Empirical Analysis of Posterior Sampling as Policy Selection in MuZero

Saiham Rahman
Student ID: 210769593
School of Electronic Engineering and Computer Sciences
Queen Mary University of London
London E1 4NS, United Kingdom
ec21286@qmul.ac.uk

***Abstract*—DeepMind, in their latest paper, introduced MuZero, which combines deep learning (DL) and reinforcement learning (RL) for self-play to achieve human-level performances in games. The algorithm pUCT used in MuZero is provided without justification and valid mathematical proof. Recently, Posterior Sampling or Thompson Sampling techniques have shown state-of-the-art empirical performance in many industrial applications and can be extended to domains of MABs and MDPs problems. This Bayesian RL framework also has shown near-optimal performances in MCTS. In this paper, we will gain a background on Thompson Sampling and discuss its implementations in MCTS and MuZero. The objective of this paper is to provide an empirical analysis of Thompson Sampling as an alternative to the pUCT algorithm.**

***Index Terms*—Deep Learning, Deep Reinforcement Learning, Reinforcement Learning, Monte Carlo Tree Search, Posterior Sampling, Thompson Sampling, Probability Matching**

I. Introduction

In the past few years, the widespread applications of reinforcement learning (RL) have piqued the interest of researchers in the machine learning (ML) and artificial intelligence (AI) sectors. Along with supervised learning and unsupervised learning, RL is one of the three fundamental ML paradigms (Russell and Norvig 2010). RL aims to determine how intelligent agents should behave in each environment to maximise the cumulative reward in the terminal states. It contrasts with supervised learning, which depends heavily on labelled data and requires explicit corrections to any outputs which are not optimal. Unsupervised learning models use unlabelled training data to predict patterns or cluster data. RL, on the other hand, without a training dataset, will inevitably gain knowledge by interacting with the environment. Making decisions simultaniously is the foundation of RL. It generates input modelled from the previous output for the next output-input pair. A classic RL problem is a Multi-Armed Bandit (MAB) Problem (Katehakis and Veinott 1987)(Auer, Cesa-Bianchi, and Fischer 2002). It is a mathematical problem in probability theory and machine learning in which a bandit must pull arms in order to maximise its expected payoff. Each of the bandit's decisions is only partially known at the time of initiation and will become more completely understood as time passes or the statistics for the chosen arm are updated. This classic problem illustrates the importance of balancing the dilemmas of exploration (of previously unknown knowledge) and exploitation (of discovered knowledge). For the MAB problems, algorithms such as Epsilon-greedy(Sutton and Barto 2018), Upper Confidence Bounds for contextual bandits(Li et al. 2010), Probability matching strategies (Scott 2010), also known as Thompson Sampling(Thompson 1935)(Chapelle and Li 2011), Epoch-greedy(Langford and Zhang 2007), and Exp3 for Adversarial bandits(Seldin et al. 2013) are popular. Recently, RL has had a significant impact on the use of Probability matching strategies. According to research, it outperformed popular bandit algorithms in solving MABs and Markov Decision Process (MDP) problems(Scott 2010)(Granmo 2010)(Chapelle and Li 2011)(Garivier and Cappé 2011).

The concept behind Thompson sampling (Thompson 1933) in MAB's is that the number of pulls for a particular lever should correspond to its actual probability of being the optimum lever(Malcolm 2000).It involves selecting the action to take that maximises the expected reward compared to a randomly chosen belief. If we can sample from the posterior for the mean value of each action, it is remarkably simple to implement(Scott 2010)(Chapelle and Li 2011)(D. J. Russo et al. 2018)(D. Russo and Van Roy 2014). The dilemma of exploration is a critical component of RL. This relates to the issue of how to decide the agent's action to take as it gains knowledge of the environment. In contrast, during the exploitation phase, actions are chosen to maximise the predicted reward relative to the estimate provided by the value function. Exploration and exploitation can be naturally balanced in a valid mathematical framework in the Bayesian RL framework(Malcolm 2000). Polices over the entire information state (or belief), including model uncertainty, can be expressed. In that scenario, the best Bayesian strategy would be to choose actions based on the rewards they obtain as well as the amount of knowledge they store on the domain's parameters, which might subsequently be used to increase rewards. By combining tree search methods with posterior sampling approaches, MCTS can locate near optimum policies in the domains of MDPs. MCTS with sampling approaches has had remarkable success in game play, planning under uncertainty and Bayesian reinforcement learning (Gelly and Silver 2011)(Mark, Björnsson, and Saito 2010)(Silver and Veness 2010)(Wu, Zilberstein, and Chen 2011)(Guez, Silver,

and Dayan 2012)(Asmuth and Littman 2011).

Deep Reinforcement Learning (DRL) is a fairly new approach in the field of ML; it combines Deep Learning (DL) and RL. DL uses an artificial neural network to convert a set of inputs into a set of outputs. The high-dimensional states of the MDPs, such as raw images from an Atari game or in many real-world decision-making issues, prevent typical RL algorithms from solving them. In order to solve such MDP problems, DRL approaches use deep learning, frequently modelling the learned functions as neural networks and using algorithms that would excel in these high-dimensional states and environment. DeepMind demonstrates strong learning using deep RL to play Atari video games. (Mnih et al. 2013) DeepMind's AlphaGo (Silver, Huang, et al. 2016) had taken a giant leap forward in the field of Artificial Intelligence and Reinforcement Learning(RL). It developed a super-human strategy for playing the game of Go and by defeating the world champion, its research has received much recognition. AlphaGo and its descendants employ a Monte Carlo Tree Search(MCTS) technique to determine their moves based on information previously obtained using a deep learning approach after extensive supervised training by playing both human and computer. The optimal movements and winning percentages of these moves are identified using a neural network. The strength of the tree search is improved by this neural network, resulting in better move selection in the following iteration. A variant of the algorithm AlphaGo Zero(Silver, Schrittwieser, et al. 2017) has introduced self-play, where the algorithm plays with itself in finding the best strategies that would beat its previous found strategies. It requires no supervised learning of expert moves and the policy net is updated to match MCTS actions. For which, it has outperformed AlphaGo. A couple of days later, AlphaZero (Silver, Hubert, et al. 2018) was introduced which was more generalized and can beat world-champion programs at chess and shogi. There are main improvements done over the previous algorithms. Firstly, AlphaZero can be applied to any game having perfect information. That is, no other information about the game is required except for fully knowing the game state and rules of the game. Secondly, restructuring the input and output representations for chess and shogi while also restructuring the self-play algorithm.

