Bayesian Reinforcement Learning

Aman Taxali
ataxali@umich.edu

Yung-Chun Lee
yungclee@umich.edu

Abstract

In this report, we explain a general Bayesian strategy for approximating optimal actions in Partially Observable Markov Decision Processes, known as sparse sampling. Our experimental results confirm the greedy-optimal behavior of this methodology. We also explore ways of augmenting the sparse sampling algorithm by introducing additional exploration conditions. Our experimental results show that this approach yields a more robust model.

1 Introduction

In recent years, the successes of DeepMind's DQN, AlphaGo and AlphaZero have made reinforcement Learning a popular subject of discussion. Surprisingly, incorporating Bayesian techniques into Reinforced Learning has not been wildly researched. In this report, we will attempt to use Bayesian methods such as sparse sampling, Gaussian Processes and Thompson Sampling to solve the underlying problem of Reinforcement Learning.

2 Methodology

The general reinforcement learning problem tries to answer the question of how to behave optimally in an unknown *Markov decision process*.

2.1 Markov Decision Process

The Markov Decision Process (MDP) is a mathematical framework for sequential decision-making in systems where the outcome is partly under the control of an external decision maker and partly random. This idea was first proposed by Bellman in 1957, and has since been applied in many domains like robotics, economics and manufacturing.

An MDP is an extension of Markov chains known as discrete time stochastic control process. At each time-step, the process is in some state and a decision maker is required to choose an action. Based on the action, the process moves to a new state and returns to the decision maker the reward corresponding to the movement. Now, a new time-step begins, and based on the process these steps are either repeated until a terminal state is reached or indefinitely.

Model 1 (Markov Decision Process) Define an MDP $\mathcal{M}$ to be a 5-tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t results in transition to state s' at time $t + 1$
- $R_a(s, s')$ is the reward received after transitioning from state s to s' as a result of action a
- $\gamma \in [0, 1]$ is a factor that determines the discount rate of future rewards.

For an MDP, $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, an action policy π is a function that maps each state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$, defined as $\pi : \mathcal{S} \to \mathcal{A}$. The main problem of MDPs is to find the policy π such that the cumulative reward are maximized over a potentially infinite horizon.

For a given policy π, we define the value of a state s, denoted $V^\pi(s)$, as the expected sum of discounted rewards when the agent starts in the state s and follows policy π thereafter, for an infinite horizon.

$$V^\pi(s) = \boldsymbol{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \middle| s_0 = s \right] \quad \text{where } \boldsymbol{E}_\pi \text{ is the expected value under policy } \pi \tag{1}$$

A similar state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can be defined as the expected sum of discounted rewards when the agent starts in state s

and takes action a, and follows policy π thereafter.

$$Q^{\pi}(s,a) = \boldsymbol{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}R_{a_t}(s_t,s_{t+1})\middle| s_0=s, a_t=a\right] \tag{2}$$

The goal for solving an MDP is to find the optimal action policy π^* such that value function in equation (1) is maximized for all states $s \in \mathcal{S}$. So, the optimal policy is such that $V^*(s) \geq V^{\pi}(s)$ for all $s \in \mathcal{S}$ and all policies π.

Using recursive properties, equation (1) can be written as what is known as the *Bellman Equation* (Bellman, 1957):

$$\begin{aligned} V^{\pi}(s) &= \boldsymbol{E}_{\pi}\left[R_a(s_0,s_1)+\gamma V^{\pi}(s_{t+1})\middle| s_0=s\right] \\ &= \sum_{s'} P_{\pi(s)}(s,s')\left(R_a(s,s')+\gamma V^{\pi}(s')\right) \end{aligned} \tag{3}$$

It can be proven that the solution for the optimal policy V^*, called the *Bellman Optimality Equation*, satisfies:

$$V^*(s) = \max_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}} P_a(s,s')\left(R_a(s,s')+\gamma V^*(s')\right) \tag{4}$$

Bellman Optimality Equation (4) states that value of a state under an optimal policy equals the expected discounted reward for the best action in that state. This means the *greedy* optimal action given the optimal state value function V^* is:

$$\pi^*(s) = \arg\max_{a}\sum_{s'\in\mathcal{S}} P_a(s,s')\left(R_a(s,s')+\gamma V^*(s')\right) \tag{5}$$

The greedy policy is important because it is the optimal policy with respect to V^*. The analogous optimal state-action value function is:

$$Q^*(s,a) = \sum_{s'} P_a(s,s')\left(R_a(s,s')+\gamma\max_{a'}Q^*(s',a')\right) \tag{6}$$

The Q-value functions (2), (6) are useful in cases when the P and R functions are unknown. Instead of forward reasoning, the Q-value functions compute the weighted summation over different alternatives.

