An Evaluation of Reinforcement Learning Agents

Team Members : Aasish Tammana (1225545568), Akhilesh Udayashankar (1225622476),
Prem Suresh Kumar (1225492151), Devadutt Sanka (1225362138)

[1]

Abstract—**In the world of Artificial Intelligence, Q learning is one of the prominent techniques used for reinforcement learning which helps the agent make the best decision under a given situation. Approximate Q-learning has some limitations; the policies may not be similar, it does not explore other routes if available and is quite slow. Therefore, we have implemented an on-policy algorithm called True online Sarsa into the Pacman code base of project 4 and compared the results at the end of the report. We find out the difference between the two algorithms and perform the student's t-test on various environments and analyze our algorithms based on the t-test results.**

Index Terms—**Artificial Intelligence, Reinforcement Learning Agent, Q-Learning, Linear Function Approximation, True online Sarsa, Student's T-Test, Eligibility Trace, Features**

I. INTRODUCTION

Reinforcement learning is a method of learning that allows an agent to learn in an environment that is communicative by using feedback from its actions and experiences in a trial-and-error fashion. This type of learning allows us to reward the agent positively for desired actions and penalizes it for undesired actions to achieve an optimal solution. With continuous feedback on its correct learning, the agent avoids the negative rewards and tries to seek the positive rewards. The current implementation of reinforcement learning is in the fields of robotics, games and management. Reinforcement learning performs well in complex and equivocal situations. These algorithms reward the right actions and penalize the agent for wrong actions whenever the agent makes an action.

In particular, this project compares two reinforcement learning algorithms - Q learning with linear function approximation and the True Online Sarsa. For the comparison, both the environments and the parameters are the same. The Approximate Q learning which was implemented in project 4 is an off-policy algorithm while Sarsa is an on-policy algorithm. Sarsa, being an on-policy algorithm, will update its policy with the help of the Q value of the next state and the current state policy action. It calculates the state action pair while assuming it follows the previously used policy. Sarsa will avoid high-risk paths and prevent the Pacman agent from exploring risky routes, it avoids the optimal path if there is a high negative reward in the optimal path initially. It would, later on, learn the optimal path through exploration.

In this project, we aimed to implement the True Online Sarsa algorithm to the existing code base of Project 4 which already has Q learning algorithms implemented. We discuss this further in the following section. The performance and the results are presented in the last section

II. TECHNICAL APPROACH

As was introduced in the earlier section, Sarsa is an on-policy algorithm that allows an agent in a state S to take an action A to reach S' for a particular reward because of taking that action. Here, we discuss the algorithms implemented and compared ie., the approximate Q learning algorithm and the True Online Sarsa algorithm.

The Q learning algorithm is to learn the Q value function iteratively. This can be achieved using the Bellman Optimality Equation. Q learning is an off-policy algorithm meaning that it will take a policy that maximizes its Q value paying no attention to what policy it followed in that state.

```
function Approximate Q-LEARNING-AGENT (percept)
returns an action
Inputs: percept, a percept indicating the current state s and reward signal r'
persistent: Q, a table of action values indexed by state and action, initially zero.
        $N_{sa}$, a table of frequencies for state–action pairs, initially zero
        s, a, r , the previous state, action, and reward, initially null
        W, weights vector
if TERMINAL? (s) then Q[s,None] ← r'
if s is not null then
        increment $N_{sa}$[s,a]
        difference ← [r+$\gamma$max$_{a'}$ Q[s',a']]-Q[s,a]
        Q[s,a] ← Q[s,a]+$\alpha$( difference )
        w ← w + $\alpha$(difference)$N_{sa}$
s,a,r ← s',argmax$_{a'}$,f(Q[s',a'],$N_{sa}$[s',a']),r'
return a
```

Fig 1 Approximate Q-Learning Algorithm

While the basic Q learning algorithm keeps a table of all q-values, it is impossible to learn every single state as there are too many states to visit them in the training, and also more memory is needed to store the states.

Therefore a feature of vectors that captures important properties of states is used in Approximate Q learning. The algorithm interprets that the weights are adjusted for all active features.

Sarsa on the other hand is an on-policy algorithm. It means that we use the existing policy considered to take action and update the value function. Sarsa also uses the TD approach i.e, continuously updates the Q table.

