Playing Atari with Bayesian Deep Expected Sarsa

Ebubekir Karamustafa
University of Southern Denmark
ebkar24@student.sdu.dk

Abstract

This project investigates an on-policy algorithm, Deep Expected SARSA, for playing Atari games. Inspired by the success of Deep Q-Learning, the project aims to train an agent to play Atari games as well as address challenges of Q-Learning such as instability and overestimation. The algorithm integrates Bayesian layers into the neural network architecture to enhance decision-making in stochastic environments. A detailed description of the algorithm is provided, including the update rules and network design. Moreover, the pseudo-code and detailed experiments that compare our algorithm with baseline methods are presented. While Deep Expected SARSA shows promising performance, further improvements and experimentation are required to enhance its performance and stability.

1 Introduction

Drawing inspiration from the "Playing Atari with Deep Reinforcement Learning"(1) paper, this project proposes an on-policy algorithm that aims to refine and improve the learning strategy for Atari games.

While Deep Q-Learning has shown impressive performance in Atari games, it faces several challenges such as instability and overestimation. Off-policy methods like Q-learning update based on maximized future rewards, which can result in divergence or oscillations during learning. This lack of stability is exacerbated in complex environments, potentially leading to divergence, slower convergence, or convergence to suboptimal policies. In contrast, on-policy methods like SARSA offer a more stable alternative. By directly incorporating the current policy's actions into the learning process, SARSA mitigates the risks of overestimation and divergence. In fact, SARSA has been tested for playing Atari games. While it demonstrated slower convergence compared to Q-Learning, it achieved better performance with less possibility of divergence in Breakout and Seaquest enviroments. (2).

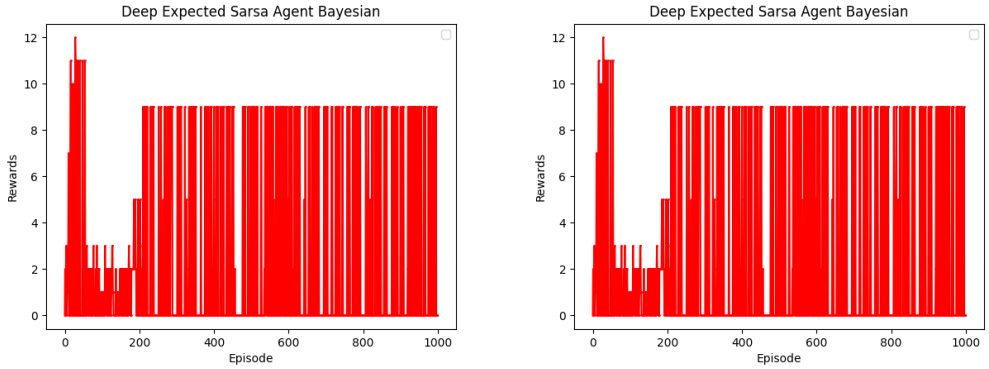
2 Algorithm

Expected SARSA provides further performance improvement for our case. By considering the expected value of future rewards under the current policy, Expected SARSA provides smoother learning and less variance. As such, we integrate the following update rule into our model's algorithm:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \sum_{a'} \pi(a'|s')Q(s',a') - Q(s,a)]$$

For the expectation term, we take the weighted sum of the outputs of all actions using the following formula:

$$\sum_{a'} \pi(a'|s')Q(s',a') = Q(s',a') \cdot (1-\epsilon_t) + \sum_{a} \frac{\epsilon_t \cdot \mathrm{Q}(s',a)}{n}$$



(a) Deep SARSA

(b) Deep Expected SARSA

Figure 1: On-policy methods with bayesian layers and local reparameterization trick

ϵ_t = epsilon threshold
a' = Action that maximizes the next state s'

Moreover, we use ϵ-greedy with decaying factor when the agent interacts with the environment.

$$\epsilon_t = \epsilon_n + (\epsilon_s - \epsilon_n) \cdot e^{-\frac{\alpha}{\lambda}} \tag{1}$$