The relationship between Q^* and V^* is:

$$V^*(s) = \max_{a} Q^*(s,a) \tag{7}$$

And equation (5) can be written as:

$$\pi^*(s) = \arg\max_{a} Q^*(s,a) \tag{8}$$

2.2 Solving MDPs: Model-based Methods

In situations where a (perfect) model of the MDP is available, and the state space is manageable in size, the optimal policy of the MDP can be solved iteratively using *dynamic* or *linear programming*. Two core model-based methods are *policy iteration* (Howard, 1960) and *value iteration* (Bellman, 1957). However, in this report we will focus on the alternative *model-free* methods.

2.3 Solving MDPs: Model-free Methods

Reinforcement learning (RL) attempts to solve the MDP problem in which an agent (or action taker) interacts with a dynamic and incompletely known environment; with the goal of finding an action policy that optimizes the rewards in the long-term. RL is a *model-free* method because it does not rely on prior knowledge of the MDP's transition and reward functions. RL also adds to the MDP problem the topic of incomplete information and approximation, and thus the need for sampling and exploration. This is especially useful when either the state space is too large, the system is only available as a simulator or no system model is available at all.

Within model-free RL, there are two choices for modeling the optimal action policy. The first learns the transition and reward functions from the interaction with the environment. Once the learned model of the environment is sufficiently correct, dynamic programming methods can be applied. This is called *indirect RL*. The second method attempts to directly estimate value for actions, without estimating the underlying model of the MDP. This is called *direct RL*. It is worth noting that mixed forms between these two methods also exist.

Model-free RL methods fall under the general class of algorithms that interact with an environment one time-step at a time, and update their estimates after each experience. This is called *online learning*, and Algorithm 1 provides a general template for it.

2.4 Exploration-Exploitation Trade-off

In online learning models the only feedback the agent receives after interacting with the environment are reward values. In order to find the

Algorithm 1: *Online Reinforcement Learning*

foreach *time-step* **do**
  $s \in \mathcal{S}$ is initialized as the starting state
  $t := 0$
  repeat
    choose an action $a \in \mathcal{A}(s)$
    perform action a
    observe the new state s' and received reward r
    update $\tilde{P}, \tilde{R}, \tilde{Q}$ and/or $\tilde{V}$ using the experience $\langle s, a, r, s' \rangle$
    $s := s'$
  until *end of training time*

optimal action policy, the agent must learn by trial-and-error. This involves balancing training time between exploration (learning more about the environment the agent is interacting with) and exploitation (using learned knowledge to take reward-maximizing actions). This simultaneous exploration-exploitation trade-off is a core problem that needs to be addressed in all online learning models.

2.5 Bayesian Reinforcement Learning

Bayesian approaches to reinforcement learning is a growing field of research due to recent successes of Bayesian modeling in areas such as machine learning. Bayesian models are ideally suited to reinforcement learning as they offer an explicit representation of uncertainty and the concept of approximating the optimal action policy via the Bayes optimal decision. Bayesian inference is also able to incorporate streaming information, as required by online learning.

The Bayesian approach to solving the MDP is equivalent to solving the *Partially Observed* MDP.

2.6 Partially Observable Markov Decision Process

The Partially Observable Markov Decision Process (POMDP) is an extension of the MDP where the underlying state of MDP is not completely observable by the agent. Instead, the agent must maintain a probability distribution over the set of possible states based on the observations.

Model 2 (Partially Observable Markov Decision Process) Define an POMDP $\mathcal{M}$ to be a 7-tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \gamma \rangle$ where

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t results in transition to state s' at time $t + 1$
- $R_a(s, s')$ is the reward received after transitioning from state s to s' as a result of action a
- $\mathcal{O}$ is the set of observations
- $\Omega(o|s, a)$ is the probability distribution over possible observations $o \in \mathcal{O}$, conditioned on action a taken at state s
- $\gamma \in [0, 1]$ is a discount factor that determines the exponential devaluation rate of future rewards.

