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, Chess, Shogi 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.

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 possible by three components: a representation function , dynamics function and prediction function . All three functions are neural networks with denoting the network parameters (weights).

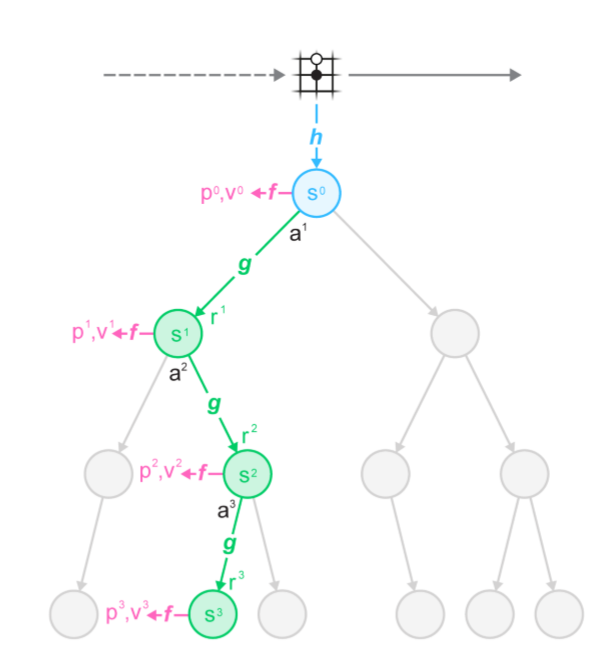


Figure - MuZero using its 3 component networks {h(), g(), f()} to perform internal planning for k hypothetical steps.

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 recurrent 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 . In games this would be the predicted winner, in Atari the predicted final score. The discount factor is denoted by Discounting means rewards further into the future are valued less, the variation of this hyperparameter reflects the trade-off between short and long term planning.

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 iteratively apply a planning algorithm to the state space induced by the dynamics function to search over hypothetical future trajectories when acting and learning. 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 and receives an environment reward . The values of the actions in the search policy are proportional to their visit count during the MCTS. The value, search policy and environment reward as result of the MCTS are later used as improved targets for the training of the neural networks. 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 action 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**During selection nodes are traversed from the root to a leaf node with actions being selected based on maximisation of an upper confidence bound. This is calculated using the stored statistics for each edge.
2. **Expansion**When reaching a leaf node (unless the node is terminal) the search tree is expanded by adding a child of that node to the tree. Edge statistics are initialised for the new connection.
3. **Backup**At the end of a simulation statistics along the trajectory are updated.

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 value function in zero sum games [8]. Note that authors used 800 simulations per search for board games and 50 for Atari because of the smaller branching factor in Atari.