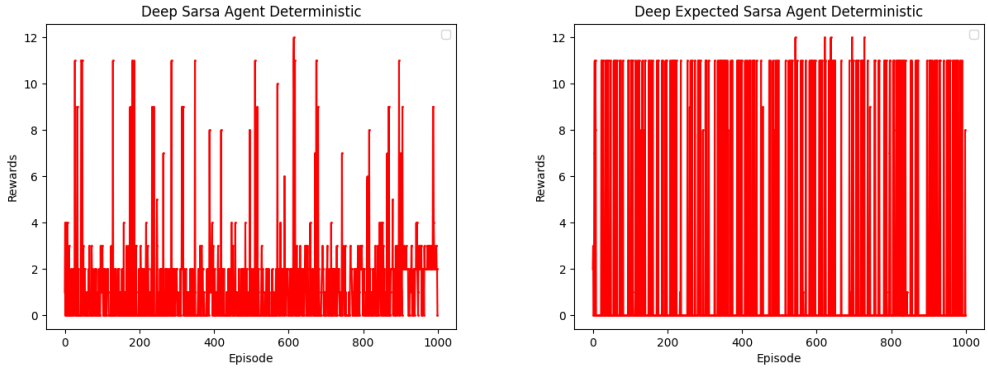
ϵ_s = tunable epsilon start value
ϵ_e = tunable epsilon end value
λ = tunable decaying factor
α = number of steps

We utilize a replay buffer to train the Deep Expected SARSA agent, with a tunable number of samples (κ) collected from the environment. Initially, our algorithm is configured to train the Deep Expected SARSA agent with 128 samples from the environment.

2.1 Neural Layer and Network Design

The neural network architecture used in this project closely mirrors that outlined in the paper "Playing Atari with Deep Reinforcement Learning", chosen for its demonstrated efficient performance. This design employs a sequence of convolutional layers to process preprocessed environment state data, which consists of 84x84 images with 4 channels. The initial layer consists of 16 filters of size 8x8 with a stride of 4, followed by a ReLU activation function. Subsequently, the second layer with 32 filters of size 4x4 and stride 2, also followed by ReLU, is applied, followed by a third layer with 64 filters of size 3x3 and stride 1, again with ReLU activation. The resulting output is flattened and passed through fully connected linear layers: the first containing 512 ReLU units, and the second mapping to the valid action space for the given environment. To ensure stable learning during the training, output normalization is performed by dividing 255. This neural network architecture has been proven to provide effective performance in Q-Learning applications (1). We use this structure to build a policy network for our SARSA and Expected SARSA agents.

Furthermore, Bayesian layers are incorporated into both linear and convolutional layers to benefit from the stochasticity provided by the Bayesian framework. We enable the agent to make more robust decisions, particularly in environments with high stochasticity such as Atari games. Additionally, the Bayesian framework quantifies the uncertainty in the neural layer which can be incorporated into the decision-making process to improve exploration strategies. While overfitting was not a concern in this project, the Bayesian structure may help mitigate overfitting by capturing and incorporating uncertainty. To implement the Bayesian structure within the neural network framework, we adopt the Bayes by Backprop algorithm (3). Moreover, the local reparametrization trick is employed to reduce variance and efficiently estimate gradients during training (4).



(a) Deep SARSA

(b) Deep Expected SARSA

Figure 2: On-policy methods with deterministic layers and local reparameterization trick

3 Pseudo code

Algorithm 1 Deep Expected SARSA

Initialize $Q_\theta(s, a)$ arbitrarily
Initialize finite set $D := \emptyset$, $\kappa = 128$
for each episode **do**
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy with decaying factor λ)
  for each step of episode **do**
    Take action A, observe R, S'
    $D = D \cup \{S, A, R, S'\}$
    if $|D| = \kappa$ **then**
      $v \leftarrow \sum_{a'} \pi(a'|S')Q_\theta(S', a')$ {Expected value over next actions by taking weighted sum}
      $Q_\theta(S, A) \leftarrow Q_\theta(S, A) + \alpha[R + \gamma v - Q_\theta(S, A)]$
      $D := \emptyset$
    end if
    $S \leftarrow S'$
  end for
end for=0