Since the state of the process is not directly observable by the agent in an POMDP, the agent must rely on the recent history of actions and observations $\{a_0, o_0, ..., a_t, o_t\}$ to construct a probabilistic distribution over states. This *belief-state* b_t is defined over the state probability simplex, b

$$b_{t+1}(s') \propto \Omega(o_{t+1}|s, a_t) \int_{\mathcal{S}} P_{a_t}(s, s') b_t(s) ds \tag{9}$$

Solving for the optimal policy π^*, that maximizes the sum of discounted expected future rewards for all belief states, is defined using a variant of the Bellman equation:

$$\mathrm{V}^*(s) = \max_{a \in \mathcal{A}} \left[\int_{\mathcal{S}} R_a(s, s') b_t(s) ds + \gamma \int_{\mathcal{O}} Pr(o|b_t, a) V^*(b_{t+1}) do \right] \tag{10}$$

In the Bayesian methodologies for solving the Partially Observed Markov Decision Process (POMDP), generally Ω, R_a are modeled using Bayesian methods. This means that the greedy optimal Bayesian policy maximizes equation (10) for each state.

2.7 Bayesian Sparse Sampling

The *sparse sampling* method (Kearns et al, 2001) provides a general strategy for approximating greedy optimal actions in POMDPs (and MDPs). Algorithm (2) provides an outline of the sparse sampling method (for practical reasons, a finite look-ahead horizon is considered).

The sparse sampling method applies a generative model using the Bayesian probabilistic approximations for Ω, R_a. A look-ahead decision tree is constructed by simulating the belief-based states of the MDP, which are model internally, and the optimal action is computed by dynamic

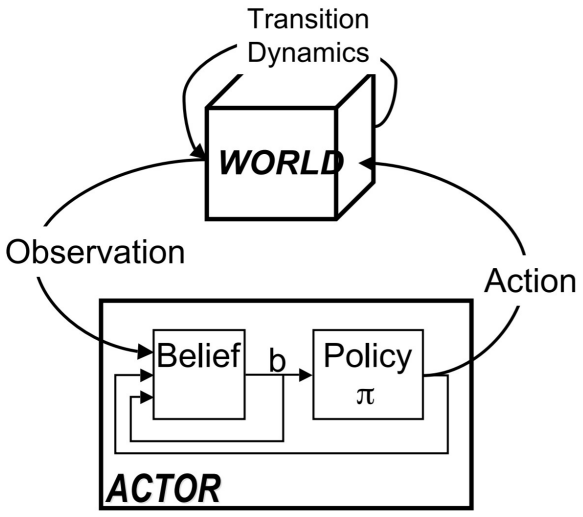

Figure 1: Partially Observable Markov Decision Process. *Image credit: Stefan Kopp, https://www.techfak.uni-bielefeld.de*

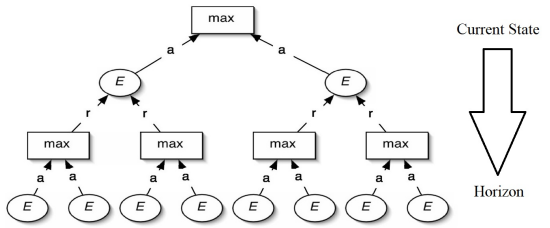

Figure 2: Visualization of a look-ahead tree, with outcome (expectation) nodes and decision (maximization) nodes. *Image credit: Wang et al., Bayesian Sparse Sampling for Online Reward Optimization, 2005*

programming (estimating rewards and optimal actions at the horizon, and then progressively backup to each earlier belief-state). Kearns et al. showed that this method approximates the greedy Bayes optimal action, given an POMDP and a set of belief-states, for a sufficiently large horizon and branch factor.

2.7.1 Experimental Results

To implement a Bayesian sparse sampling methodology we try to solve an MDP constructed as a game. As seen in figure 3, in this game, the agent (mouse) is in a maze. Also in the maze are terminal loss states (cats), invalid states (walls) and a terminal win state (goal). The terminal loss states are not static, the cats move in predetermined periodic paths (as seen in figure 4). Further, the agent is unaware of the number of cats in the maze, and the cats are indistinguishable from one another. It can be seen that from the agent's perspective, this is a POMDP.

Algorithm 2: *Sparse Sampling*

```
GrowSparseTree (node, branchfactor, horizon):
    if node.depth = horizon then
        return
    if node.type = 'decision' then
        foreach a ∈ A do
            child = ("outcome", depth, node,
              belief_state, a)
            GrowSparseTree(child,
              branchfactor, horizon)
    if node.type = 'outcome' then
        for i in 1,...,branchfactor do
            [reward, obs] =
              sample_obs(node.belief_state,
              node.action)
            updated_posterior =
              posterior(node.belief_state, obs)
            child = ("decision", depth+1,
              updated_posterior, [reward,obs])
            GrowSparseTree(child,
              branchfactor, horizon)
EvaluateSubTree (node, horizon):
    if node.children = empty then
        mev = MaxExpected-
          Value(node.belief_state)
        return mev * (horizon - node.depth)
    if node.type= 'decision' then
        return
          max(EvaluateSubTree(node.children))
    if node.type= 'outcome' then
        values =
          EvaluateSubTree(node.children)
          return avg(node.rewards + values)
```

