Which Approximate Method is the best for your environment?

Applying Multiple Approximate Methods in a Pacman Game

Yeonjung Lee
1211201664
Arizona State University
ylee197@asu.edu

Upasana Biswas
1222426790
Arizona State University
ubiswas2@asu.edu

Pavan Kumar Gurram
1220139960
Arizona State University
pavangurram@asu.edu

Pavan Sai Varma Budde
1219477246
Arizona State University
pbudde@asu.edu

***Abstract*—Reinforcement learning method is developed from true value method to approximate value method and from on-policy method to off-policy method. From the experiment with different reinforcement methods, we reviewed state-of-the-art algorithm and discussed the benefit of each method. After the analysis, we pursue to find a more improved algorithm. We created three new environments, and compared the performance of different algorithms on each environment on the basis of average rewards and win rate. From this experiment, we reviewed contemporary techniques and sought a way to improve it.**

***Index Terms*—Reinforcement learning, Approximate Method, Approximate Q-learning, Episodic Semi-gradient SARSA, True Online SARSA and Pacman domain**

I. Introduction

We studied, implemented and tested various approximate reinforcement learning algorithms - Q-learning, Episodic Semi-Gradient Sarsa and True Online Sarsa(λ) - for controlling our rational agent in the Pacman Domain. Approximate Q-learning with linear feature approximation enables us to learn the state values even in large state-spaces. To optimise the learning behavior of our agent, we implement Episodic Semi-gradient SARSA, which is an on-policy learning method. However, since semi-gradient methods are not true gradient descent algorithms, we want to get a better estimate of the expected return, and implement True Online Sarsa which uses Dutch traces to efficiently estimate state-action values.

A. Approximate Q-learning

Approximate Q-learning method is invented for efficiency. In this method, we use linear equation. This linear equation provides more efficient way to predict next state with smaller storing space and computation time.

Q-learning method started with true values. However, it requires large storage spaces for state-action pair. For example, in Pacman domain, the standard grid has dimensions of $20 \times 11 = 220$, each block can have a substate out of 1,280 substates ($2^5 \times 40 = 11,280$) states. Therefore, there would be about 281k states, that can occur randomly.

Space is not the only issue in Q-learning. This method is also time-consuming and inefficient. Thus, people seek a way to use approximate values. This approximate value is not same as true values. However, when it works through enough iteration and update, we can get a merged value to the true value.

Commonly used function approximators are linear combination of features and neural networks. Linear function approximation with Stochastic Gradient Descent is most famous one because of the simplicity of weight updates and the guarantee of converging to the global optimum. Since the objective function is convex in shape, the Stochastic Gradient Descent is relatively simple to find the optimal weights. In this reason, we will use the Stochastic Gradient Descent model.

B. Episodic Semi-Gradient SARSA

Sarsa acts on a behaviour policy through generating experience in the environment. In this way, Sarsa can learn the Q values from each state-action pair, unlike Q learning learns the Q values with the greedy policy.

While Approximate Q-learning is an off-policy learning method, Sarsa is on-policy method. The problem with Q learning is that it could end up with a negative reward because it chooses a random action to learn the optimal policy at each status. On-policy Sarsa learns action values based on the policy that is currently following. Therefore it is more conservative when it comes with the agent's behaviour during the learning process. This Sarsa method can overcome a downside of Q-learning. In this reason, in real-world, an algorithm like Sarsa is typically preferred over Q-learning where we care about the rewards gained whilst learning and the mistakes of the agent could be costly. In algorithms where the target value functions are bootstrapped, they are not independent of the current value of w, which violates a key assumption of gradient update rule. Therefore, semi-gradient methods include only a part of the gradient and ignore the effect of changing the weight vector on the target. We will use Episode Semi-Gradient Sarsa for the approximate on-policy method.

C. True Online SARSA

One of the main problems in Reinforcement Learning is to make predictions about the return i.e. the sum of future rewards, following a certain policy in an unknown environment. Temporal Difference learning (hereafter TD learning) is a technique which allows us to learn estimates of the expected return by bootstrapping. TD(λ) is a method which combines TD Learning with the concept of eligibility traces: a technique to maintain history of how the reward we are observing is affected by previous states we have visited.