4 Preliminary Results and Analysis

We tested our algorithm in a gym Breakout environment with 4 frame skip. The parameters for this experiment were set as follows:

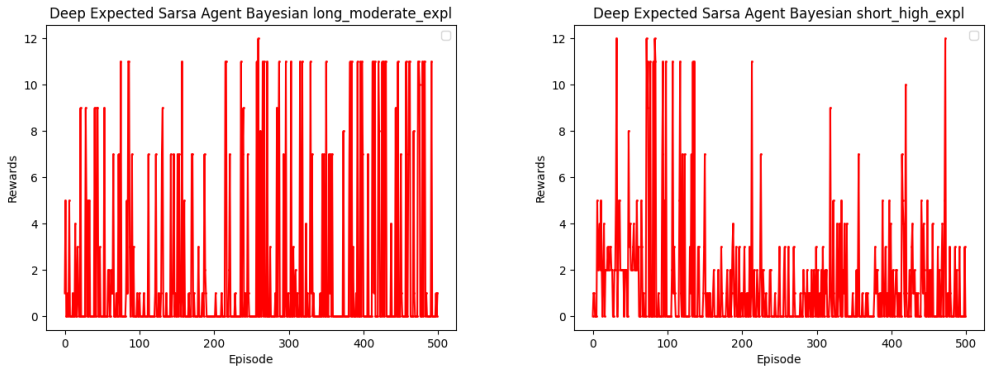
$$\epsilon_s := 0.99, \qquad \epsilon_e := 0.1 \qquad \lambda := 1000 \qquad \kappa := 128 \qquad \alpha := 10^{-4}$$

The experiments are conducted to evaluate the performance improvement using a Bayesian neural network instead of deterministic neural networks. In these experiments, layers employ the local reparameterization trick. Furthermore, we analyzed the performance difference between SARSA and Expected SARSA to determine if Expected SARSA offers enhancements.

As shown in Figure 1, Expected SARSA significantly improves the performance and reduces variance. As anticipated, Expected SARSA consistently achieves higher rewards after exploring a proficient policy/path from approximately episode 200. However, it is important to realize that the algorithm achieved even higher rewards in initial episodes but could not establish a proficient policy until approximately episode 200. Moreover, the Expected SARSA experiences intermittent drops which demonstrates the instability of the algorithm.

Table 1: Exploration Parameter Sets for Testing

Strategy	ϵ_e	ϵ_s	λ
Initial Exploration Parameters	0.9	0.05	1000
High and Long Exploration	0.9	0.1	1500
Moderate and Long Exploration	0.7	0.1	2000
Moderate and Short Exploration	0.7	01	500
High and Short Exploration	0.9	0.1	500



(a) Long And Moderate Exploration

(b) Short and High Exploration

Figure 3: Exploration Parameters Tests

Figure 2 demonstrates the performance comparison between SARSA and Expected SARSA when the neural network utilizes deterministic layers. Both algorithms exhibit enhanced performance by employing deterministic layers. This improvement is more obvious in Expected SARSA with higher average rewards compared to the algorithm with the Bayesian framework. However, it's important to realize that the Bayesian framework enabled the agent to achieve slightly higher rewards in the initial episodes compared to the deterministic framework. Therefore, figure 2 provides a valuable reference for subsequent observations into exploration strategies to find the optimal policy with the Bayesian framework.