DeepMind removed all assumptions from a specific dynamics model in its most recent study, MuZero(Schrittwieser et al. 2020). Hidden state was created in order to do MCTS with a learnt model initiated with a learned root state initialization. To put it another way, the algorithm's main goal is to predict behaviours that are directly related to planning. The model receives the observation and converts it into a hidden state. A recurrent process that receives the previous hidden state and a predicted next action updates the hidden state constantly.The model forecasts the policy (the move to play), value function (the expected mean), and immediate reward at each of these phases. The model is typically trained with the sole purpose of precisely predicting these three critical values in order to match the more accurate policy, value, and reward predictions given by search. This unique strategy for model-based RL algorithms achieves state-of-the-art performance in visually demanding RL domains in addition to Go, Chess, and Shogi. Furthermore, it outperformed model-free RL algorithms in complex Atari games.

The selection policy for MCTS introduced in AlphaZero later also used in MuZero uses a so-called pUCT algorithm. This pUCT algorithm, although effective, was provided without proper justifications. It contains complex hyper-parameters and constants which are more than a usual UCT algorithm in MCTS would have. As posterior sampling strategies are considered to be a valid mathematical framework in MCTS for Bayesian learning, we would propose posterior sampling as an alternative to the pUCT algorithm. In this paper we will be studying different selection policies for MuZero, focusing on Thompson Sampling. In this paper, we will be discussing two main variants of Thompson Sampling, namely, Bernoulli Thompson Sampling and Gaussian Thompson Sampling.

Preliminary experimentation on the selection policies were conducted in an MCTS environment and the concept was transfered over to the MCTS in MuZero. In MCTS environment by experimentation it was found that Gaussian Thompson Sampling reached near optimal solutions almost as similar to the classic UCT approach. Similar performances could be found for Gaussian Thompson Sampling in MCTS for MuZero as well but pUCT manages to attain zero regret at a smaller number of training steps compared to Gaussian Thompson Sampling. Bernoulli Thompson Sampling performance was inferior to both UCT and Gaussian Thompson Sampling in the MCTS setting, as well as, in MuZero to both pUCT and Gaussian Thompson Sampling.

The rest of the paper is organized in sections, firstly, providing descriptions of the background required in the methodology section. Then we will describe the posterior sampling methods and how it was implemented in MCTS and MuZero. After describing the experimentation environments, we will be discussing the empirical results of the agents and conclude the paper with some brief idea of our future work.

II. Methodology

A. Bandit Algorithms

Bandit Algorithms is the term used to define a set of Reinforcement Learning Algorithms which aim to solve the exploration-exploitation dilemma. These algorithms are used as a way to find the optimal solution in a given problem set. The most common problem set in probability theory is the Multi-Armed Bandit problem. Algorithms such as Greedy, Epsilon Greedy, Optimistic Greedy, Upper Confidence Bounds(UCB), and Thompson Sampling along with some variations of these algorithms are used to attain the maximum reward possible from a machine which provides rewards based on a probability distribution for that machine. The objective of the algorithms is to attain new knowledge based on trials and provide an optimal decision on which lever to pull next to attain more rewards based on exploration. Once the algorithm has enough knowledge about the certain arms with

Algorithm 1 Thompson Sampling

Initialize prior statistics for some distribution $f(x_0)$
for each run $t = 1, 2, \ldots, N$ **do**
  for each arm $i = 1, 2, \ldots, K$ **do**
    Sample θ_i from the distribution $f(x)$
  end for
  Pick Arm $a \leftarrow$ *argmax*$_i \theta_i$
  Reward $r \leftarrow$ *Reward(a)*
  $f(x_t) \leftarrow$Update posterior distribution statistics
end for

the maximum rewards, it begins to exploit it to accumulate a greater sum of rewards.

B. Posterior Sampling

If the posterior distribution belongs to the same probability distributions as the prior probability distribution, the prior is known as a conjugate prior, the prior for the likelihood function, in Bayesian probability theory. The prior and posterior are known as conjugate distributions, and the prior is known as a conjugate prior, the probability function's prior. In Bayesian Statistics, Posterior Sampling is the method of sampling from a prior distribution. The method of updating the posterior beliefs based on the observed data is known as Bayesian Inference. As a general concept, posterior sampling extends in both the cases of bandit problems and MDPs. Posterior Sampling can be summarized into four main steps. The agent maintains a probability density over the action spaces, by sampling an instance according to the prior it then acts optimally based on the sampled instance. Using Bayesian Inference, the posterior probability densities are updated for those actions. In this way the agent will choose optimal actions where the probability density is concentrated around the true parameters. In this paper we will be discussing about Thompson Sampling also known as probability matching and posterior sampling proposed in 1933 (Thompson 1933) (Thompson 1935). The Algorithm (1) shows the general algorithm for Thompson Sampling. We choose one arm in each round of multi-arm bandit situations using Thompson Sampling based on the maximum score sampled from each arm's prior probability density. We then update the posterior distributions for each arm after each round. The effectiveness of the algorithms is supported by empirical data. Two significant variations of the algorithm, Gaussian Thompson Sampling and Bernoulli Thompson Sampling, will be used in our study.

