"BaghChal: Reinforcement Learning Based Self-Play for Move the Tigers Game"

Rojina Panta, Hem Regmi
rpanta@email.sc.edu, hregmi@email.sc.edu
December 06, 2021
Department of Computer Science and Engineering
University of South Carolina

1. Introduction

Two player games in grid world have been in frontier of Reinforcement Learning task since the use of Reinforcement Learning algorithm to master the game of Go using AlphaGoZero. In this project *BaghChal*, we are using the concept from algorithms like AlphaGoZero and AlphaZero to implement the reinforcement learning state-of-art method to master the *BaghChal* game for both agents (tigers and goats) through self-play [1]. In past, self-play techniques have been utilized to master complex games such as Chess, Shogi, and Go. And AlphaZero shows it can be generalized across different board games. So, we try to borrow this idea of self-play and self-master in the *BaghChal*.

The game of *BaghChal* is popular only in specific region of South East Asia which makes implementation of this game both interesting and difficult. We have represented this board game with multiple layers of binary representation which is then passed to Deep Neural Network to predict policy and value. Then we are using Monte Carlo Tree Search (MCTS) implementation of AlphaZero paper to generate the training samples with self-play. The data thus generated from MCTS is again used to train deep neural network to improve the policy and value. We have implemented a working model of board game *BaghChal* using concept from algorithms like AlphaGoZero and AlphaZero and will discuss about it further in the paper.

2. Related Work

Project *BaghChal*'s primary goal is to train and build an RL agent which can optimally play the game. MuZero [2] implements the general outline for the AlphaZero that includes the games such as *atari, breakout, cartpole, connect4, gomoku, gridworld, lunarlander, simple grid, spiel, tictactoe,* and *twentyone*. However, all these games have the action space where an agent moves into the adjacent edges, but *BaghChal* has the two stages of the game, where it first have goat placement phase and goat movement phase, as well as Tiger can stride into open position to kill a goat if the target position is empty. The way an RL agent can move in the board is similar to the game implementation of *Chess*. We follow [4] to find out the number of possible state spaces. The article suggests to dividing the *BaghChal* into total 6 sub states; S_0, S_1, S_2, S_3, S_4, and S_5. S_0: represent all the tiger and goats position that can happen during placement phase. S_1-S_5: represent all the position that can happen during the sliding phase where numbers 1 to 5 represents the number of goats killed[1]. The analysis shows that S_0 has nearly 33 billion states and the game tree complexity of *BaghChal* during placement phase is 10^{41}.

3. Approach

There is a limited amount of research works on this particular game as it is only played on a certain part of Southeast Asia. But there are multiple types of research that are close to *BaghChal* and are played on board with particular strategies. Previous works have implemented the *BaghChal* programs using search algorithms based on the different variations of alpha-beta pruning [5]. Prior works have also claimed that the game of tigers and goats at optimal is draw [4]. However, our work is focused on the implementation of deep reinforcement learning which combines idea of AlphaGoZero and AlphaZero [1].

The overall architecture of the game we have implemented is shown in the figure 1. Firstly, we collect the data from different sources like using random simulation algorithm, Minimax algorithm and some human play data. The data thus collected will be stored in replay buffer. This data in replay buffer is then passed to deep neural network architecture so that we can pre-train our neural network. This neural network will contain policy and value network which we will discuss in detail in out deep neural network section. The network will be then used to initialize Monte Carlo Tree Search Algorithm. This will provide updated states for the game *BaghChal*. We will use this output to update replay buffer which is again used to train neural network. The model is trained after it goes through few iterations of model update and MCTS update with self play. Then the trained model will be evaluated by comparing it with result of Minimax algorithm and Random Simulation.