For Sarsa(λ), the only difference in the algorithm is that it uses the approximate action values instead of the approximate state values. The values computed by true online Sarsa are exactly the same as those computed by the ideal online λ-return algorithm. In this reason, we named it "truer" even it's approximation. We will use True Online Sarsa for the approximate off-policy method.

In section 2, we discuss about the each approximated method and our implementation and we discuss and analyze our result in section 3. The final section 4, we provide our conclusion and future work.

II. Technical approach

Approximated reinforcement learning method starts from the approximated Q-learning algorithm. Approximated Q-learning algorithm uses each feature and calculates based on the features. The weight is updated during a training process. After training the learning algorithm for enough iterations, you can find a point that each feature weight is converged to. Through the weight updating process, each feature can find its proper weight values, thus leading to a good result. Therefore, we need to define features and initialise them with values at the very beginning. For the Pacman domain, we use the following features: Eatable food in current position, Distance from the next food, Ghost location in the next step, and Collision with the ghost. To train the Pacman agent, we set up the reward function. When the agent eats food, it can earn 10 points. If there is no more food, the agent gets 500 points and the game is ended. When the agent collides with a ghost, it loses 500 points and the game ended. For the update of the weight, we set the learning rate as 0.2. Thus, 20% of the differences in the weight is applied to the new weight.

A. Approximate Q-learning

Approximate Q-learning is the simplest method among all three approximate reinforcement learning algorithms. Approximate Q-learning calculates estimated Q values with the following equation.

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + ... + w_n f_n(s,a) \quad (1)$$

$$difference = \left[r + \gamma \max_{a'} Q(s^{'}, a^{'}) \right] - Q(s,a) \quad (2)$$

$$w_i \leftarrow w_i + \alpha \left[difference \right] f_i(s,a) \quad (3)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[difference \right] \quad (4)$$

This equation(1) considers only the weight of each feature value. Then, we calculate the differences between actual sample values and estimated Q values(2). With these differences, we can calculate the new weight(4). Updating this weight(4) enough times, we can drive our agent to finish its goal. To save the weight vector, we uses a dictionary structure (Python data structure).

B. Episodic Semi-Gradient SARSA

In this section, we introduce a new concept SARSA. SARSA is one of the reinforcement methods and it considers a state, an action, a reward, the next state and the next action. Episodic semi-gradient SARSA uses the general gradient-decent update for the next weight. However, Episodic semi-gradient SARSA considers a state, an action, a reward, the next state and the next action rather than an action-value, so formula is the same as the follows.

$$difference = R_{t+1} + \hat{q}(S_{t+1}, A_{A+1}, w_t) - \hat{q}(S_t, A_t, w_t) \quad (5)$$

$$w_{t+1} = w_t + \alpha \left[difference \right] \hat{\nabla(q)}(S_t, A_t, w_t) \quad (6)$$

Using above equation(6), it updates weight in each episode. The weight is calculated with the difference(5) between approximate next state-action Q values and approximate current state-action Q values. Thus, we can get the weight that has stable Q values after enough weight updates.

C. True Online SARSA(λ)

The True online SARSA is an off-policy method in the category of approximate methods. Basic idea of the True online SARSA is that the latest weight is the closest weight to the actual weight. To get the better result, we need to find the most similar weight with the actual value. Since we train the model enough to get the closest weight, the last weight is the most closest weight to the actual weight.

$$\begin{array}{cccccc} w_0^0 & & & & & \\ w_0^1 & w_1^1 & & & & \\ w_0^2 & w_1^2 & w_2^2 & & & \\ w_0^3 & w_1^3 & w_2^3 & w_3^3 & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ w_0^T & w_1^T & w_2^T & w_3^T & \cdots & w_T^T \end{array} \quad (7)$$

To get the better weight, we recalculate whole sequence of episode from the very beginning using equation(7). Every episode, it recalculates the weight from the first episode, and the method takes the last weight such as $w_0^0, w_1^1, w_2^2, \ldots, w_T^T$ for the True-online SARSA weight calculation.

