sample more appropriately. (ALTERATION perhaps) Also, Atari has intermediate rewards which makes some samples can teach us a lot more than others i.e when something crashes and you lose loads of points from an intermediate reward that would probably

When learning Atari samples are selected with the following priority

# Results

ASK CHUEN IF THIS SHOULD BE IN HERE

# Proposed Alterations

# 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. |