[1] Game is over when more than 5 goats are killed by tiger

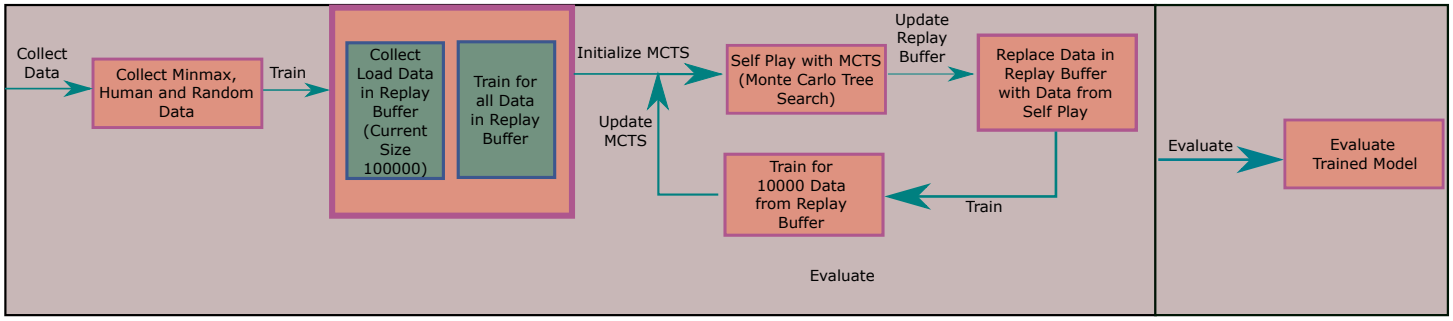


Figure 1: *Architecture of game BaghChal.*

3.1. Action Space Determination

Based on the rule of game, we evaluated the board to obtain the action spaces where each action is possible in the game but may not be valid based on the position of the game. The Deep Neural Network policy hence estimates the probability of each action. However, the actions that are predicted by the network might not be valid for both tiger and goat. We also listed out the connected locations in the board where the goat and tigers are allowed to move. Let's consider the board is represented from (1,1) to (5,5) with total 25 spots. We can see in the figure 3 that we can not make move from (1,2) to (2,1) because it is not connected, however it is possible to move from (1,1) to (1,2) because that path is connected. In summary, considering all possible actions that can be taken by either goat or tiger or both, it sums to 217.