To model this problem as a POMDP, the following steps were taken:

- Let $\mathcal{S}$ be all permutations of agents, cats, walls and goal in the maze
- $\mathcal{A}$ is the set of actions available to the agent, $\{$up, down, left, right$\}$, at every state $s \in \mathcal{S}$
- The reward function is trivial, so it does not need to be modelled probabilistically. The agent receives a reward of -10 for terminal loss, 10 for terminal win and -0.1 for any other move.

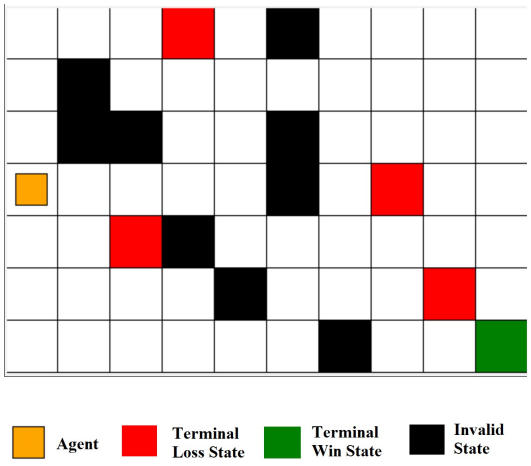

Figure 3: Cat-Mouse game

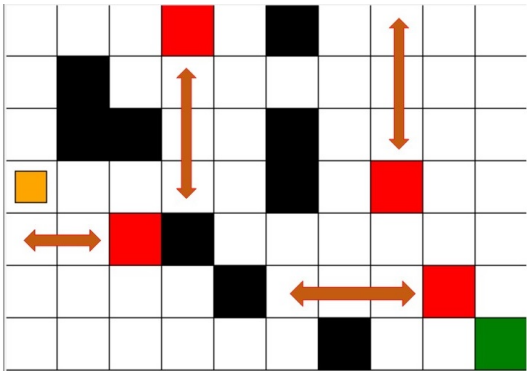

Figure 4: Cat-Mouse game, indicated the paths of movement of cats

- $P_a(s, s')$ is also trivial because the agents position after any action at any state is deterministic in this game.
- The probabilistic belief-states b_t, which are based on the agent's observation history, model the agent's belief of the position of the terminal loss states (cats). This is explained in more details below.

The horizontal and vertical positions of the terminal loss belief-states b_t are modelled using two sets of Gaussian Processes with periodic Exp-Sine-Squared kernel.

- Let $X = \{(x_1, t_1), ..., (x_n, t_n)\}$, where x_i is the horizontal position where the agent observes a terminal loss state and t_i is the number of moves since the start of the game. It is important to note that t_i are periodic because once the agent reaches any terminal state, the game resets and training resumes.
- In our application, due to the presence of more than one terminal loss state in the maze and the periodic nature of t_i, more than one unique x_i value can be observed at any time t_i. Fitting a Gaussian Process to such data does not result in a good fit of observations (large sum of squared errors). To resolve this, the values in X are first clustered using the Manhattan Distance Nearest-Neighbor classification of x_i's; and a Gaussian Process is then fitted to each of these clusters.
- The two steps above are repeated for $Y = \{(y_1, t_1), ..., (y_n, t_n)\}$, where y_i is the vertical position where the agent observes a terminal state at t_i.

These Gaussian Processes are also referred to as the internal belief models of our sparse sampling approach.

To construct the look-ahead tree, the belief positions of the terminal states are estimated for each time-step by sampling each of the Gaussian Processes constructed above. In our experiments, the 95% confidence interval was also computed using the sample variances, as reported by the Exp-Sine-Squared kernel. The number of terminal loss belief-states included per time-step in the look-ahead tree is controlled by the branching factor.

Figure 7 shows the scores of 50 training runs of the Bayesian sparse sampling algorithm on our game, over 2000 training time-steps. The solid line indicates the run that achieved the highest final score, and the shaded bands show the 95% quantile interval of the scores at each time-step. The greedy-optimal nature of this method appears to be confirmed by the wide, growing confidence bands. We can observe in Table 1 that while the maximum score increases with training time, so does the variance in scores. The last column of Table 1 reports the percentage of runs that had scores greater than zero (winning) at each time-step. The maximum scores and percentage of winning-runs take a convex shape over training time.