5 Subsequent Experiments and Observations

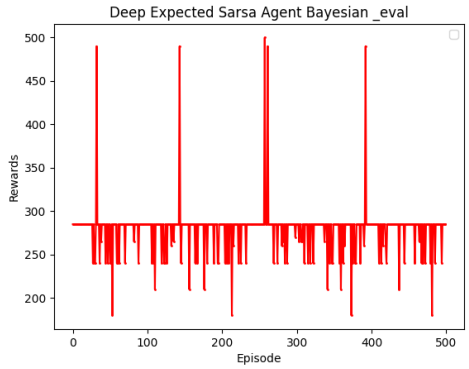
Based on the preliminary results and analysis, the following issues and aspects have been addressed for further performance improvement. Moreover, the experiments have been expanded to include more baseline models and to compare our model with these baselines in other environments such as Pong and Space Invaders.

5.1 Exploration Strategy

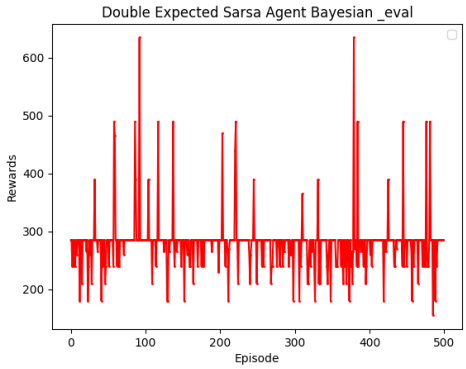
We proceed with ϵ greedy due to its proven efficiency in exploration. To determine the best exploration strategy, we tested our model with the parameters listed in Table 1.

Figure 3 [1] demonstrates informative patterns for exploration strategies. Notably, it is clear that the moderate exploration strategy with long duration demonstrates the most stable and best performance among the four parameter sets. When the model adopts a high exploration strategy with shorter duration, it can achieve higher rewards in the initial episodes compared to a moderate exploration strategy with a longer duration. However, this approach suffers from the same issue as our model did with the initial set of parameters: an inability to establish a policy that produce higher rewards in later episodes. Furthermore, the model tends to perform poorly if it employs a high exploration strategy or

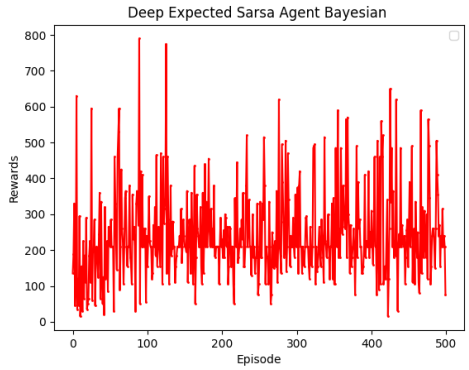
[1] To avoid redundancy, Figure 3 does not include the results for the moderate exploration strategy with short duration and the high exploration strategy with long duration due to their poor performance.



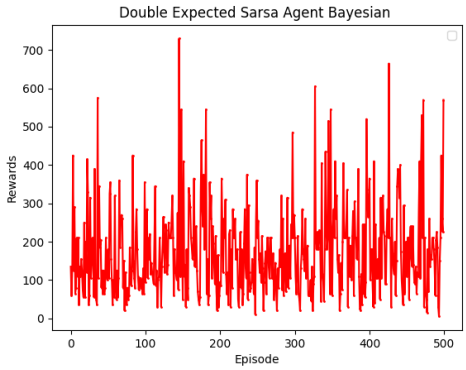
(a) Deep Expected SARSA Evaluation



(b) Double Deep Expected SARSA Evaluation



(c) Deep Expected SARSA Training



(d) Double Deep Expected SARSA Training

Figure 4: Investigating overestimation in SpaceInvaders environment

a strategy with a short duration, which indicates that the λ factor should decay slowly with moderate exploration.