## Networks

### 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 dimensions for chess would now be 8x8x1407 instead of 8x8x119.

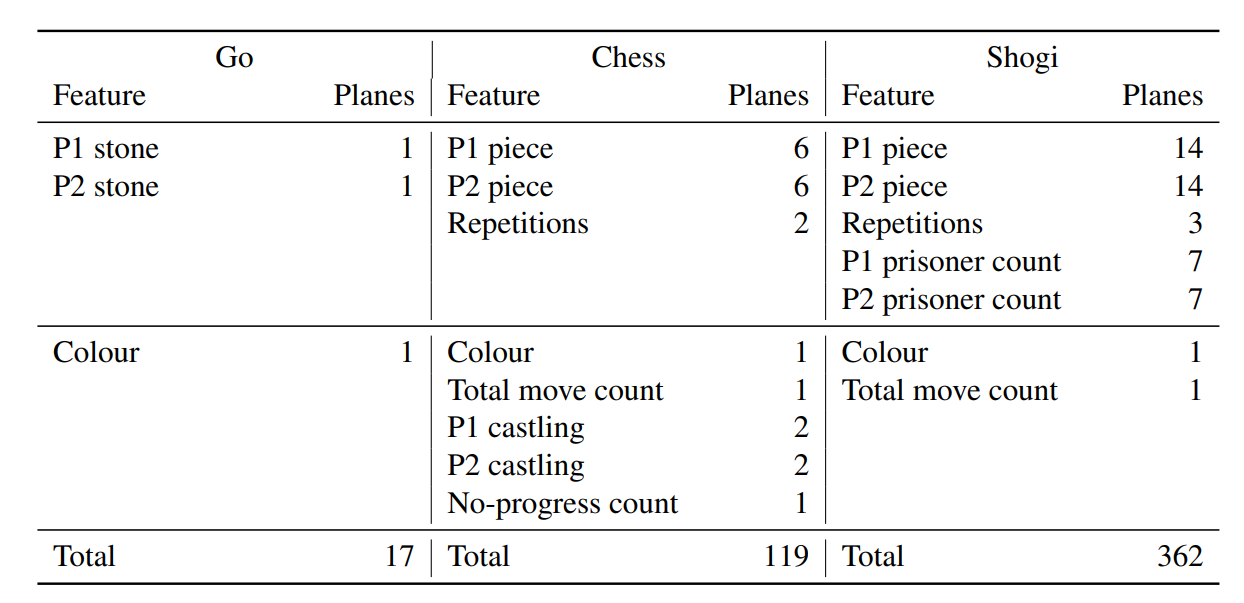


Figure - Input features used by AlphaZero in Go, Chess and Shogi. First row repeated for a T = 8-step history.

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 to match the number of simultaneous button presses.

### Architecture

The representation, prediction and dynamics functions in MuZero are approximated by deep residual convolutional neural networks [4]. The architecture is originally described in detail in the methods section of the AlphaGo Zero paper “Mastering the game of Go without human knowledge” [9]. In this previous paper the authors trial four alternative architectures for their single network:

* **Dual-res**: Single 20-block residual tower branching to a policy head and a value head
* Sep-res: Two 20-block residual towers each with their own output, one tower for policy the other for value
* Dual-conv: Single 12-block convolutional tower branching to a policy head and a value head
* Sep-conv: Two 12-block convolutional towers each with their own output, one tower for policy the other for value

The authors allude to the 20-block dual-res as the reference architecture for AlphaZero. The single network in AlphaGo Zerois the equivalent of the prediction function in MuZero (outputting policy and value). The authors state that the prediction function has the same architecture as AlphaZero, with the dynamics and representation functions being similar although having 16 instead of 20 residual blocks. The representation and dynamics functions are new to MuZerohowever still comprised of the same building blocks; a series of convolutional and/or residual layers.

#### Convolutional Layer

A convolutional layer in the algorithm is comprised of the following:

1. A convolutional of 256 filters using a 3x3 receptive field with a stride of 1
2. Batch Normalisation
3. A rectifier nonlinearity (ReLU)

#### Residual Layer

A residual layer is just two convolutional layers stacked with a skip connection included that allows the original input to the layer to be added to the output. A residual layer in the algorithm is comprised of the following:

1. A convolutional of 256 filters using a 3x3 receptive field with a stride of 1
2. Batch Normalisation
3. A rectifier nonlinearity (ReLU)
4. A convolutional of 256 filters using a 3x3 receptive field with a stride of 1
5. Batch Normalisation
6. A skip connection that adds the input to the block output
7. A rectifier nonlinearity (ReLU)

A screenshot of a cell phone

Description automatically generated

Figure - Representation of residual layer.

The representation function’s architecture is unique for Atari because the input observations need to be significantly down sampled from the original 96x96 input frames. The architecture for the Atari representation function is stated explicitly in Appendix F of the MuZero paper [1].

### Training

MuZero runs in two ways: it performs self-play to generate experiences and it performs training to learn from past experiences. Experience generation follows the same exploration/exploitation scheme as AlphaZero by decreasing a temperature parameter to select actions from the MCTS visit-count distribution more greedily as training progresses.

At the end of an episode of gameplay trajectories are sent to the training job to be stored in a replay buffer. The replay buffer stores the most recent 1 million games and the most recent 125,00 200 length sequences for Atari. Sequences are sampled by selecting a state from any sequence 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 for K=5 steps by passing each hidden state and the actual past action taken into the dynamics function.

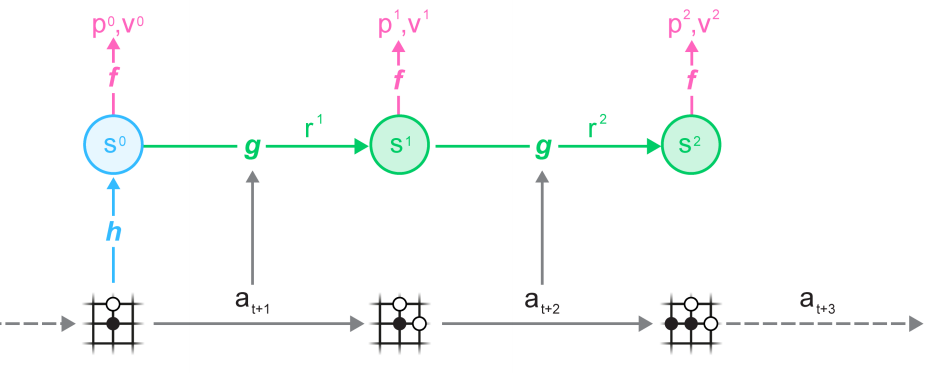


Figure - MuZero training by unrolling from the replay buffer.