1) Bernoulli Thompson Sampling: The behaviour of a random variable can be represented by the Bernoulli distribution when there are only two possible outcomes. The reward probabilities are modelled using Thompson Sampling. The Beta distribution is the best choice to depict this kind of probability when the rewards are binary. From the family of conjugate distributions, it can be said that when a value is drawn from a Bernoulli distribution and the resulting likelihood is multiplied by a value drawn from the prior probability distribution,

Algorithm 2 Bernoulli Thompson sampling

Initialize parameters $\alpha \leftarrow 1, \beta \leftarrow 1$
for each run $t = 1, 2, \ldots, N$ **do**
  for each arm $i = 1, 2, \ldots, K$ **do**
    Sample $\theta_i \sim$ *Beta*(α_i, β_i)
  end for
  Pick Arm $a \leftarrow$ *argmax*$_i \theta_i$
  Reward $r_t \leftarrow$ *Reward(a)*
  if $r_t > 0$ **then**
    $\alpha_{i+1} \leftarrow \alpha_i + 1$
  else
    $\beta_{i+1} \leftarrow \beta_i + 1$
  end if
end for

which is a Beta distribution, then the resultant value from the posterior probability distribution also has a Beta distribution.

Algorithm (2) summarizes Bernoulli Thompson sampling in a MAB environment. Alpha and beta are the two parameters needed for the beta distribution. These variables can be viewed as the number of successes and failures, respectively.Prior Probability is the probability that an event will occur before we have any knowledge; in this case, it is represented by the Beta distribution Beta(1,1). It is the first estimate of the likelihood of the arm producing output. We can modify our belief of how likely it is that an arm will pull the lever after testing it and receiving a reward. The Posterior Probability is generated after some knowledge has been collected through trials. This is given by a Beta distribution where the values of alpha and beta have been updated to reflect the value of the returned reward. As the number of trials of the arms rises, so does confidence in the estimated mean; this is reflected in the probability distribution narrowing, and the sampled value will be selected from a range of values closer to the true mean. As a result, exploration declines, and exploitation grows as the arms with a higher probability of pulling the lever and providing a reward are chosen with increasing frequency. Arms with a low estimated mean, on the other hand, will be picked less frequently and will be discarded earlier in the selection process. As a result, their true mean may never be found.

2) Gaussian Thompson Sampling: The simplest model was to assume an environment that emits binary rewards. In the previous section, we used Bernoulli Thompson Sampling by modeling the outputs to have binary outcomes for updating the posterior. When a value is drawn from a Bernoulli distribution, we get the conjugate prior with a Beta Distribution. Bernoulli Thompson Sampling only works when we have a discrete set of rewards. However MDPs with large action spaces are characterised by more complex reward functions. For those cases, it is safe to assume that the rewards are Gaussian Distributed. In Gaussian Thompson Sampling, we model the rewards using a normal distribution and by updating its mean and variance parameters we refine this model.

The general algorithm for Thompson Sampling is as the Algorithm (3). Here, we assume that the rewards have a fixed

Algorithm 3 Gaussian Thompson Sampling

Setting initial precision λ_y, λ_0
Initial mean $\mu_0 \leftarrow 0$
for each run $t = 1, 2, \ldots, N$ **do**
  for each arm $i = 1, 2, \ldots, K$ **do**
    Sample $\theta_i \sim \mathcal{N}(\mu_t, \lambda_t^{-1})$
  end for
  Pick Arm $a \leftarrow$ *argmax*$_i \theta_i$
  Reward $r_t \leftarrow$ *Reward(a)*
  Mean $\mu_{t+1} \leftarrow \dfrac{\lambda_y \cdot r_t + \lambda_t \cdot \mu_t}{\lambda_y + \lambda_t}$
  Precision $\lambda_{t+1} \leftarrow \lambda_t + \lambda_y$
end for

precision λ_y. Where λ_y is just the inverse of the variance σ^2, $\lambda_y = 1/\sigma^2$. Similarly, we assume a prior precision λ_0 for the unknown reward. Using this prior precision and mean μ, we sample from scores for each arms from the distribution. By taking the argmax of the scores, we choose the action a and observe the outcome. The reward mean generated from the environment following given action is assumed to be from a Gaussian Distribution with fixed precision λ_y. After then, we update the the mean μ_0 and total precision λ_0. This posterior update will help the agent to get closer to the true mean of the distributions. After exploring all the arms the true means will be found and the arms with highest means will chosen more frequently.

C. Markov Decision Process

Markov decision processes (MDPs) (BELLMAN 1957) provide a general framework for planning and learning under uncertainty. An MDP is considered a probabilistic state model where the environment is fully observable. The most common MDP is the Discounted-Reward MDP. It can be defined as a tuple containing $(S, s_0, A, P, r, \gamma)$. Here, S is considered the state space of the environment, A defines the action space in the MDP, usually these can be small or large action spaces. A(s) is the action taken in that state, $P_a(s^t|s)$ denotes the probability of reaching the state s^t starting from the state s. r(s,a) is the reward emitted by the state-action pair traversing to state s^t given the action A(s) from state s. MDPs are mainly used to find an optimal policy which would maximise the accumulated expected reward from the initial state s_0. To do this, the agent uses a policy π function which tells the agent the decision rule to map states to actions $\pi: \rightarrow$ A and which action $\pi(s)$ should it take in each of the states. Therefore, the expected reward can be defined as the following equation (1).

$$V^{\pi}(s) = E_{\pi}[\sum_{i} \gamma^i r(s_i, a_i, s_{i+1}) \mid s_0 = s, a_i = \pi(s_i)] \quad (1)$$

where, $V^{\pi}(s)$ is the expected discounted accumulated reward value following the policy π from initial state s_0 and by transitioning the states with the action selected according to the policy $\pi(s_i)$ at each time step i.

D. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an online learning algorithm. Its basic framework is to build a tree based by evaluating a state-action pair. Each node of the tree stores the number of times the node has been visited, pointers to the parent node and a set of children nodes. In solving MDPs, MCTS incrementally builds the search tree until a termination condition has been reached. There are four main steps of the algorithm. In the selection stage, it selects an unexpanded node with at least one child node, the best action is selected using a bandit algorithm action selection strategy. During expansion, a child node is added to the search tree which was reached optimally using a selection policy. The resulting state is then evaluated using a rollout policy by recursive simulations until a terminal state or condition has been met, descending through the search tree from the root node. The statistics of the selected nodes are then updated by backpropagating from the new node to the root node.