2.8 Sparse Sampling Method with Thompson Sampling

Although sparse sampling can provide an optimal greedy decision strategy, it is not a sophisticated action value estimator. This is represented by the fact that the sparse sampling algorithm employs a balanced look-ahead tree. An alternative to this uniform action value estimation is to propose a look-ahead tree that expands intelligently.

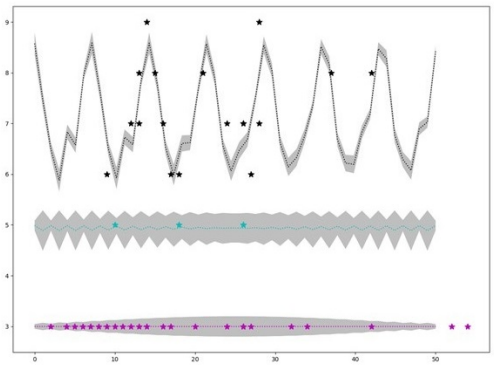

Figure 5: Loss-state belief horizontal positions, 95% C.I. (per fitted GP) in grey

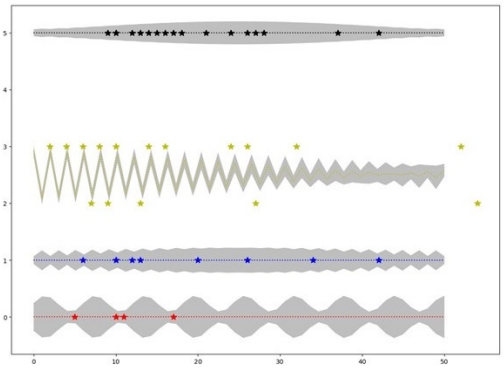

Figure 6: Loss-state belief vertical positions, 95% C.I. (per fitted GP) in grey

2.8.1 Thompson Sampling

Thompson Sampling algorithm (Thompson, 1933) is a heuristic for choosing actions inspired by the multi-arm bandit problem. In this method, at each time step t, we sample a parameter $\hat{\theta}$ from a posterior P_θ and select the optimal action conditional on $\hat{\theta}$. Finally, the observation resulting from the action is used to update the posterior P_θ.

2.8.2 Look-ahead Tree with Thompson Sampling

We propose the idea of augmenting the look-ahead tree from Bayesian sparse sampling with Thompson Sampling. The *greediness* of the sparse sampling method can be reduced by forcing the look-ahead tree to only evaluate a subset of actions, $A^* \subset \mathcal{A}$, at the root node. In such a scenario, the sparse sampling method will be forced to choose the closest approximation to the belief-state Bayes optimal action from A^* (which may not be the same action as if the complete action set had been evaluated). Recall the exploration-exploitation tradeoff; this scheme is analogous to forcing the sparse sampling method to explore more.

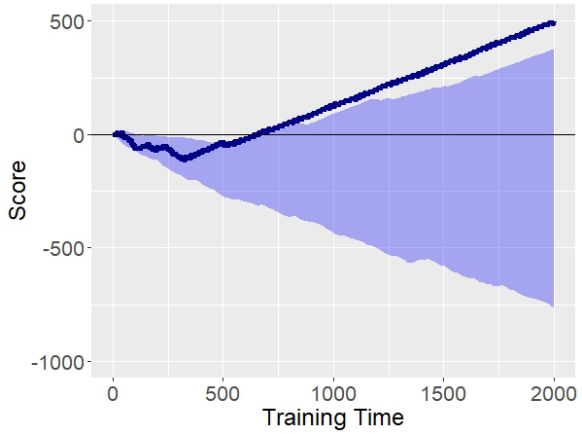

Figure 7: Training performance of sparse sampling method

Time	Min	Mean	Max	St.Dev	Pct. Win
100	-99.1	-32.39	10.4	26.01	13.33
500	-355.1	-120.17	-7.0	81.44	0.00
1000	-503.5	-151.87	187.0	148.60	16.67
1500	-830.1	-182.81	307.5	246.77	20.00
2000	-1166.8	-218.76	499.9	355.88	20.00

Table 1: Training scores of sparse sampling method

As a simple model, we propose sub-sampling the action set as a function of training time. That is, we would like the look-ahead tree to grow fewer branches early in the training time and progressively approach a balanced tree as training time increases. Another way to explain this is that we force additional exploration on the sparse sampling method early in the training, and reduce this condition as training time increases.