$$G_{t:t+n} \doteq R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1}) \\ , t+n < T \quad (8)$$

$$w_{t+1} \doteq w_t + \alpha [G_t - \hat{q}(S_t, A_t, w_t)] \nabla \hat{1}(S_t, A_t, w_t) \\ , t = 0, ..., T-1, \quad (9)$$

To calculate the weight, we use equation(9). Since this equation requires recalculating the whole sequence of episode every time, we need to save state-action information of each episode in a list. Every episode, the weight is updated after calculating with whole sequences of episode with new weight. For this intermediate weight, we created another weight dictionary (Python data structure) to save the values. After this series of iterations, we get the final weight - w_T^T. The weight vector affects the current $\hat{q}$ value. It makes differences in equation (13) because there is a difference between $\hat{q}_{old} = w_{T-1}^{T-1}$ and $\hat{q} = w_T^T$.

$$\hat{q} \leftarrow \boldsymbol{w_T^T} * \boldsymbol{x} \tag{10}$$

$$\delta \leftarrow r_k + \gamma \hat{q}' - \hat{q} \tag{11}$$

$$\boldsymbol{z} \leftarrow \gamma\lambda\boldsymbol{z} + (1 - \alpha\gamma\lambda\boldsymbol{z}^\top\boldsymbol{x})\boldsymbol{x} \tag{12}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(\delta + \hat{q} - \hat{q}_{old})\boldsymbol{z} - \alpha(\hat{q} - \hat{q}_{old})\boldsymbol{x} \tag{13}$$

D. New Environment

For analysis of the algorithms, we created a new environment as a maze, and then added Pacman, Ghost and Food positions according to our requirements. We went through several maze-generation algorithms and used Recursive Backtracker Algorithm since it is fast to implement and is memory-efficient: proportional to the size of the Maze. This algorithm works by randomly walking through the maze until it can no longer go forward. Once this happens the walker takes one step back and tries another direction. This process continues until the walker is back at the starting location. The name is derived from this fact, as the walker is 'Backtracking' and repeating its actions recursively.

III. Result

In this section, we analyze and discuss about the results of our experiment. We have implemented three reinforcement learning methods in Pacman domain and compared its results in different environments. We have used six different environments for our Pacman domain which are SmallGrid (7x7), MediumGrid (7x7), MediumClassic (20x11), RandomMedium (11x11), RandomMediumWithGhosts (11x11) and RandomMediumWithFood (11x11). We have performed our tests using T-test and ANOVA: single Factor test for comparing results on three learning algorithms. *We have chosen ANOVA test because ANOVA signifies best results on three or more groups of independent variables.* We implemented T-test also for this experiment but T-test works well when only two dependent or independent groups are present. In our case, we have to perform T-test selection on two of the three variable groups at one time and this would not result in best analysis. Unlike T-test, ANOVA test works perfectly on three groups at a time and display results signifying the difference between them.

As described in the Fig. 1 the MediumClassic environment exhibits the best scores for all three algorithms than the remaining five environments. MediumClassic is the appropriate layout for Approximate Q-learning, Semi Gradient and