UCB1 is a popular bandit algorithm used with MCTS. The approach is termed as UCT and is used to model planning under uncertainty in MCTS. Algorithm 4 refers to the implementation of UCT in an MCTS setting. In the algorithm, we get a UCB score for all the child nodes from the leaf node. Based on the score, the node with the best action is expanded and added to the search tree. The statistics for the The exploration constant added to the mean value $Q(s, a)$ makes the agent explore more nodes and based on the optimal path of the search tree, the nodes with the least UCB scores are discarded.

Bernoulli Thompson Sampling when used as an action selection policy in MCTS, the UCB scores are replaced by sampling from Beta Distributions. The parameters for the distribution are initialized as α and β. Based on the maximum score of the actions from the Beta Distribution, the node is expanded. The posterior statistics are updated based on the value of the accumulated discounted return. Algorithm (5) summarizes the Bernoulli Thompson Sampling implementation.

In Gaussian Thompson Sampling for MCTS, the scores are sampled from a Gaussian Distribution. The parameters for this distribution are λ_0, μ_0 and fixed precision λ_y. Using the inverse of λ_0 to get variance σ^2, we sample the nodes to get the best score. The distributions where the mean is close to the true mean will have a higher score. Algorithm (6) shows the working process of Gaussian Thompson Sampling in MCTS. The action with the highest score is used to expand the node. After observing the new mean, the posterior λ_0 and μ_0 are updated for the distribution of that node.

E. MuZero

1) Neural Networks for Representation, Dynamics and Prediction model: The MuZero paper shows that it does not matter if it does not understand the game's rules. By building a dynamic environment model in its memory and maximising its returns, MuZero learns how to play the game. MuZero

Algorithm 4 UCT selection policy in MCTS

Initialize value for constant c
for each episode $t = 1, 2, \ldots, N$ **do**
  for each child node $i = 1, 2, \ldots, K$ **do**
    UCB score $\theta_i \leftarrow Q(s,a) + c\sqrt{\frac{2\ln(\sum_b N(s,b))}{1+N(s,a)}}$
  end for
  Action $a_t \leftarrow argmax_i \theta_i$
  Observe $s_t \leftarrow Step(s_{t-1}, a_t)$
  Mean Value Reward $Q(s_t, a_t) \leftarrow r_t$ {discounted $r_t \in (-\inf \leq 0 \leq +\inf)$}
  Visit Count $N(s_t, a_t) \leftarrow N(s_{t-1}, a_{t-1}) + 1$
end for

Algorithm 5 Bernoulli Thompson Sampling selection policy in MCTS

Setting initial nodes' $\alpha_i \leftarrow 0$, $\beta_i \leftarrow 0$
for each episode $t = 1, 2, \ldots, N$ **do**
  for each child node $i = 1, 2, \ldots, K$ **do**
    Sample $\theta_i \sim Beta(\alpha_i + 1, \beta_i + 1)$
  end for
  Action $a_t \leftarrow argmax_i \theta_i$
  Observe $s_t \leftarrow Step(s_{t-1}, a_t)$
  Mean Value Reward $Q(s_t, a_t) \leftarrow r_t$ {discounted $r_t \in (-\inf \leq 0 \leq +\inf)$}
  Visit Count $N(s_t, a_t) \leftarrow N(s_{t-1}, a_{t-1}) + 1$
  if $Q(s_t, a_t) > 0$ **then**
    $\alpha_{i+1} \leftarrow \alpha_i + 1$
  else
    $\beta_{i+1} \leftarrow \beta_i + 1$
  end if
end for

requires three networks namely prediction, dynamics and representation networks to do it. Using the representation network MuZero creates the game state. The game state that MuZero acts on is now a hidden representation that it learns how to evolve through a dynamics neural network. MuZero using a prediction neural network generates a reward and a new state by the dynamics network using the current hidden state and the selected action. Details of these networks can be found in the MuZero paper.

2) Inference MCTS and training: MuZero requires two inference functions to traverse the MCTS tree and provide predictions. The initial observation is passed to the initial inference network. The inference network then predicts the value, reward, and policy from the current position. For each action, the recurrent inference network is used to predict the next value, reward, and policy based on the current hidden state. The predictions produced by the recurrent inference are scaled by dividing it by the number of rollout steps to match the weighting of the initial inference. The predictions are compared by a loss function described in the MuZero paper. Using the loss function the three MuZero neural networks

Algorithm 6 Gaussian Thompson Sampling selection policy in MCTS

Setting initial precision λ_y, λ_0
Initial mean $\mu_0 \leftarrow 0$
Initial Reward $R_0 \leftarrow 0$
Initial Child Visit Count $N \leftarrow 0$
for each episode $t = 1, 2, \ldots, N$ **do**
  for each child node $i = 1, 2, \ldots, K$ **do**
    Sample $\theta_i \sim \mathcal{N}(\mu_t, \lambda_t^{-1})$
  end for
  Action $a_t \leftarrow argmax_i \theta_i$
  Observe $s_t \leftarrow Step(s_{t-1}, a_t)$
  Mean Value Reward $Q(s_t, a_t) \leftarrow r_t$ {discounted $r_t \in (-\inf \leq 0 \leq +\inf)$}
  Mean $\mu_{t+1} \leftarrow \frac{\lambda_y \cdot Q(s_t, a_t) + \lambda_t \cdot \mu_t}{\lambda_y + \lambda_t}$
  Precision $\lambda_{t+1} = \lambda_t + \lambda_y$
  Visit Count $N(s_t, a_t) \leftarrow N(s_{t-1}, a_{t-1}) + 1$
end for