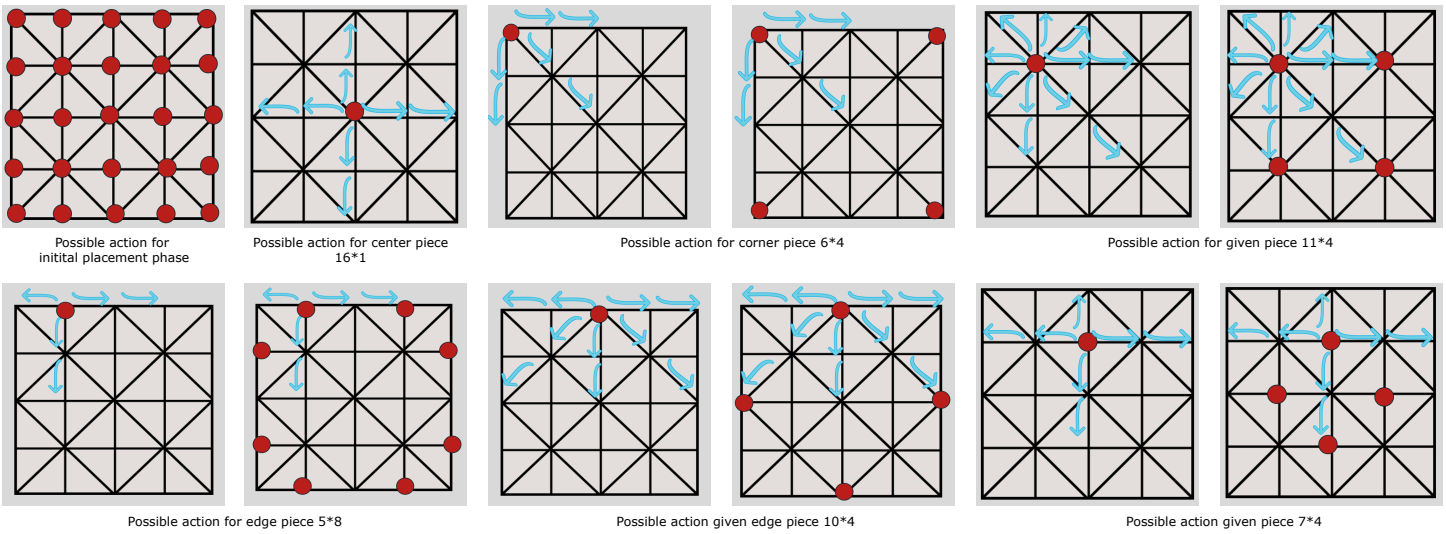


Figure 2: *Total possible actions that can be taken in board game of BaghChal.*

We also can see from figure 2 how the actions can be determined for each of the positions in the board. Initially for board with all empty position there are 25 possible places where player can be inserted. Then player can take action in 16 possible directions if its center piece, 6*4 for corner piece, 5*8 and 10*4 for edge piece and 11*4 and 7*4 for other pieces. The sum of all possible actions thus can be taken in board of *BaghChal* is 217.

3.2. Deep Neural Network

To teach an agent to learn self-play, we will utilize the policy ($\pi(a|s)$) and value function ($V(s)$) that are estimated from the Deep Neural Network. AlphaZero architecture that we have used in the paper uses the concept of having same neural network to predict policy and value. The same network will have two different output to predict policy and value. Policy of the network gives probability of next action to be chosen. We take the action with best probability. And value gives us the estimate of whether a player is likely to loose, make draw or win the game from the current state.

The output of both value and policy network is shown in figure 4. *BaghChal* game's state at any moment will be represented with the binary encoding of goat's position, tiger's position, how many tigers are trapped, and how many goats are killed, etc. Thus, the deep neural network will have a multidimensional binary representation of the board as the state. The deep residual network will use this multidimensional binary representation of the board to predict policy and value functions.