For a given POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \gamma \rangle$, the elements of our simple training-time based Thompson Sampling algorithm are:

- hyper-parameter ρ which determines *how quickly* the additional exploration condition is reduced from the sparse sampling method
- parameter θ with determines the *amount* of additional exploration imposed on the sparse sampling algorithm, at each time-step
- the prior distribution $P(\theta)$
- the posterior distribution $P(\theta|\mathcal{O}) \propto P(\mathcal{O}|\theta)P(\theta)$, where $\mathcal{O}$ is the set of past observations

2.8.3 Experimental Results

Using the convenience of the conjugate prior properties of the Beta distribution, we implement a

simple Thompson Sampling extension to Bayesian sparse planning.

We let

- $P(\mathcal{O}|\theta)$ follow a Bernoulli distribution, where $P_{\mathcal{O}}(o) = 1$ for any observation o during training
- the prior $P(\theta)$ follow the Beta distribution with parameters (1, ρ)
- then, we know that the posterior $P(\mathcal{O}|\theta)P(\theta)$ is a Beta distribution with parameters $\left(1 + \sum_{i=1}^{n} x_i, \rho\right) = \left(1 + n, \rho\right)$, where n is the length of training time

Thus, at each training step, the Thompson Sampling parameter $\hat{\theta}$ can be estimated from the posterior $\theta|\mathcal{O} \sim Beta(1 + n, \rho)$; and the posterior $P(\theta|\mathcal{O})$ is updated at the end of the training step using the new observation.

$\hat{\theta}$ tells us the *amount* of additional exploration that must be imposed on the sparse sampling algorithm at the current time-step. Recall that the complete action set $\mathcal{A}$ for our MDP game is {up, down, left, right} for all states $s \in \mathcal{S}$. In our experimental analysis, we defined the conditions:

$$\begin{cases} \text{if } \hat{\theta} \leq 0.33, & \text{reduce action set to size 2} \\ \text{if } 0.33 < \hat{\theta} \leq 0.66, & \text{reduce action set to size 3} \\ \text{if } \hat{\theta} > 0.66, & \text{leave action set untouched} \end{cases}$$

Once we have decided the size of the action subset to use in the sparse sampling algorithm, the final step is to choose which actions from the complete set are included in this subset A^*. Again, we implement a basic Bayesian strategy here.

We let the posterior $P(a|\theta, \mathcal{O})$ for each $a \in \mathcal{A}$ follow a Dirichlet distribution. Using the properties of conjugate priors and pseudo-counts, we define the parameters of the posterior $P(a|\theta, \mathcal{O})$ to $\alpha_i = 1 + \sum_{j=1}^{n} \gamma x_j$, where $x_j = 1$ if the agent took action a_i at time-step t_j (0 otherwise), and $\gamma \in [0, 1]$ is a discounting factor. This posterior scheme deprioritizes backtracking (eg: moving down after moving up).

Figure 8 shows the scores of 50 training runs of the sparse sampling algorithm with Thompson Sampling in our MDP game, over 2000 training time-steps. Predictably, the additional exploration comes at the cost of score maximization. This new approach achieves much lower maximum scores compared to the balanced sparse sampling method. However, we also notice lower variance in scores with the introduction of Thompson Sampling (see Table 2). The confidence intervals in Figure 8 also appear to suggest a much tighter band in score quantiles during time-steps when added exploration is high, conversely the confidence intervals diverge as the models behaves more *greedily*. The wide, growing confidence intervals we observe during the later stages of training in this model also match our experimental observations with the sparse sampling method earlier.

However, it is reasonable to claim that with the additional exploration conducted with the introduction of Thompson Sampling, the internal belief-based models of this new agent may be more accurate than in the greedy sparse sampling case. In order to highlight this potential difference, we propose a test environment for our two optimal action selection models.

The test environment is the same MDP game (Cat-Mouse) as in training, except now the position of the goal (terminal win state) is determined randomly at the start of every game. The motivation behind this is to test the robustness of the internal belief based models in the two methods. After training for 2000 time-steps, as seen before, the 50 models for both methods were asked to interact with the test environment for 200 moves (Thompson Sampling was disabled during testing). Tables 3 and 4 report the performances of sparse sampling and sparse sampling with Thompson Sampling methods, respectively.

The most notable results from the robustness testing are that the average scores of the method with Thompson Sampling were higher, while the variance was lower. This appears to confirm our initial claim of the presence of more robust belief based models with more exploration. However, the impact on maximum scores and percentage of winning runs is marginal. This seems to suggest that the simple action value estimation methodology proposed by us can be improved.