are trainned simultaneously. The main MCTS function loops through a number of simulations, the simulation steps are usually defined. The simulations are then passed through the MCTS tree until it reaches a leaf node. The optimal path found gets its statistics updated by backpropagating up to the search tree. A general overview of the process would be, at first in the selection stage, MuZero compiles a list of all the actions which were performed since the game has started. Taking the root node as the current node the search path is updated. Note that the search path now includes only this node. MuZero first descends the MCTS tree by using highest pUCT score to select the nodes to be explored. In the exapansion stage, MCTS expands a new node from the leaf node, and computes the expected reward and generates the next hidden state by the dynamics function. The new node is then added to the search tree, and the policy and value are computed by the prediction network. Each edge from the newly expanded node is then initialized by setting values for N(s,a),Q(s,a),P(s,a) which denotes number of times the node has been visited, the mean value, and policy respectively. Per simulation, the search algorithm calls the dynamics function and prediction function only once and updates the statistics of the edges from the newly expanded node are not updated at this point. Therefore, the leaf node is expanded by creating new children nodes. Each of those children are also assigned a corresponding policy prior. MuZero creates a node for all the actions available in the game as information about the actions being legal are not provided. Based on the chosen action, the environment generates intermediate rewards. These rewards usually have a discount γ of less than 1. In the backup stage, internal nodes estimates are generated from the cumulative discounted reward. These estimates v^l are bootstrapped from the value function from the value network. The statistics for Q(s,a) and N(s,a) are later updated and back-propagated up the tree and along the search path. In adversarial games, the value can

Algorithm 7 MuZero policy selection using pUCT in MCTS

Initialize $N(s,a), Q(s,a), R(s,a)$
for each episode $t = 1, 2, \ldots, N$ **do**
  for each child node $i = 1, 2, \ldots, K$ **do**
    Mean Value $Q(s,a) \leftarrow R(s,a) + (\gamma * \frac{\sum_b G}{N(s,a)})$
    pba = $P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$
    pbc = $\log \frac{\sqrt{\sum_b N(s,b) + c_2 + 1}}{c_2}$
    Sample $\theta_i \sim Q(s,a) + (pba \cdot pbc)$
  end for
  Action $a_t \leftarrow$ *argmax*$_i \theta_i$
  Observe $s_t \leftarrow$ *Step*(s_{t-1}, a_t)
  Visit Count $N(s_t, a_t) \leftarrow N(s_{t-1}, a_{t-1}) + 1$
  Reward $R(s_t, a_t) \leftarrow$ *Reward*(s_t) {discounted Reward}
  $V^l \leftarrow$Value from the Value Network
  Value $G_t \leftarrow R(s_t, a_t) + (\gamma \cdot V^l)$
end for

Algorithm 8 MuZero policy selection using Bernoulli Thompson Sampling in MCTS

Setting initial values for $\alpha_i, \beta_i, N(s,a), R(s,a)$
for each episode $t = 1, 2, \ldots, N$ **do**
  for each child node $i = 1, 2, \ldots, K$ **do**
    Sample $\theta_i \sim$ *Beta*$(\alpha_i + 1, \beta_i + 1)$
  end for
  Action $a_t \leftarrow$ *argmax*$_i \theta_i$
  Observe $s_t \leftarrow$ *Step*(s_{t-1}, a_t)
  Visit Count $N(s_t, a_t) \leftarrow N(s_{t-1}, a_{t-1}) + 1$
  Reward $R(s_t, a_t) \leftarrow$ *Reward*(s_t) {discounted Reward}
  $V^l \leftarrow$Value from the Value Network
  Value $G_t \leftarrow R(s_t, a_t) + (\gamma \cdot V^l)$
  if $G_t > 0$ **then**
    $\alpha_{i+1} \leftarrow \alpha_i + 1$
  else
    $\beta_{i+1} \leftarrow \beta_i + 1$
  end if
end for

be negative based on which player is the current player. The process stops once a number of simulations has gone through the tree, and an action is selected depending on how many times each child node of the root has been visited which means that the likelihood of choosing any action is inversely correlated with the number of times it has been visited. Finally, the selected action is performed in the actual environment, and relevant values are updated. The values are then used as training data for the neural networks Until the end of the game, this procedure is repeated, starting a new MCTS tree from scratch each time and utilising it to decide on an action.

F. Selection Policies in MuZero

1) pUCT: The pUCT selection strategy in the MCTS process is designed to spend more time exploring actions which may have high reward estimates and once it has explored enough, the actions which are not as rewarding are dropped early in the process. The Algorithm (7) summarizes the selection policy in MuZero. A brief discussion of the algorithm can be referred from the previous Section(II-E2), also the details of the algorithm and it's variables are discussed in-dept in the MuZero paper. The score is a measurement that equalises the estimated value of the action Q(s,a) with an exploration bonus based on the prior probability of selecting the action P(s,a) and the number of times N(s,a) the action has already been chosen. The exploration bonus predominates early in the simulation, but as the total number of simulations rises, the value term becomes more significant. This continues until it eventually arrives at a leaf node which has not yet been expanded.

2) Bernoulli Thompson Sampling: Algorithm (8) summarizes the Bernoulli Thompson Sampling selection policy in MuZero. The way it's modeled to work with MuZero is by updating the posterior statistics in the backup phase of muzero. The algorithm starts by initializing the α and β values. Initialization of variables required by MuZero are not shown to keep the pseudocode concise. MCTS using this new selection policy samples scores of the all the actions from Beta distributions of the child nodes using the prior statistics. Taking the argmax of those actions, the node is expanded using the optimal action. When updating the posterior statistics, The Value V^l from the value prediction network multiplied by γ is added to the previous accumulated discounted rewards to get a value G_t. Based on the value G_t, α and β values are updated.

3) Gaussian Thompson Sampling: Gaussian Thompson Sampling as given in Algorithm (9) is modeled to work with MuZero by sampling nodes based on a Gaussian Distribution and updating the posterior statistics in the backup phase of muzero. The algorithm is initialized with the prior statistics λ_0, μ_0 and fixed precision λ_y values. Taking the inverse of λ_0 to get variance σ^2 we sample from the distributions of the nodes and get action scores. These samples are generated using the prior statistics. By taking the argmax of those actions, the node is expanded using the optimal action. After which, using the value V^l from the value prediction network, The value V^l multiplied by γ is added to the previous accumulated discounted rewards to get the cumulative discounted reward value G_t. Using this value G_t, we update the the posterior precision λ_0 and mean μ_0. This posterior statistic will be used as prior the when the node is again visited and sampled.