Prioritised replay [9] 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.

When learning Atari samples are selected with the following priority , where is the search value (expected return) predicted and is the observed total return. The priority says that those states with a larger relative difference in correct & predicted value will have a higher chance of being sampled from the replay buffer.

### Learning

At each unrolled step the network’s predictions have losses to the target values (policy, value & environment reward) the summation of which produce the total loss for the network . The target value is the actual return of the sample which is bootstrapped until the end of the game for Chess/Go/Shogi or 10 steps into the future for Atari.

The losses of the three target values along with an L2 regularisation term lead to the overall loss function for the model. The objectives of the model loss function are:

* 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

# Results

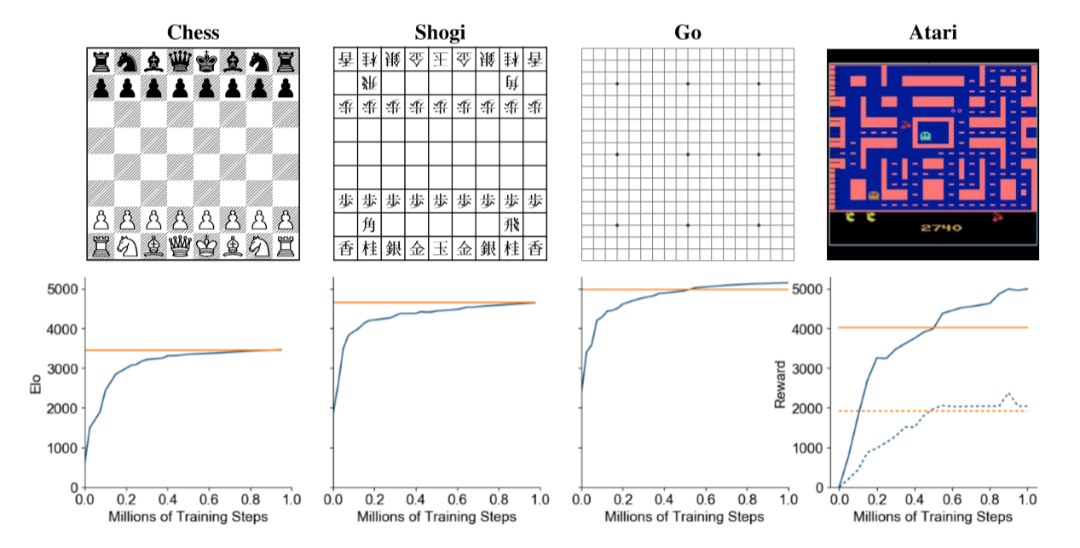


Figure - MuZero evaluation throughout training.

MuZero was trained for 1 million mini-batches with batch size being 2048 for board games and 1024 for Atari. The plots above show the progress of MuZero as it trains. It matches AlphaZero’s Elo rating[[1]](#footnote-2) in Chess and Shogi while exceeding it in Go. MuZero also exceeds R2D2 [11]; the previous (model-free) state-of-the-art for Atari. The authors compare the mean and median normalised score of past Atari agents in the table below.

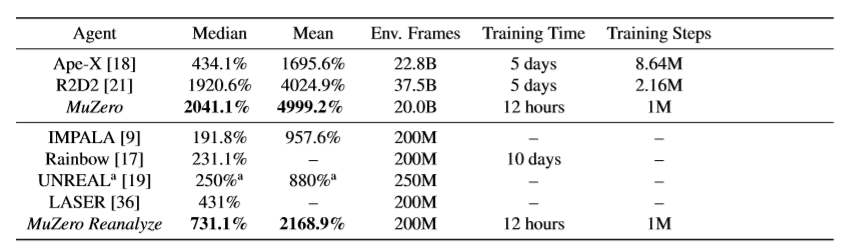


Figure - Comparison of MuZero against previous agents in Atari.