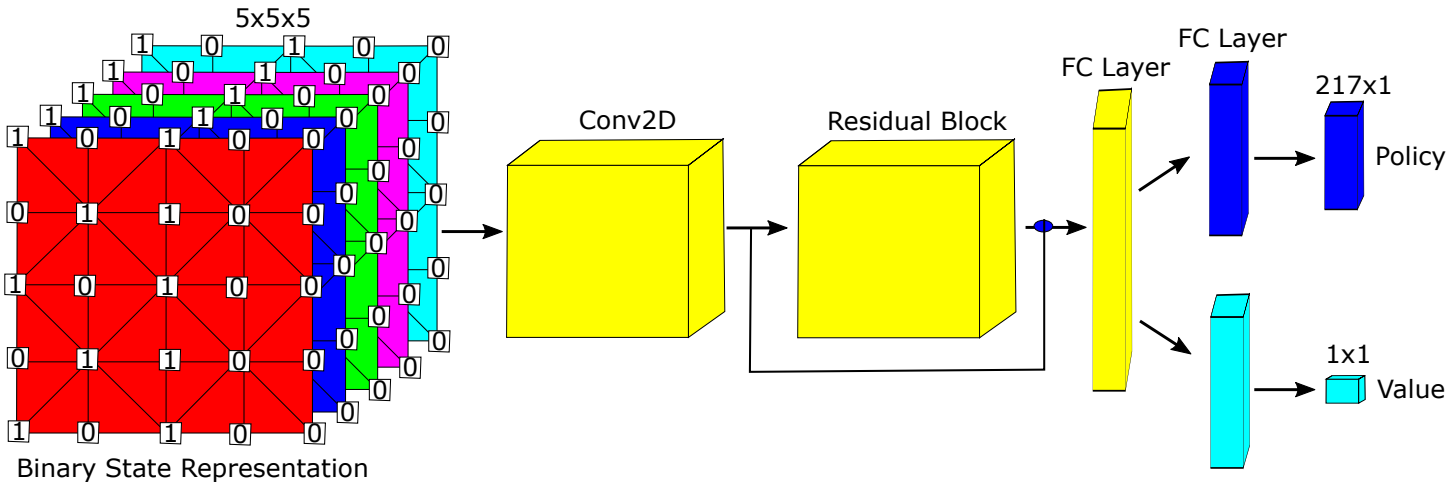


Figure 3: *BaghChal state representation (Red:Goat pieces, Blue: Tiger pieces, Green: number of goats captured, Pink: number of tigers trapped, Cyan: whose turn to play (Goat = 1, Tiger = 0)), Deep network architecture, and policy-state value.*

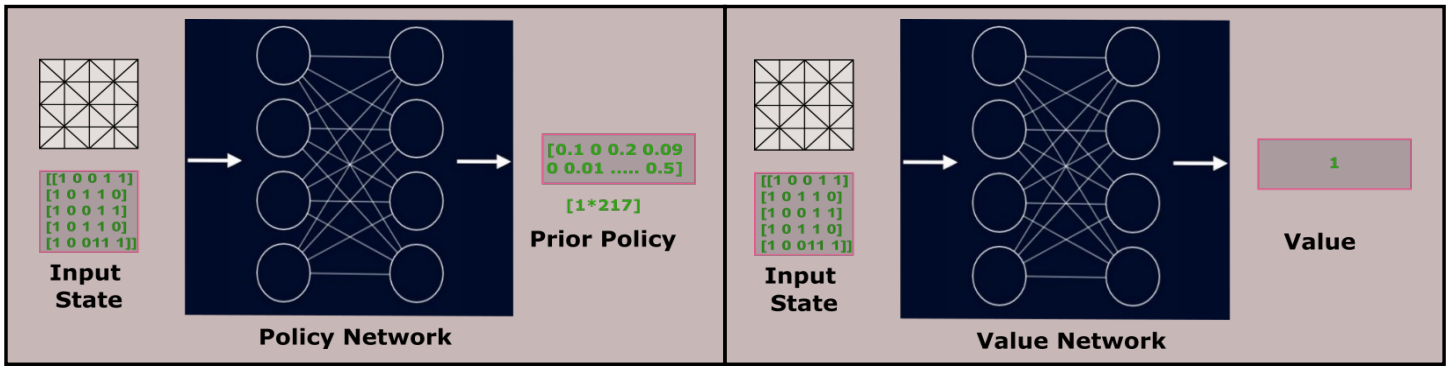


Figure 4: *Input and output of policy and value network.*

Network Loss Functions: To update the deep neural network, we are planning to use the cross-entropy loss (Eqn. 2, K represents the number of classes) for policy values because it represents the probabilities of taking action (K = 217) and MSE loss (Eqn. 1, N represents the number of samples in a batch) for value functions because it is similar to the regression problem. The Deep Neural Network (DNN) is trained with combination of these two loss values (Equation 3), where λ_1 and λ_2 are hyper-parameters that are tuned during training.

$$\mathbf{L}_{Value} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2} \tag{1}$$

$$\mathbf{L}_{Policy} = -\sum_{i=1}^{K} t_i log(c(s_i)) \tag{2}$$

where $c(s_i)$ and t_i are the predicted and actual probabilities of i^{th} class (categorical output).

$$\mathbf{L}_{DNN} = \lambda_1 \times \mathbf{L}_{Policy} + \lambda_2 \times \mathbf{L}_{Value} \tag{3}$$