[1]

True online Sarsa(λ) for estimating $w^T x \approx q_\pi$ or q_*

```
Input: a feature function $x: \delta^+ \times A \rightarrow R^D$ Rd such that x(terminal, ·) = 0
Input: a policy π (if estimating $q_\pi$)
Algorithm parameters: step size α > 0, trace decay rate λ ∊ [0, 1], small $\varepsilon$ > 0
Initialize: w ∊ $R^D$ (e.g., w = 0)
Loop for each episode:
    Initialize S
    Choose A ~ π (·|S) or $\varepsilon$ - greedy according to $\hat{q}$(S, ·,w)
    $x \leftarrow x(S,A)$
    $z \leftarrow 0$
    $Q_{old} \leftarrow 0$
    Loop for each step of episode:
        Take action A, observe R, S′
        Choose A′ ~ π (·|S) or $\varepsilon$ - greedy according to $\hat{q}$(S′, ·,w)
        $x' \leftarrow x(S',A')$
        $Q \leftarrow w^T x$
        $Q' \leftarrow w^T x'$
        $\delta \leftarrow R + \gamma Q' - Q$
        $z \leftarrow \gamma\lambda z + (1 - \alpha\gamma z^T x)x$
        $w \leftarrow w + \alpha(\delta + Q - Q_{old})z - \alpha(Q - Q_{old})x$
        $Q_{old} \leftarrow Q'$
        $x \leftarrow x'$
        $A \leftarrow A'$
    until S′ is terminal
```

Fig 2: True Online Sarsa Algorithm

In the algorithm mentioned, the input is a set of features that contains a state, action, and policy. We keep the alpha size greater than 0 and lambda in the range of 0 to 1. The values of the weights and the eligibility traces for all actions are updated in each episode. The discount factor is calculated and updated in each episode's action taken. The corresponding Q value is updated after the calculations and the most rewarding action according to the policy is chosen.

As mentioned, the Q learning agent was already present in the Project 4 code base and we implemented a class TrueOnlineSarsa for the Sarsa agent. The Sarsa class code is present in the same code base where the Q learning agent is also present.

The update function of the Sarsa class is implemented according to the Sarsa algorithm mentioned above. The corresponding algorithms were tested for their functionality.

The difference is subtle between the two algorithms based on their update functions, Q-learning takes the prime Q-value from the state where the transition occurred whereas, Sarsa, will wait for the action to be taken and takes the prime Q- value for that action. They only differ to a certain extent only when the agent is in the exploration state. It is because Q learning takes the max value obtained ignoring the policy. Sarsa on the other hand is more pragmatic.

III. RESULTS AND ANALYSIS

After the implementation of the True Online Sarsa algorithm, we generate various layouts automatically through a script. We design layouts by defining some probabilities for the various components of the maze i.e, wall, food pellet, ghost, Pacman. First, we ensure the boundaries are the walls. Then we randomly place ghosts by limiting the maximum number of ghosts to 3. We also check that the agent has a path to reach all the food pellets and that the game can be completed. This is done by assigning boundary probabilities to the various components and restricting the layouts to handle corner cases.

Once the layouts are generated, we run the True Online Sarsa and Approximate Q learning algorithms on these layouts. For better comparison, we run the True Online Sarsa training on these layouts about 5 times for each layout by varying lambda values between [0.1, 0.3, 0.5, 0.7, 0.9]. We run the approximate Q learning once on each layout. Below are the sample results of our analysis.

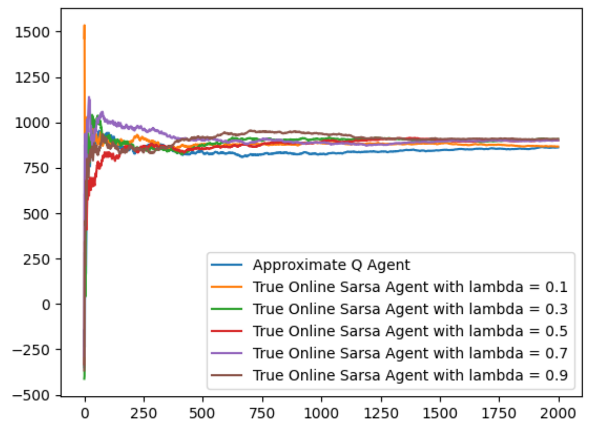

Fig 3 Tricky Classic Layout Output

As observed in Figure 3, The lambda value of 0.3 has the best convergence. Also, we can observe that in general, the true Online Sarsa algorithm performs better than the Approximate Q learning.