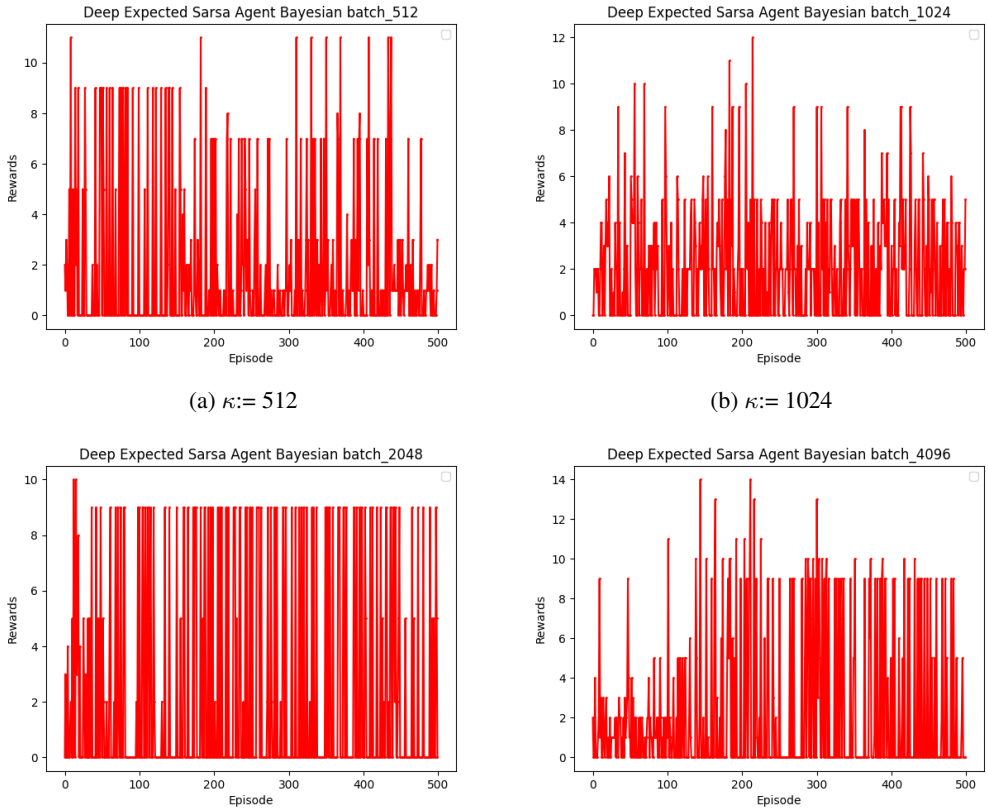
5.2 Overestimation

As mentioned, on-policy methods do not suffer from overestimation. Moreover, we further reduce the possibility of overestimation with a Bayesian framework. To test our hypothesis, we evaluated our model in the Space Invaders environment. The reward model in Space Invaders provides insightful information regarding overestimation.

To understand whether overestimation occurs, we conducted two analyses:

1. **Comparison of Average Rewards**: We compared the average rewards collected during both evaluation and training modes to observe any trends indicating overestimation.
2. **Double Deep Expected SARSA**: We implemented Double Deep Expected SARSA to demonstrate any resemblance in trends compared to our model. Double Deep Expected SARSA is known to reduce overestimation, as shown in previous studies (5).

Figure 4 demonstrates insightful trends. Based on subfigures (a) and (c), it can be concluded that overestimation is observed in our model. However, this trend is also present in Double Deep Expected SARSA. Although Double Deep Expected SARSA performed better compared to our model in evaluation mode in some episodes, the difference between the average rewards collected during evaluation is insignificant. Additionally, our model exhibits lower average deviation than Double Deep Expected SARSA, indicating greater stability. Therefore, it can be concluded that our model does not suffer from overestimation.



(a) κ:= 512

(b) κ:= 1024

(c) κ:= 2048

(d) κ:= 4096

Figure 5: Investigating model stability across different κ values

The poor performance observed in both our model and Double Deep Expected SARSA might stem from the fact that, although the models demonstrates signs of learning, they cannot establish an optimal policy to achieve high performance during evaluation under the current setup. As discussed in 6, the computational resources were not adequate for comprehensive experimentation.