2.9 Sparse Sampling with Episodic Reset and Bootstrapping

Extensive research has recently been published on the topic of *Neural Episodic Control* (Pritzel et al., 2017), building on the idea of *Deep Q-Network agents* (Mnih et al., 2015).

These ideas introduced two fundamental concepts to deep neural network reinforcement learning: *episodic training* and *memory replay*. In this section, we will try to augment our models with

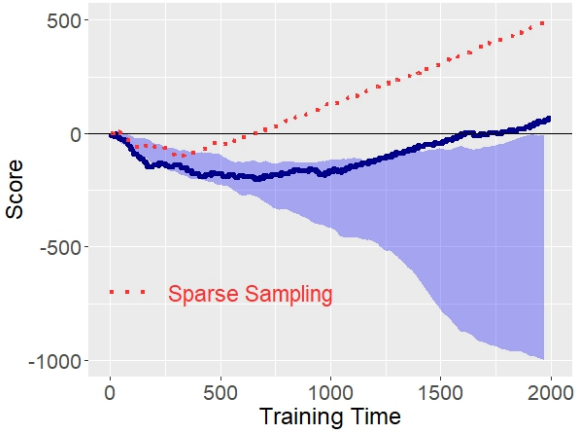

Figure 8: Training performance of sparse sampling with Thompson Sampling

Time	Min	Mean	Max	St.Dev	Pct. Win
100	-89.2	-49.75	-9.4	20.19	0.00
500	-256.5	-166.60	-67.6	43.22	0.00
1000	-454.6	-246.16	-65.3	98.10	0.00
1500	-870.9	-300.92	-44.0	212.01	0.00
2000	-1285.0	-397.60	67.8	335.25	3.33

Table 2: Training scores of sparse sampling with Thompson Sampling

Time	Min	Mean	Max	St.Dev	Pct. Win
50	-14.9	1.80	25.3	11.89	43.33
100	-69.4	2.64	40.5	26.50	53.33
150	-64.1	5.52	65.8	38.23	53.33
200	-78.8	9.03	81.0	49.20	56.67

Table 3: Testing scores of sparse sampling

Time	Min	Mean	Max	St.Dev	Pct. Win
50	-14.9	3.99	15.2	8.99	62.96
100	-39.7	2.87	40.5	23.22	59.26
150	-54.6	8.77	65.8	33.10	55.56
200	-59.6	12.06	81.0	42.89	55.56

Table 4: Testing scores of sparse sampling with Thompson Sampling

analogies of these ideas.

The concept of *episodic training* simply highlights the fact that most reinforcement learning problems can be broken down into a series of episodes of interactions between an agent and a system. The exact definition of an episode depends on problem and the MDP, but the end of a training episode is usually defined by the time-step when the agent reaches a terminal state. In the next time-step, the MDP resets, the agent is placed back to an initial state, and a new training episode begins. In the context of our Cat-Mouse game, one training episode is one game. We can use the concept of episodic training to reset the Thompson Sampling posteriors at the start of every new game. This means that instead of introducing additional exploration as a function of total training time, we now introduce additional exploration as a function of game length (number of actions since start of game, before reaching terminal state).

Memory replay in Deep Q-Networks stores all observations during training in a replay memory. At each training step, samples from the replay memory are re-used as 'new' observations, alongside the actual observations resulting from the agent's interaction with the system. This technique is successful in DQNs because it breaks the similarity of subsequent training data, which might otherwise drive the network into a local minimum. In our Bayesian optimal action approximation methods, an analogy of memory replay would be bootstrapping. At each training step, we can perform a simple random sample of the observations from the agent's training history, and re-use them to fit the internal belief based models.

Figure 9 and Table 5 show the training results of the sparse sampling method with our suggested analogies of episodic training and bootstrapping. It is immediately clear that, compared to the two previous model, the variance in training scores is significantly smaller, and the minimum scores are higher. However, the mean and maximum training scores severely under-perform the two previous models. This model appears to display very risk-averse characteristics.

This proposed method's performance in the testing environment also appears to confirm the risk-averse behaviour. As seen in training, the variance in scores is low, but the mean and maximum scores under-perform the models seen previously. Interestingly, Table 7 shows that a far

majority of the testing runs for the sparse sampling algorithm with episodic training and bootstrapping never actually finished a game (neither a win nor loss terminal states were ever reached by the agent). The next most popular outcome of testing was a winning final score (more wins than losses).