3.3. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) uses random sampling for deterministic problems to solve difficult problems. In MCTS, the game is played to the end with random moves and updates the weight of nodes in the game tree so that better moves are more likely to get selected in the next sampling. In other words, MCTS tries to provide more chances in the next sampling for that moves which helps to win the game. As shown in figure 5, we repeat the process of select, expand and evaluate, and backup to update Q values based on the upper confidence bound [3; 6]. During backup, it updates the Q values of parent nodes (pink and green Q in figure 5). Additionally, it also generates the policy π which is select by expanding the tree to multiple ahead steps. Thus, during training we use Q and π obtained from MCTS to train our policy-value network. Finally,

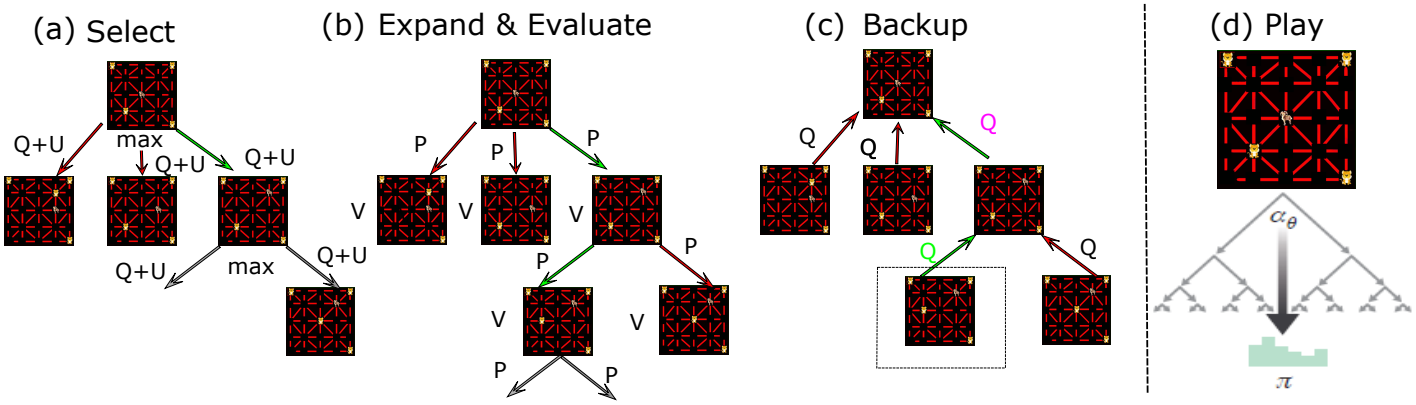


Figure 5: *Process of MCTS to generate samples for training; (a) Select, (b) Evaluate, (c) Backup, and (d) Play.*

the DNN Network learns to predict the optimal policy and state value after the repeated process of MCTS and train. In AlphaZero, they have 1600 game simulations, equivalent to 0.4 sec of thinking time on fast computing devices.

In our project we are using simulation from range 10 to 200. The more number of simulation means the more time and computational complexity. But it also means the algorithm is able to explore more board states in future.

3.4. Rewards

Finally, we are providing reward of 1 for the win, 0 for the draw, and -1 for the loss. The game is declared a draw if there are no possible or some repeating moves from both players or a maximum number of steps reached without any outcome. Rewards are also used to update Q values if MCTS simulation reaches end of the game [6]. While giving the rewards we had to increase the number of steps of draw from 3 to 20. With network having 3 steps repetition for draw due to [4] we were only having draw as outputs. Increasing the number to 20 let the bagh and goat explore for more steps so that it can produce loose or win as result.