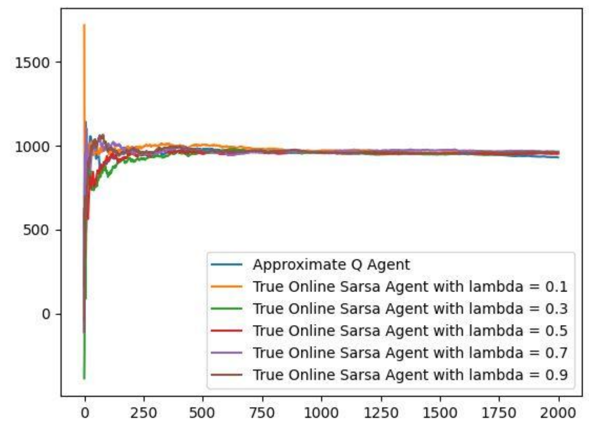

Fig 4 Medium Classic Layout Output

Similar results are observed in the Medium Classic Layout output in Figure 4. The results of all the layouts and the graphs have been plotted. Another observation was that the lambda value of 0.9 was almost all the time underperforming in comparison to all other lambda values.

The results of each run here were stored in separate JSON files which were then parsed using a script to compute the Average value for the episodes. For the purpose of the comparison in this project, we have generated 20 layouts with sizes ranging between 7 * 7 and 15 * 15. We combine all the data for each layout and plot them in one graph. This would result in a total of 30 graphs which then be visualized to finalize the lambda value for our analysis. A couple of these graphs are depicted in Figures 5 and 6 after we finalize the lambda values.

Then we run the Student's T-test for statistical inference. T-Test was developed by William Gosset to mathematically compare multiple distributions and determine how similar or different they are by using the null hypothesis as the base. The null hypothesis assumes that both samples are almost similar. In this project, we perform the one-tailed paired two sample t-tests. We do the paired t-test to compare the same episodes from each sample. For this, we have to consider samples of the same size and assume they have similar variances.

Mathematically, this can be represented as follows:

$$T = \frac{(\bar{X} - \mu)}{S/\sqrt{n}}$$

$\bar{X}$ indicates the sample mean,
μ represents the hypothesized group mean,
S is the standard deviation of sample,
n is the number of observations in the sample

We can also compute p-values to perform the student's T-test. The basic understanding of the p-value is as follows. If the p-value is lesser than or equal to 0.05, then we can say that the difference between the two paired samples is significantly higher. The p-value is in fact calculated corresponding to the abs value of the T-Test statistics based on the degrees of freedom which is given by n-1.
In this project, we have used the ttest_rel() module from scipy.stats to compute our p-values directly on the data and the results of our analysis on some of the layouts have been represented in Table 1. We have tabulated results for some of the best layouts where our layouts were resulting in the p-value which is less than 0.05.

Layouts	p
generatedlayout15	0.014622317550698847
generatedlayout10	0.03474889597809033
generatedlayout12	0.04893450370767481
generatedlayout0	0.04988572660259601
contestClassic	0.01010969525287796

Table 1: p values from the t-test

Episode	Approximate Q Learning	True Online Sarsa
500	450.26	489.63
600	460.51	484.26
700	469.58	483.15
800	470.84	481.07
900	465.24	480.32
1000	473.15	478.76

Table 2: Average rewards for a range of episodes

As you can see from the above table, the average rewards converge and remain the same for the Sarsa algorithm than the Approximate Q learning algorithm. Over a range of episodes from the games run on the generated layouts, we find that the Sarsa algorithm converges faster than the Approximate Q learning algorithm. This convergence can be viewed in the following figures.