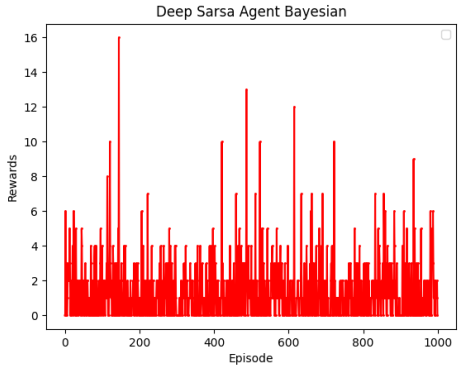
5.3 Model Update Strategy

After carefully selecting the best exploration strategy in Section 5.1, we further investigated the impact of different κ values on performance in Breakout environment. The update interval is crucial for the model to update and establish an stable optimal policy while consistently exploring various paths. The initial κ value for the preliminary experiments was 128. We experimented with different κ values: 512, 1024, 2048, and 4096.

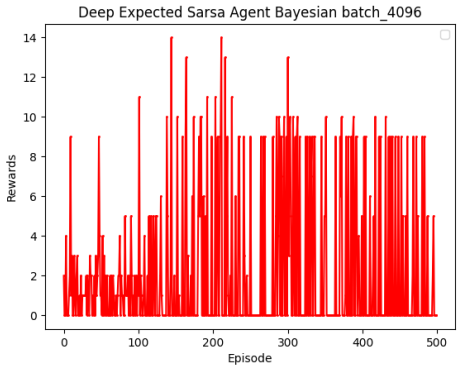
Figure 5 demonstrates a clear trend among different κ values, indicating that the model performs best when κ is set to 4096. This suggests that less frequent updates lead to better performance. The model achieved rewards higher than 12 for the first time with $\kappa = 4096$. Several factors might contribute to this pattern. Frequent updates with fewer samples can result in high variance in gradient estimates, causing instability and poor convergence, which ultimately reduces performance. In contrast, accumulating more experiences allows the model to develop a more accurate representation of the state-action space, which enables the Bayesian structure to produce better uncertainty estimates.

5.4 Comparative Analysis

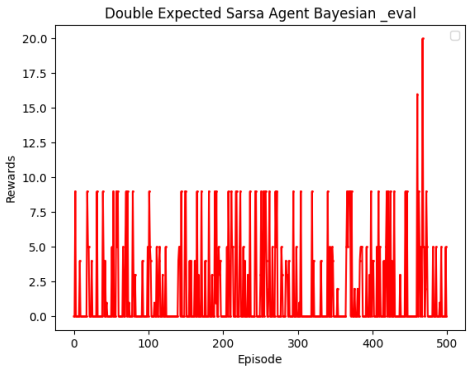
After carefully selecting the best exploration strategy and κ, we provide further comparative analysis by experimenting several models, including our own, in three Atari environments: Breakout, Space



(a) Deep SARSA



(b) Deep Expected SARSA



(c) Double Deep Expected SARSA Training

Figure 6: Evaluation mode in Breakout Environment

Invaders, and Pong. The tests are performed using a moderate exploration strategy with long duration, with κ set to 4096. [2]

5.4.1 Breakout

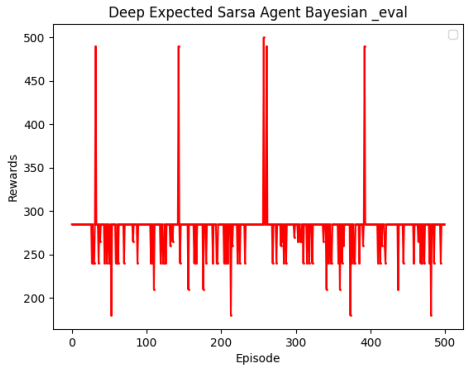
Figure 6 demonstrates the comparative performance of Deep Expected SARSA, Deep SARSA, and Double Deep Expected SARSA models in Breakout environment. Our model distinctly outperforms the other models, producing higher average rewards during evaluation mode. Among the models tested, Deep SARSA exhibits the worst performance. While Double Deep Expected SARSA achieves higher rewards than our model in specific episodes, it cannot establish a policy that sustains its improved performance.