4. Experimental Results

4.1. Model Architecture and Implementation

The model is implemented in Tensorflow API with python programming lanaguage. The details of network architecture used is shown in figure 6. We are using two convolution layer with width 64, 128 followed by five resnet block of with 64 and convolution layer of width 64. We have explored MCTS with varying number of simulations from 10 to 50. The learning rate used for neural network is 2e-3. The total data we have used for pre training neural network before starting self play is 200. While selecting the buffer size, we have observed that selecting 10K is very low and also we will have states in range of 100-200 in a single game (also depends on game length). Thus, it is suffering from catastrohphic forgetting with low buffer size. Hence, we increased the buffer size to 100K to include most experiences. However during selfplay, we only select 10K states out of 100K to update the policy-value network after successive games.

4.2. Training with Prior Data

We are using training data that is generated from random sampling algorithm, Minimax algorithm and and also have human play data. The minmax player uses the various depths to mimic different level of intelligence from computer. In *BaghChal*, we have used the tree depth from 4 to 6 and collected 50 games for each on average. Games with depth 6 takes higher amount of time to select the game and complete the game. We also observed that based the depth, the winner of the game are decided. With depth 4, we were seeing game being draw while with depth 5 Bagh is winner in most of instances and Goat is winner with depth 6. This helps to include wide range of data for the policy-value network to train. In addition to this, we have few human play data that also contains some moves which we found to be effective in winning the game.

The main reason to include these observations to pretrain the policy-value network before self is that we tried training neural network with self play generated data only as mentioned in AlphaZero paper and we were not able to see the game of *BaghChal* learning from only self play data. So, we also took the idea from AlphaGoZero paper to include dataset before passing it to self play architecture. This improved learning capability of network.

4.2.1. Network Losses. The network loss with previously collected data are provided to the policy-value net and trained upto certain number of epochs. The training curve is show in figure 7(b).

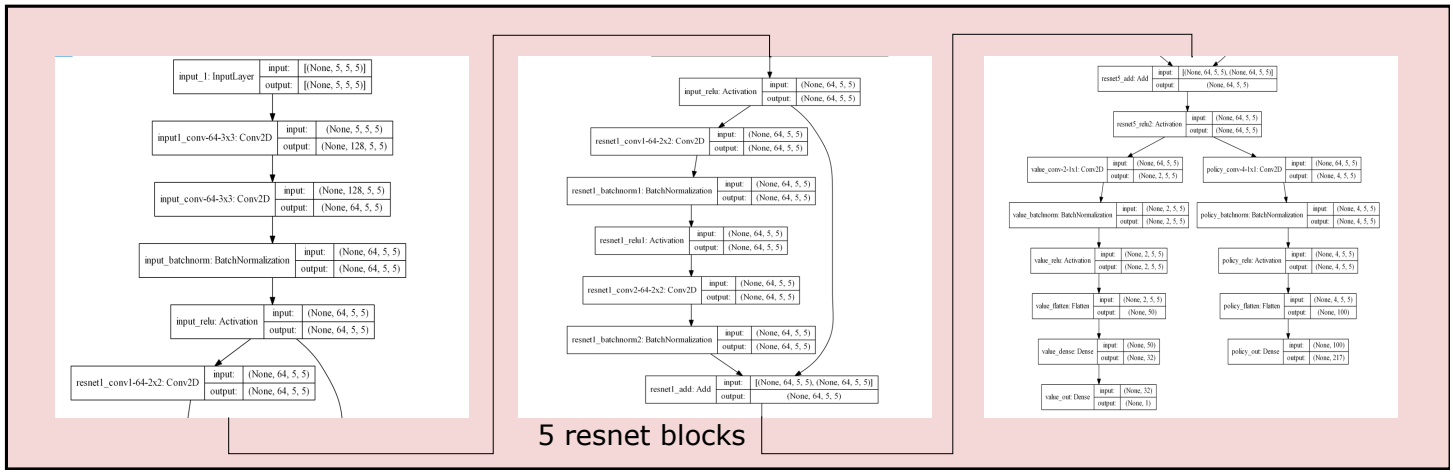
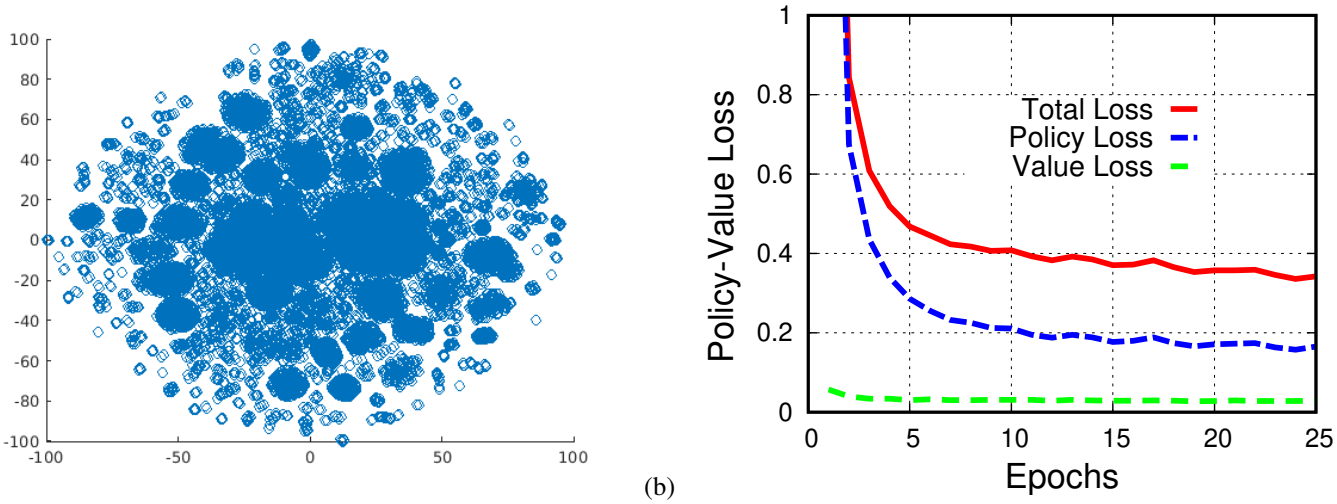