III. Experimentation

Experiments were conducted using the open-source code(Werner Duvaud 2019) for MuZero. Modifications were done in the self_play python file in such a way that the Posterior Sampling strategies could work with MuZero. The games used for experimentation in MuZero are simple_grid, tictactoe and gridworld. Preliminary experiments were done in an MCTS environment, the template code for MCTS was provided by the project supervisor. In the MCTS environment,

Algorithm 9 MuZero policy selection using Gaussian Thompson Sampling in MCTS

Setting initial precision λ_y, λ_0
Initial mean $\mu_0 \leftarrow 0$
Initial Reward $R_0 \leftarrow 0$
Initial Child Visit Count $N \leftarrow 0$
for each episode $t = 1, 2, \ldots, N$ **do**
  for each child node $i = 1, 2, \ldots, K$ **do**
    Sample $\theta_i \sim \mathcal{N}(\mu_t, \lambda_t^{-1})$
  end for
  Action $a_t \leftarrow argmax_i \theta_i$
  Observe $s_t \leftarrow Step(s_{t-1}, a_t)$
  Visit Count $N(s_t, a_t) \leftarrow N(s_{t-1}, a_{t-1}) + 1$
  Reward $R(s_t, a_t) \leftarrow Reward(s_t)$ {discounted Reward}
  $V^l \leftarrow$Value from the Value Network
  Value $G_t \leftarrow R(s_t, a_t) + (\gamma \cdot V^l)$
  Mean $\mu_{t+1} \leftarrow \dfrac{\lambda_y \cdot G_t + \lambda_t \cdot \mu_t}{\lambda_y + \lambda_t}$
  Precision $\lambda_{t+1} = \lambda_t + \lambda_y$
end for

the game environment was a small frozen lake environment. In both MCTS and MuZero, each of the policy selection strategies were rigorously tested and run at least five times to get the final averaged result. Hyperparameter search was conducted for Gaussian Thompson Sampling, it involved over 20 runs with 24 parallel executions of the MuZero games. The agents were trained for more than fifty thousand training steps in those games.

A. Environment

Experiments were performed on local machine and on the University's Compute Servers. Final experiments were done in Queen Mary's Apocrita HPC nodes. Each task used 32 compute nodes with a total of 24GB memory.

B. Hyper-parameter selection

We prefer to use the same hyperparameters from MuZero in order to simplify the analysis. Parameter values and network architectures can be referred from the individual games configuration files in the source codes. Two key hyperparameters were used to update the posterior of Bernoulli Thompson sampling, which are the initial values for the α and β. for the MCTS nodes. These parameter values were adjusted slightly during preliminary experimentation to make the agent more optimistic or pessimistic. After some experiments the parameters were left with the same value for initiation to keep the algorithm true to the original implementations. Also the for the Gaussian Thompson Sampling, two main hyperparameters are the prior precision λ_0 and the fixed precision λ_y values. These parameters are unique to each game. We conducted hyperparameter search using varied standard deviation σ values to get the optimal λ_0 and λ_y values for each of the games. The parameter values can be referred from the source codes provided with the supporting materials.

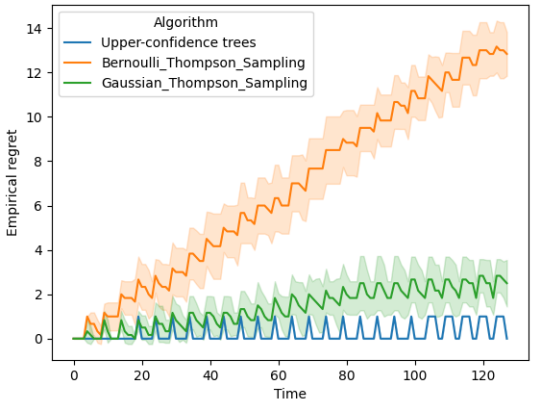

Fig. 1. Comparison of Thompson Sampling strategies with UCT in an MCTS environment

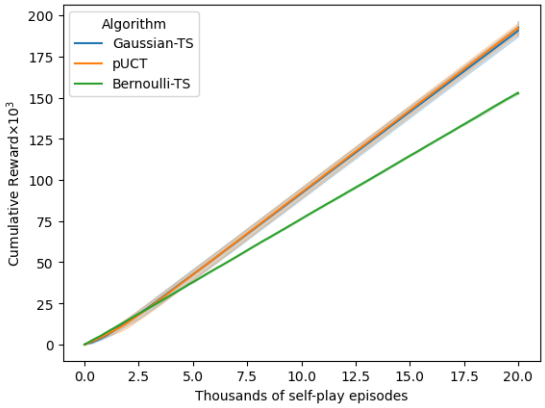

Fig. 2. Cumulative accumulated reward in MuZero simple_grid

C. Results

Using the selection policies for MCTS in the Frozen Lake environment, the figure (1) shows that Bernoulli Thompson Sampling had the most empirical regret. Gaussian Thompson Sampling depicts almost similar performances to UCT considering the small difference in empirical regret. Figure (2) and (3) depict that Bernoulli Thompson Sampling had the least cumulative accumulated reward compared to that of the other algorithms. Gaussian Thompson Sampling on the other hand, had near similar accumulation of rewards in fig.(2) for the game of simple grid where optimum hyper-parameters were found. In figure (4), in the adversarial game of tictactoe, Both Bernoulli Thompson Sampling and Gaussian Thompson Sampling had a similar cumulative reward to pUCT. After finding optimal parameters for Gaussian Thompson Sampling, the total cumulative reward accumulated by Gaussian Thompson Sampling was around 90-95% of that accumulated by pUCT. In the games of simple_grid and gridworld, figure (5) and (6) respectively, it can be seen that Gaussian Thompson Sampling had almost similar regret to pUCT up to a certain number of training steps.