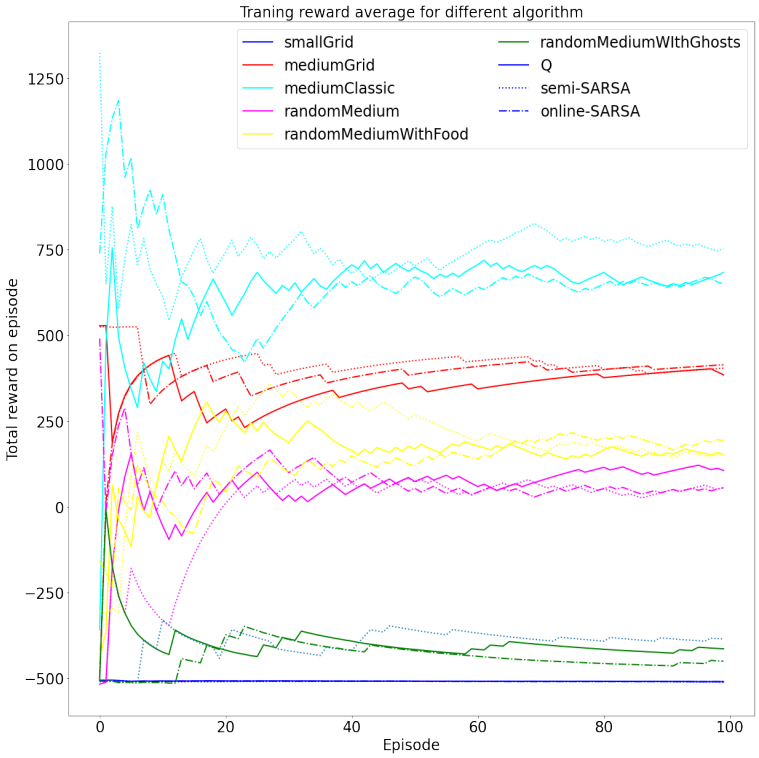

Fig. 1. Average Scores for Methods in 6 Layouts

True Online Sarsa. The SmallGrid environment can also be a good environment for this experiment but it does not show significantly high learning courves with 100 times training. According to our experiment, the smallGrid shows reasonable result over 1,000 times training. We discused about the reasons why the smallGrid need more training. Our conclusion is that the smallGrid environment have few foods and lots of empty space. It gives to the agent to more option to move for the next step. Therefore, it needs more training time. The Random Medium environment is made by adding couple of blocks in MediumGrid. However, the added blocks create some dead end situations. In the dead end situation, Pacman cannot avoid the ghost. Since our program only checks the next state, the Pacman cannot avoid this situation that would have require to check more than 5 steps forward. On the whole, the MediumClassic Layout has given the best scores of these six layouts. Below are the analysis on the MediumClassic Layout.

Fig. 2 clearly indicates that True online SARSA shows exponentially increased run times than Episodic Semi-Gradient SARSA and Approximate Q- learning methods in MediumClassic Layout. The average run time is 55.14 for True online SARSA, 4.62 for Approximate Q learning and 4.69 for Episodic Semi-Gradient SARSA. We can observe that average times for Approximate Q learning and Episodic Semi-Gradient SARSA is similar in our experiment but Episodic Semi-Gradient SARSA performs better than Approximate Q learning which is discussed below.

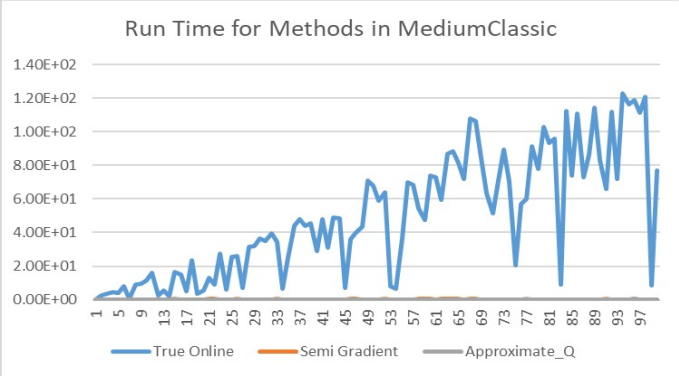

Fig. 2. MediumClassic Running Time

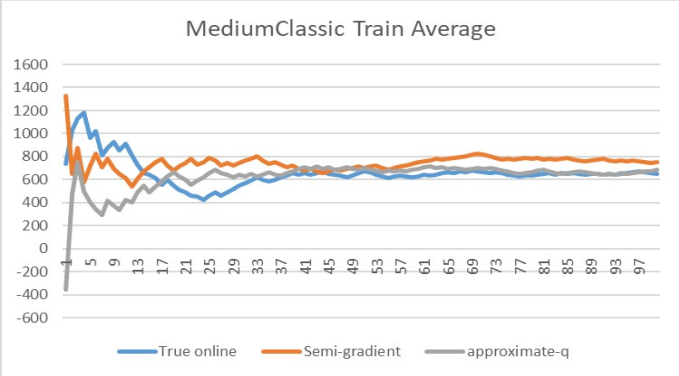

Fig. 3. MediumCLassic Training Scores

In Fig.3, for the training set we have taken 100 samples for each algorithm and computed the average score for each sample on these three learning algorithms. The training set has given average score of 659.31 for True online SARSA, 746.74 for Episodic Semi-Gradient SARSA, 624.99 for Approximate Q-learning.