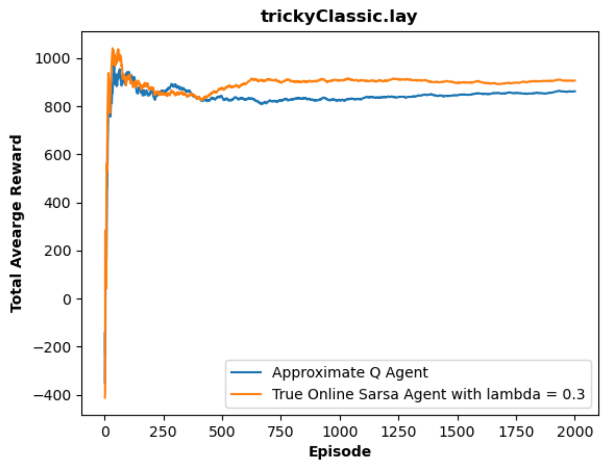

Fig 5 Tricky Classic Layout Output

We plot Average Rewards against episodes trained. In figure 5, we depict the output of the tricky classic layout. Apart from better convergence, the True Online Sarsa algorithm also has significantly higher average rewards

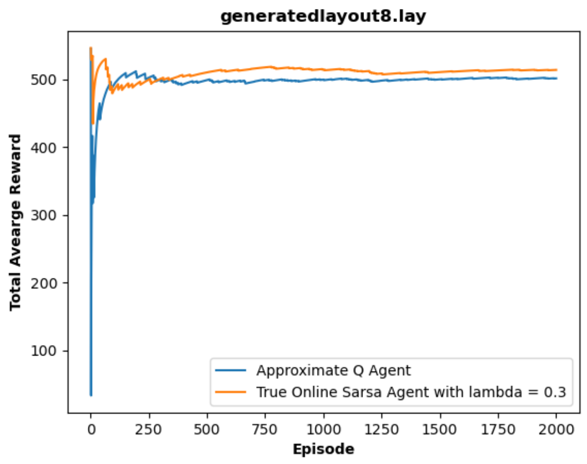

Fig 6 Generated Layout Output

Similarly, in figure 6, we have depicted the output for one of the layouts generated through our script. This has been named Generated Layout 8 and has similar observations as the earlier scenario.

IV. Conclusion

In this project, we have implemented True Online Sarsa in the existing Pacman code base of Project 4 and compared the results of the Q-learning algorithm with linear function approximation and True Online Sarsa. We found that the True Online Sarsa performs better than the Q learning algorithm in various environments but we concede that the Approximate Q learning algorithm performs better in some cases. Initially, we try many lambda values and determine which parameters perform better for our environments. We run our analysis on Approximate Q learning versus Sarsa and plot various graphs analyzing the convergence between the two algorithms. We also performed Student's t-test in the final section of our project and determined the significance of the two algorithms. In conclusion, for the layouts generated by our script, we have determined that the lambda value of 0.3 has resulted in a notable amount of difference between the approximate Q value algorithm and the true online sarsa algorithm. We have observed that, out of 33 layouts available, 26 layouts resulted in a better convergence for the True Online Sarsa Agent as indicated by the T-test.

References

[1] Richard S. Sutton and Andrew G. Barto, ”Reinforcement Learning: An Introduction”. Second Edition. pp. 105-110. MIT Press, Cambridge, MA. November 5, 2017. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[2] D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," 2010 Second International Conference on Machine Learning and Computing, 2010, pp. 317-320, doi: 10.1109/ICMLC.2010.38.

[3] Judith Fechter, Andreas Beham, Stefan Wagner & Michael Affenzeller (2019) Approximate Q-Learning for Stacking Problems with Continuous Production and Retrieval, Applied Artificial Intelligence, 33:1, 68-86, DOI: 10.1080/08839514.2018.1525852

[4] Rafipoor, Hossein. 2013. Reinforcement Learning Part 2: SARSA vs Q-Learning. Studywolf. https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/.

[5] Aldrich J (2008) Comment on S. L. Zabell's paper: on Student's 1908 paper. The probable error of a mean. J Am Stat Assoc 103(481):8–11

[6] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[7]van Seijen, Harm & Mahmood, A. & Pilarski, Patrick & Machado, Marlos & Sutton, Richard. (2016). True Online Temporal-Difference Learning. Journal of Machine Learning Research. 17. 1-.