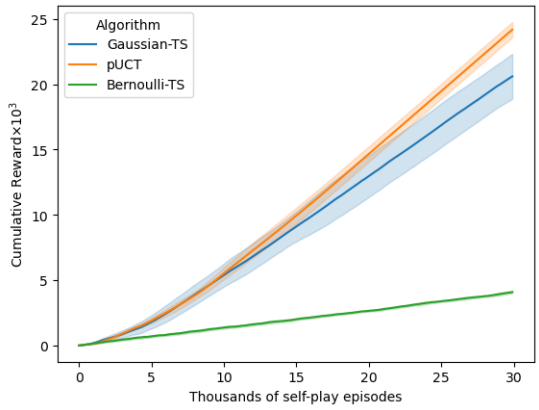

Fig. 3. Cumulative accumulated reward in MuZero gridworld

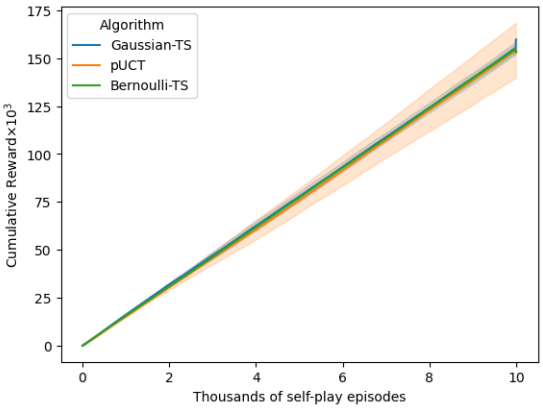

Fig. 4. Cumulative accumulated reward in MuZero tictactoe

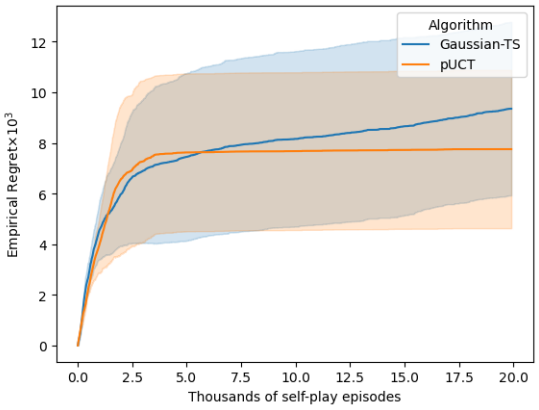

Fig. 5. Regret comparison between pUCT and Gaussian Thompson Sampling in MuZero simple_grid

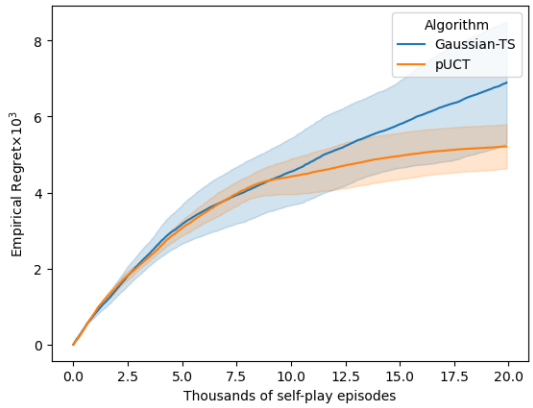

Fig. 6. Regret comparison between pUCT and Gaussian Thompson Sampling in MuZero gridworld

D. Analysis

For our evaluation of the algorithms, we plot the empirical regret and cumulative rewards for each of the games. In almost all the results, Bernoulli Thompson Sampling was the least performing algorithm. It is because, within the MCTS and MuZero, the rewards are considered as discounted rewards and the posterior update for the Beta distribution was inappropriate in this setting, as the rewards were not Bernoulli. Bernoulli Thompson sampling had the least accumulation of regards, it is because the rewards from the value prediction network are usually not discrete enough to be modelled as a Bernoulli trial. Whereas Gaussian Thompson Sampling models the rewards from the network as Gaussian Distributions to update the means and precision of the posterior. The empirical regret for Gaussian Thompson Sampling in MCTS and MuZero was almost similar to pUCT. The reason behind it being inferior might be because suitable hyper-parameters were not found for those environments. The main drawback of using Gaussian Thompson Sampling was that the results were found based on the assumption of a fixed precision. Although previous papers have shown better performances in their experiments with assumed precision. If suitable hyper-parameters were found for Gaussian Thompson Sampling, it would have been almost similar to or better than UCT in the experiments of this environment. Considering the original pUCT algorithm's results as baseline for comparison, we can see that pUCT has accumulated more rewards over both Bernoulli and Gaussian Thompson sampling over the same number of games played. Regardless, it is evident that Gaussian Thompson sampling has similar accumulation of rewards to the original algorithm upto a certain number of games played. It might be the case that Gaussian Thompson sampling is still exploring more nodes after a point or has started exploiting the actions it already knows.

IV. Conclusion

In our paper we have discussed about Posterior Sampling strategies and how it can be modeled to be used in the MuZero setting. To the best of our knowledge, these Posterior Sampling strategies were not applied in MuZero by anyone before us. Our paper, first provides a background on the required theoretical knowledge and then extend the theory to practical implementations with concise, easy to apprehend algorithms. Experimentation described the results of using Bernoulli Thompson Sampling and Gaussian Thompson Sampling in a

deep reinforcement learning setting like MuZero. Although the techniques applied were not as good performing as the original, these techniques have good theoretical background with valid mathematical proofs unlike the pUCT algorithm in MuZero which was provided without justifications. The main limitation of the project is that, due to the short time frame, experiments were only conducted in only three games. Also, due to the nature of Gaussian Thompson sampling depending on the fixed precision representing how much the reward given by an action will vary and prior precision representing how much we are unsure about the mean, optimal hyper-parameters were not found. The Posterior Sampling techniques mentioned in this paper were only modeled for deterministic games as MuZero only works with deterministic games. Regardless, It could be said that Posterior Sampling strategies could be a good alternative to costly solutions like pUCT. In future, experiments will be extended to stochastic environments for MCTS using Posterior Sampling. Posterior Sampling could also be extended to deep reinforcement learning agents for stochastic environments as MuZero only works with deterministic environments. Also, a new strategy could be introduced where both the mean and precision will be unknown, and the agent, by learning, will find the optimum parameters. When we have unknown mean and variance, we can use a normal-gamma where the parameters are unknown mean and precision. This strategy will remove the limitation of assuming the fixed precision in Gaussian Thompson Sampling.