5.4.2 Space Invaders

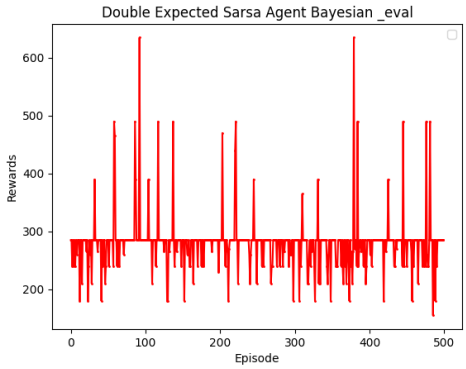
Figure 7 illustrates the comparative performance of Deep Expected SARSA, Deep SARSA, and Double Deep Expected SARSA models in the Space Invaders environment. Our model slightly outperforms the others, yielding higher average rewards during evaluation mode. Although SARSA achieves higher rewards in some episodes, it results in lower average rewards with higher variance. Additionally, this experiment reveals an informative pattern: both our model and Double Deep Expected SARSA seem to be stuck at a reward threshold and fail to achieve higher rewards frequently. This behavior is due to the environment dynamics and the update strategy.

The Space Invaders environment provides a larger action space with more complex action-state relationships compared to test environments like Pong and Breakout. The Breakout environment

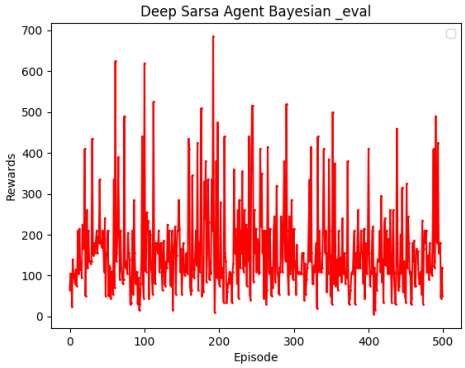
[2]Due to insufficient computational resources, we were unable to train the Deep Q-Learning and Double Q-Learning agents with Bayesian framework.



(a) Deep Expected SARSA



(b) Double Deep Expected SARSA



(c) Deep SARSA Training

Figure 7: Evaluation mode in Space Invaders environment

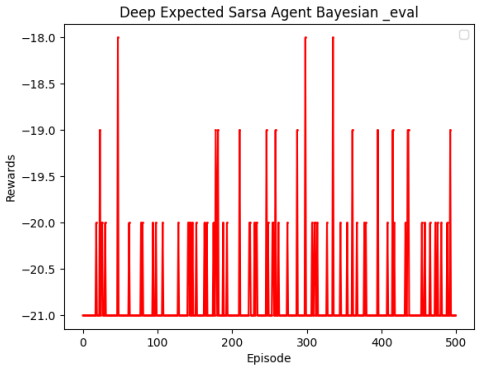
provides two discrete actions, simplifying the action-state relationship. Pong provides five discrete actions, though two are only applicable at the beginning of the episode. In contrast, Space Invaders offers five distinct actions that are always applicable during the episode. As the action space expands, the complexity and the causal relationship between actions and states increases. Our model and Double Deep Expected SARSA struggle to extract these complex action-state relationships due to their update strategy. The expectation term leads to conservative updates, causing the models to be stuck in a suboptimal policy that consistently achieves the same average reward (300 in our case). Therefore, Deep SARSA produces high rewards more frequently due to its update strategy.