In this table ‘*MuZero’* and ‘*MuZero Reanalyze’* are the best Atari agents. MuZero Reanalyze is a variant of MuZero that re-executes search on past experience to potentially get a better stored search policy for that experience than the one produced by the original search. The fresh policy is used as the policy target over the original policy for 80% of updates during training. MuZero Reanalyze also uses more recent parameters in a target network to provide a fresher target for the value function.

MuZero was compared against a strong model-free algorithm to better understand the benefit of a model-based approach. MuZero was adapted into a Q-Learning algorithm [12] by replacing its training objective, changing its output to a single Q-value and omitting search during training and evaluation.

Original MuZero and the Q-Learning alternative were trained to play Ms. Pacman. The performance of the Q-Learning version was similar to R2D2 but learned slower and performed worse than MuZero. The authors conjecture that this is due to the search-based policy improvement providing better learning targets with less bias and variance than those used by Q-Learning.

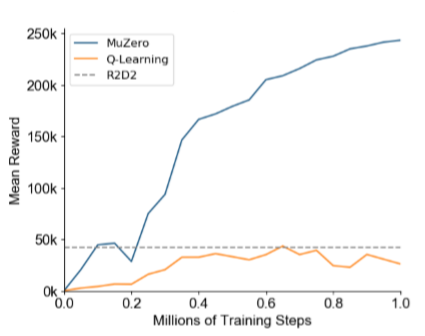


Figure - Comparison of training on Ms. Pacman with MCTS and Q-Learning within the MuZero framework.

# Proposed Alterations

I was thinking about intermediate rewards having some influence over the sampling during prioritised replay because in Atari there are some actions that have large rewards that would be intuitive to want to learn more from (i.e. blowing up the big boss spaceship in Space Invaders). However I understand what the authors were thinking by basing sampling solely on state value because the final output of the game is all that really matters plus that final value is essentially just a discounted summation of immediate rewards anyway. The authors mention unrolling past gameplay for K=5 steps perhaps the number of unroll steps could be increased based on the size of the observed immediate reward because we would want to do more learning from an experience with larger reward. Although I am aware that this is what the value is doing anyway because it is bootstrapped 10 steps into the future therefore taking account of the future 10 summed rewards for Atari. Maybe all that is needed is to increase the number of bootstrapped steps based of the difference in value (probability of being selected by prioritised replay) but then

# Limitations & Future Directions

The dynamics function is deterministic which means all state transitions succeed with certain probability. The authors mention that “extension to stochastic transitions is left for future work” this could easily be incorporated because the original RL problem modelled as an MDP is defined as containing a matrix of transition probabilities. The original equation for the value/utility of a state as defined by the Bellman Equations [8] is shown below. The outcome probabilities for each action are . In the deterministic world of MuZero these probabilities are all 1.

DeepMind’s goal from the start has been to “Solve intelligence” and then “Use it solve everything else”. MuZero doesn’t know the rules of the game but still does brilliantly. The ability to learn a model and still achieve so well could be used in many real-world domains with unknown environment dynamics. Some of the aims briefly mentioned by Demis Hassabis the founder of DeepMind [13] were:

* Classification in Radiography
* Protein Folding
* Drug Discovery
* Simulation of Mars exploration

Since MuZero is an extension and generalisation of their previous AI framework I think it is pushing closer to the goal of general intelligence and therefore closer to solving problems in these domains and many more.

# 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. |
| [8] | S. J. Russel and P. Norvig, Artifical Intelligence: a modern approach, Pearson Education, 2014. |
| [9] | D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Gues, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel and D. Hassabis, “Mastering the game of Go without human knowledge,” *Nature,* vol. 550, pp. 354-359, 2017. |
| [10] | T. Schaul, J. Quan, I. Antonoglou and D. Silver, “Prioritized experience replay,” in *International Conference on Learning Representations*, 2016. |
| [11] | S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan and R. Munos, “Recurrent experience replay in distributed reinforcement learning.,” in *International conferenc on learning representations*, 2019. |
| [12] | C. J. Watkins and P. Dayan, “Q-Learning,” *Machine Learning,* no. 8, pp. 279-292, 1992. |
| [13] | A. A. o. Achievement, “Demis Hassabis interview on YT,” May 2018. [Online]. Available: https://www.youtube.com/watch?v=1X7Koxx4qJE. [Accessed 07 04 2020]. |

1. Elo rating is the relative skill level of players in zero-sum games. [↑](#footnote-ref-2)