Acknowledgement

The MCTS template code using UCT was provided by the project Supervisor.

This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. http://doi.org/10.5281/zenodo.438045

References

Learning is planning: Near bayes-optimal reinforcement learning via monte-carlo tree search (2011). AUAI Press, pp. 19–26.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47, pp. 235–256. DOI: 10.1023/a:1013689704352. URL: https://link.springer.com/article/10.1023/A:1013689704352.

BELLMAN, RICHARD (1957). "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6, pp. 679–684. URL: http://www.jstor.org/stable/24900506 (visited on 08/15/2022).

An empirical evaluation of thompson sampling (2011). Advances in Neural Information Processing Systems 24 (NIPS), Curran Associates.

The KL-UCB algorithm for bounded stochastic bandits and beyond (2011).

Gelly, Sylvain and David Silver (July 2011). "Monte-Carlo tree search and rapid action value estimation in computer Go". In: *Artificial Intelligence* 175, pp. 1856–1875. DOI: 10.1016/j.artint.2011.03.007. (Visited on 04/30/2019).

Granmo, Ole-Christoffer (Aug. 2010). "Solving two-armed Bernoulli bandit problems using a Bayesian learning automaton". In: *International Journal of Intelligent Computing and Cybernetics* 3, pp. 207–234.

Pereira, F et al., eds. (2012). *Efficient bayes-adaptive reinforcement learning using sample-based search*. Vol. 25. Curran Associates, Inc., pp. 1034–1042. URL: https://proceedings.neurips.cc/paper/2012/file/35051070e572e47d2c26c241ab88307f-Paper.pdf.

Katehakis, Michael N. and Arthur F. Veinott (May 1987). "The Multi-Armed Bandit Problem: Decomposition and Computation". In: *Mathematics of Operations Research* 12, pp. 262–268. DOI: 10.1287/moor.12.2.262. (Visited on 12/09/2020).

The epoch-greedy algorithm for contextual multi-armed bandits (2007). Vol. 20. Advances in Neural Information Processing Systems. Curran Associates Inc., pp. 817–824.

Li, Lihong et al. (2010). "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th international conference on World wide web - WWW '10* 661-670. DOI: 10.1145/1772690.1772758.

A bayesian framework for reinforcement learning (2000). Proceedings of the Seventeenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc., pp. 943–950.

Mark, Yngvi Björnsson, and Jahn-Takeshi Saito (2010). "Monte carlo tree search in lines of action". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2, pp. 239–250.

Mnih, Volodymyr et al. (2013). "Playing atari with deep reinforcement learning". In: *ArXiv* abs/1312.5602.

Russell, Stuart and Peter Norvig (2010). *Artificial intelligence : a modern approach*. 3rd ed. Pearson, pp. 694–695.

Russo, Daniel and Benjamin Van Roy (Nov. 2014). "Learning to Optimize via Posterior Sampling". In: *Mathematics of Operations Research* 39, pp. 1221–1243. DOI: 10.1287/moor.2014.0650. (Visited on 02/24/2022).

Russo, Daniel J. et al. (2018). "A Tutorial on Thompson Sampling". In: *Foundations and Trends® in Machine Learning* 11, pp. 1–96. DOI: 10.1561/2200000070.

Schrittwieser, Julian et al. (Dec. 2020). "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 588, pp. 604–609. DOI: 10.1038/s41586-020-03051-4.

Scott, Steven L. (Nov. 2010). "A modern Bayesian look at the multi-armed bandit". In: *Applied Stochastic Models in Business and Industry* 26, pp. 639–658. DOI: 10.1002/asmb.874. (Visited on 07/16/2020).

Deisenroth, Marc Peter, Csaba Szepesvári, and Jan Peters, eds. (2013). *Evaluation and analysis of the performance of the EXP3 algorithm in stochastic environments*. Vol. 24.

PMLR, pp. 103–116. URL: https://proceedings.mlr.press/v24/seldin12a.html.

Silver, David, Aja Huang, et al. (Jan. 2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–489. DOI: 10.1038/nature16961.

Silver, David, Thomas Hubert, et al. (Dec. 2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362, pp. 1140–1144. DOI: 10.1126/science.aar6404.

Silver, David, Julian Schrittwieser, et al. (Oct. 2017). "Mastering the game of Go without human knowledge". In: *Nature* 550, pp. 354–359. DOI: 10.1038/nature24270. URL: https://www.nature.com/articles/nature24270.

Lafferty, J et al., eds. (2010). *Monte-carlo planning in large POMDPs*. Vol. 23. Advances in Neural Information Processing Systems. Curran Associates, Inc.

Sutton, Richard S and Andrew Barto (2018). *Reinforcement learning : an introduction*. The Mit Press.

Thompson, William R. (Dec. 1933). "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25, p. 285. DOI: 10.2307/2332286. (Visited on 12/10/2021).

– (Apr. 1935). "On the Theory of Apportionment". In: *American Journal of Mathematics* 57, p. 450. DOI: 10.2307/2371219.

Werner Duvaud, Aurèle Hainaut (2019). *MuZero General: Open Reimplementation of MuZero*. https://github.com/werner-duvaud/muzero-general.

Online planning for ad hoc autonomous agent teams (2011), pp. 439–445.