Figure 6: *Architecture of game BaghChal.*



(a) . (b)

Figure 7: *BaghChal's ; (a) t-SNE of game states for different games played with trained MCTS player. (b) Policy and Value Loss with previously collected samples.*

4.3. *BaghChal*'s Seflplay

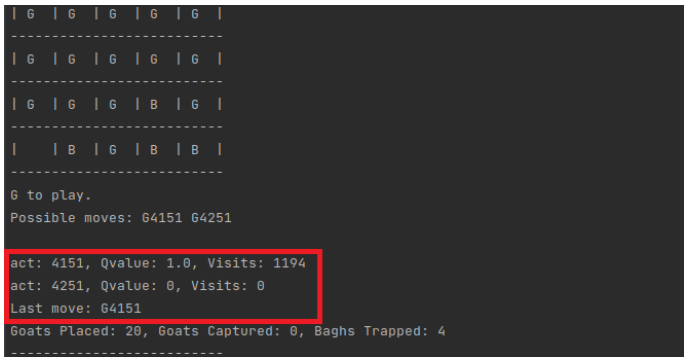
In addition to the training with prior data, we also allow the *BaghChal* to selfplay and gain experience by itself. During, selfplay we randomly initialize the policy-value network and with empty buffer. As it collects the experience through play, it pushes those observations into replay buffer and learns from that experiences gathered through MCTS. We allowed the network to play 1000 games in total in selfplay mode without any human or computer played data. The model's performance is less efficient than the policy-value network with prior data available for training. We left the exploration of reasoning in details for future work.

4.4. Abilation Study of *BaghChal*

After the network is trained with collected data from human play, random play, minmax play. We train the network for certain duration to allow network to start from some knowledge. While comparing the states that are visited during the game, we found that most of states are congregated together, thus during each game, either player only visits certain states that are close to each other as shown in figure 7(a). This also infers that, our MCTS policy-value network can generalize over states that have not been observed yet and still works.