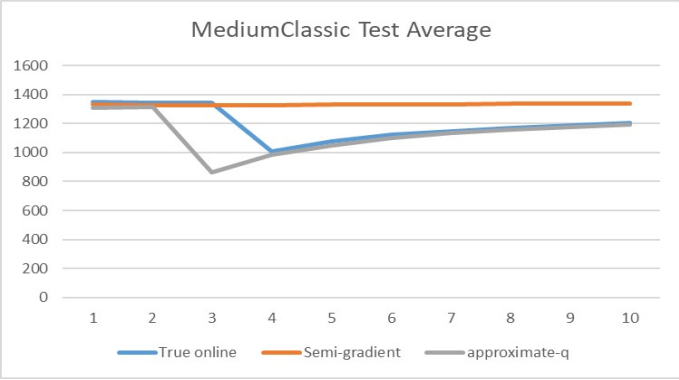

Fig. 4. MediumClassic Test Scores

In Fig.4, for the test set we have taken 10 samples for each algorithm and computed the average score for each sample on these three learning algorithms. The test set has given average score of 1194.66 for True online SARSA, 1331.08 for Episodic Semi-Gradient SARSA, 1127.96 for Approximate Q-learning.

Groups	Count	Sum	Average	Variance
True Online	110	77,878.29	707.9845	39,453.8
Semi Gradient	110	87,985.71	799.8701	34,181.34
Approximate Q	110	73,779.54	670.7231	39,408.11

TABLE I
ANOVA SCORE MEASURES FOR METHODS IN MEDIUMCLASSIC

Source of Variation	SS	df	MS	F	P-Value	F crit
Between Groups	972,044.9	2	486,022.4	12.90	$0.406*10^{-5}$	3.02
Within Groups	327	37,681.08				

TABLE II
ANOVA TEST RESULTS FOR METHODS IN MEDIUMCLASSIC

We perform ANOVA test on MediumClassic environment and test results are shown in Table I and Table II. Here in the ANOVA test, the SS value indicates the degree of variation among the data of these three algorithms, DF indicates degree of freedom, MS indicates mean squares and P indicates prism values.

The results of ANOVA test are promising for this experiment as the value of F is greater than value of Fcrit. This implies that we can reject the null hypothesis which means all the three algorithms are not similar and atleast one of them is different. Also the average scores for total sample for all three algorithms are also provided. These scores indicates that Episodic Semi Gradient SARSA is the best performing reinforcement learning algorithm.

IV. ANALYSIS

A. Method

We have performed our experiment on three reinforcement learning algorithms - ApproximateQ, Semi-Gradient and True Online SARSA. When we run these three methods in the six mazes exhibited some astonishing results. The execution time of True Online SARSA is found out to be more than the execution times of other two learning methods and Other two methods had similar kind of execution times. Another Important analysis of True Online SARSA is that it performs poorly when food and ghosts count is increased. This can be attributed to the behaviour of pacman to not forsee the future states beyond the next state. So the win rate and scores of True online SARSA is way behind than the other two algorithms. Although the execution times of Semi gradient Sarsa seems similar to Approximate Q-learning, the former performed better in both Average Scores and execution times. Its because if there is a big negative reward near the best path, it will not choose the optimal path straight away, since it will avoid the dangerous/unsafe path and gradually understand the surroundings through exploration. While, Q-learning would consider the optimal path while still keep exploring. In and

Member	Jobs	%
Yeonjung Lee	Implementing and technical approch	25%
Upasana Biswas	Creating environment and testing , abstract and Introduction	25%
Pavan Kumar Gurram	Visualization and result and analysis	25%
Pavan Sai Varma Budde	Visualization and result and analysis	25%

TABLE III
CONTRIBUTION OF EACH MEMBER

all we would choose the Semi Gradient Sarsa over the three methods for out agent to perform better.