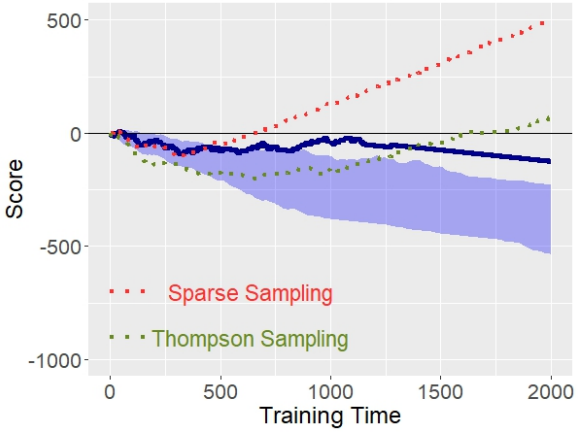

Figure 9: Training performance of sparse sampling with Episodic Reset and Bootstrapping

Time	Min	Mean	Max	St.Dev	Pct. Win
100	-79.3	-24.81	20.5	23.07	16.67
500	-237.7	-144.98	-37.5	47.78	0.00
1000	-455.4	-234.95	-35.2	90.68	0.00
1500	-494.3	-312.29	-74.3	100.83	0.00
2000	-652.4	-386.22	-124.3	117.52	0.00

Table 5: Training scores of sparse sampling Episodic Reset and Bootstrapping

Time	Min	Mean	Max	St.Dev	Pct. Win
50	-14.9	-4.14	15.2	6.72	8.33
100	-19.9	-9.95	0.1	4.27	8.33
150	-34.8	-14.02	15.7	13.84	16.67
200	-39.8	-17.31	30.9	16.75	8.33

Table 6: Testing scores of sparse sampling Episodic Reset and Bootstrapping

# of runs	Never Finished	Finished Win	Finished Lose
50	34	13	3

Table 7: Testing results count summary of sparse sampling Episodic Reset and Bootstrapping

3 Discussion

In this report we presented Bayesian interpretations to the Reinforcement Learning problem. The central model discussed was the Bayesian sparse sampling algorithm. Experimental results confirmed this methods ability to approximate greedy optimal actions in POMDPs. Next, the Thompson Sampling algorithm was employed to reduce the greediness of the sparse sampling algorithm. Using a very simple (nearly random) action selection strategy, we were able to report some improvements in the quality of the internal belief models compared to the vanilla sparse sampling method. Lastly, we attempted to create Bayesian analogies to ideas of episodic reset and memory replay from Deep Q-Networks. The result was an extremely risk-averse agent, that could not outperform the previous two models in score.

There are many potentials for improvement to the proposals we have presented in this report. Firstly, the Thompson Sampling method can be designed to grow the unbalanced look-ahead tree more intelligently. Thompson Sampling can give preference to *locally promising* actions by including rewards in the priors/posteriors. In the model with bootstrapping, a better scheme than simple random sampling (of past observations) can be implemented. DeepMind's research has shown that the quality of observations chosen for replay has a significant impact on the model's score performance.

References

Howard, Walter E. 1960. *Innate and environmental dispersal of individual vertebrates*, pages 152–161 *American Midland Naturalist*

Richard Bellman. 1957. *A Markovian Decision Process*, volume 6, pages 679–684 *Indiana University Mathematics Journal*

Martijn van Otterlo. 2009. *Markov Decision Processes: Concepts and Algorithms SIKS course on Learning and Reasoning*

Martin L. Puterman. 1957. *Markov Decision Processes. Wiley Interscience*

Wang, Tao and Lizotte, Daniel and Bowling, Michael and Schuurmans, Dale 2005. *Bayesian sparse sampling for on-line reward optimization Proceedings of the 22nd international conference on Machine learning*

Michael Kearns, Yishay Mansour, and Andrew Y, NG. 2002. *A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes*, 49, 193-208 *Machine Learning*

Richard Dearden, Nir Friedman, and David Andre. 2002. *Model-Based Bayesian Exploration. Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*

Pritzel, Alexander and Uria, Benigno and Srinivasan, Sriram and Puigdomenech, Adria and Vinyals, Oriol and Hassabis, Demis and Wierstra, Daan and Blundell, Charles 2017. *Neural episodic control. arXiv preprint arXiv:1703.01988*

Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Rusu, Andrei A and Veness, Joel and Bellemare, Marc G and Graves, Alex and Riedmiller, Martin and Fidjeland, Andreas K and Ostrovski, Georg and others 2017. *Human-level control through deep reinforcement learning. Nature,Volume 518*