5.4.3 Pong

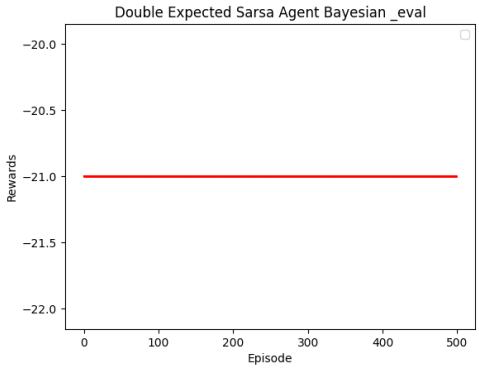
Figure 8 illustrates the comparative performance of Deep Expected SARSA, Deep SARSA, and Double Deep Expected SARSA models in the Pong environment. Deep SARSA slightly outperforms both our model and Double Deep Expected SARSA. Although there is a slight difference in the average rewards collected during evaluation between our model and Deep SARSA, the Deep SARSA agent performs better. While our model was able to achieve higher rewards for some episodes, it fails to produce higher average rewards. Moreover, our model yields higher variance, indicating instability compared to Deep SARSA. This performance pattern is attributable to the reasons explained in the previous section 5.4.2.

6 Conclusion and Future Directions

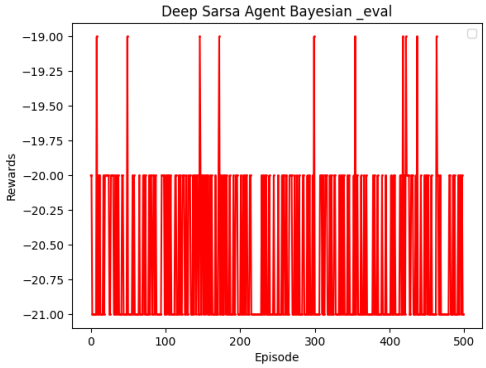
This project introduced an on-policy algorithm, Deep Expected SARSA, for playing Atari games, integrating Bayesian layers into the neural network structure to enhance decision-making in complex environments. By incorporating the Expected SARSA update rule and Bayesian framework, our



(a) Deep Expected SARSA



(b) Double Deep Expected SARSA



(c) Deep SARSA Training

Figure 8: Evaluation mode in Pong environment

model aims to improve stability and performance compared to traditional SARSA and Q-Learning methods. Through detailed experiments, the Deep Expected SARSA algorithm demonstrated promising performance, particularly in reducing variance and achieving higher rewards in several environments such as Space Invaders, Pong, and Breakout. However, the experiments also revealed challenges, such as the inability to establish an optimal policy in more complex environments with wide action space like Space Invaders and Pong. Although our experimental setup did not provide comparative results for Q-Learning, it has shown that SARSA with experience replay achieves higher rewards than Q-Learning in various environments, though with slower convergence (2).

Furthermore, this project provides valuable insights into several key areas for further enhancing the algorithm's performance. Firstly, incorporating Thompson Sampling into the exploration strategy could result in more efficient exploration and performance in complex environments (6). Secondly, expanding experimentation on update interval (κ) values can help observe the impact of accumulating more experiences before updating the model. Our experiments revealed a trend indicating that increasing the (κ) value enhances overall performance with more comprehensive representation of the state-action space, thus improving stability and performance. Lastly, employing parallel computing with CUDA technology can allow more comprehensive experimentation and comparison with other baseline models such as Deep Q-Learning and Double Deep Q-Learning.

While Deep Expected SARSA shows significant potential, further exploration of advanced exploration strategies, optimization of parameters, and an enhanced experimental setup are essential steps towards capturing its full capabilities in playing Atari games.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[2] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on sarsa," pp. 1–6, 12 2016.

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015.

[4] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," 2015.

[5] W. Hu, "Double sarsa and double expected sarsa with shallow and deep learning," *Journal of Data Analysis and Information Processing*, vol. 04, pp. 159–176, 10 2016.

[6] Z. C. Lipton, X. Li, J. Gao, L. Li, F. Ahmed, and L. Deng, "Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems," 2017.