B. Environment

We used a random maze generation algorithm to create new environments. We wanted to test the effect of the following changes in the environment:

1) Increasing Size
2) Increasing food density
3) Increasing the number of ghosts
4) Increasing number of walls

Therefore, we created three new environments, randomMedium - Increased size and number of walls, randomMediumWithFood - Food distributed in every empty space in randomMedium, randomMediumWithGhosts - 3 ghosts in the randomMedium. We observed the following:

1) Episodic Semi-gradient Sarsa performed the best on all three new environments
2) When we increased the number of food particles, we saw an improvement in Approximate Q-learning's average reward and win rate, but true online Sarsa performed best with a win rate of 9/10.
3) All three environments perform bad when we increase the number of ghosts in the environment, because it is difficult to beat three ghosts in a relatively small environment. Despite having low average rewards, approximate Q-learning and semi-gradient Sarsa learns how to beat the ghosts and win 3 out of 10 test games.

V. CONCLUSION

We reviewed, implemented and tested three approximate reinforcement algorithms - Q-learning, Episodic Semi-Gradient Sarsa and True Online Sarsa(λ). Depending on the environment and feature diversity, each algorithm shows different results. Even if, the performance of True Online Sarsa is not significantly better than Episode Semi-gradient Sarsa, when we consider the point that True Online Sarsa is the Off-policy method, it is worth to study more about True Online Sarsa. The long running time issue of True Online Sarsa definitely needs to be considered. If we have a chance to study more about this topic, the reducing time for True Online Sarsa will be the topic.

At the very last, I appriciate all my members who work hard for this project. I attached our work in Table III.

REFERENCES

[1] Russell, Stuart J. (Stuart Jonathan). Artificial Intelligence : a Modern Approach. Upper Saddle River, N.J. :Prentice Hall, 2010.

[2] Sutton, Richard S. and Barto, Andrew G.. Reinforcement Learning: An Introduction. Second : The MIT Press, 2018.

[3] Harm van Seijen, A. Rupam Mahmood, Patrick M.. True Online Temporal-Difference Learning : Journal of Machine Learning Research (JMLR), 17(145):1-40, 2016.

Method	Function name
Approximate Q-learning	ApproximateQAgent
Episode Semi-Gradient SARSA	SemiGradientSarsaAgent
True Online SARSA	TrueOnlineSarsaAgent

TABLE IV
FUNCTION NAME FOR EACH METHOD

Environment	size
smallGrid	7x7
mediumGrid	7x7
mediumClassic	20x11
randomMedium	11x11
randomMediumWithFood	11x11
randomMediumWithGhosts	11x11

TABLE V
ENVIRONMENT NAMES

APPENDIX

A. Test Environment

- Python 3.6

B. Files and Folders

- Layouts(Folder) - It contains Pacman environments.
- Output(Folder) - It contains all output folders
- Analysis.ipynb(in output folder) - This file creates the graph with all six training results.
- qlearningAgent.py - This file has all three implementation - Q-learning, Episodic Semi-Gradient Sarsa and True Online Sarsa(λ).
- learningAgent.py - This file trains and tests our reinforcement learning algorithm.
- util.py - This file has all the utilities such as datastructure including stack and queue and small functions that we use in the Pacman game.

C. Executing

- SmallGrid - python pacman.py -p FUNCTION NAME -x 1000 -n 1010 -l smallGrid
- Others - python pacman.py -p ***FUNCTION NAME*** -a extractor=SimpleExtractor -x 50 -n 60 -l ***ENVIRONMENT***
  - FUNCTION NAME is in table IV.
  - ENVIRONMENT is in talbe V.
- Example - python pacman.py -p ***SemiGradientSarsaAgent*** -a extractor=SimpleExtractor -x 50 -n 60 -l ***randomMedium***