We observe that playing as Bagh with other players such as Random, MinMax, and untrained networks has winning ratio of 0.95, 0.65, and 0.98. We observe that it leads to draw the game sometime with MinMax player because the MinMax player takes move more thoughfully than other two players *i.e.* Random and Untrained. This result indicates that RL agent can learn from various experiences to update its policy and finally start playing better. The results holds true even when RL agent is playing as Goat.

(a)



(b)

Figure 8: *BaghChal's MCTS stages; (a) Where Game is won by Goat by taking a step G4151. i.e. Q-value = 1.0. (b) Where Bagh has only one possible move to play indicating its state value negative i.e. at loosing stage of game.*

4.5. Key Observations on MCTS

We observed certain states where Goat is winning game by taking action which has highest visits among all possible action as shown in figure 8(a), where Q-value is highest for that particular action. We observe in this case that Goat wins the game in next step by taking the action G4151 out of possible actions *i.e.* G4151 and G4251. However, in figure 8(b) Bagh player has only move B2231 and it has 3 of its Bagh are already trapped. At this stage, bagh is more likely to lose than win, which is indicated in Q-value as -0.61. Q-value has range of [-1, 1], where values close to +1 represents the most likely to win and values close to -1 represents most likely to lose from that player's perspective.

4.6. Contributions of the paper

(a) Finalize the action space and valid moves of the game. (b) Design the Deep Neural Network architecture and able to train with randomly generated data. (c) Study and understand the MCTS and how we can generate the state, π, and Q pair that are required for the Deep Network training.

5. Future Milestones

In this project *BaghChal*, we have completed the training and evaluation of an RL agent to learn and play BaghChal game. We have tried various algorithms from AlphaGo Zero and AlphaZero paper to improve an agent. However, due to limitation of available enormous data and computaional resources we are not able to explore the game upto its extend. Thus, in future, we can explore more variation of different depths of minmax player, different number of simulations of an agent and train the network with other types of architecture than resnet to improve the model. Additionally, we would like to explain and observe the moves with explainable AI such as Inductive Logic Programming.

6. Conclusion

In this project, we have achieved the self-intelligence in an agent for the traditional *Nepali* game, *BaghChal*. Our goal is to introduce ourselves into the field of reinforcement learning and self-learning with help of the *BaghChal* project where we successfully implement and the results on various settings. We believe the knowledge acquired on this project can be extended to various robotics, medical and educational sectors.

References

[1] David Silver and Thomas Hubert and Julian Schrittwieser and Ioannis Antonoglou and Matthew Lai and Arthur Guez and Marc Lanctot and Laurent Sifre and Dharshan Kumaran and Thore Graepel and Timothy Lillicrap and Karen Simonyan and Demis Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," 2017.

[2] Werner Duvaud, "MuZero General," 2021. [Online]. Available: https://github.com/werner-duvaud/muzero-general

[3] Schrittwieser, Julian and Antonoglou, Ioannis and Hubert, Thomas and Simonyan, Karen and Sifre, Laurent and Schmitt, Simon and Guez, Arthur and Lockhart, Edward and Hassabis, Demis and Graepel, Thore and et al., "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, 2020.

[4] Albert, Michael H. and Nowakowski, Richard J., *Games of No Chance 3*, 1st ed. USA: Cambridge University Press, 2009.

[5] Donald E. Knuth and Ronald W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[6] Silver, David and Schrittwieser, Julian and Simonyan, Karen and Antonoglou, Ioannis and Huang, Aja and Guez, Arthur and Hubert, Thomas and Baker, Lucas and Lai, Matthew and Bolton, Adrian and Chen, Yutian and Lillicrap, Timothy and Hui, Fan and Sifre, Laurent and Driessche, George and Graepel, Thore and Hassabis, Demis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.