An Exploration (and Extension) to Robby the Robot

Alejandro Castaneda

CS441 Artificial Intelligence

Anthony Rhodes

Portland State University

ABSTRACT

As an extension to the previous programming project 3 in the class, this report goes over some more in-depth insight about what factors make Robby the Robot work more accurately and efficiently. The Q-Learning agent uses many factors to make the ‘right’ decision, yet there is no subjective truth. In this report, I aim to explore how different values can lead Robby to find different actions, and how a little tweak in the learning rate, discount rate, rewards, and stochasity will affect Robby’s performance. Finding differences in all the contrasts of values, there are definitely sweet spots in which Robby will perform the best.

INTRODUCTION

Imagine a modern robot that accepts rewards and validation, much like us humans do. Now, imagine that but as a solution to basic tasks, applicable everywhere. That's the idea behind one goal for Robby the Robot. Robby lives in a 10x10 grid in which it attempts to pick up as many cans as it can. It contains a Q-Learning algorithm, which is able to identify the best possible action in a given state through a reward system. By picking up a can, Robby is granted 10 points. However, if Robby were to crash into a wall or pick up a can when there is not one, there is a punishment of 10 or 5 points respectively.

The idea behind a Q-Learning is very simple, in theory. In a given scenario, find a state representation and every action it can take in that state. From there, you can create a 'Q-Matrix', which contains each state by each action. As the agent trains, it will load the details into the matrix, preparing the agent for the test and real runs. In a nutshell, Q-Learning can be represented by the following equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right]$$

As an extension of Programming Project 3, I decided to take the Q-Learning agent a little further, and explore how the different variables affect the intelligence of Robby. There are many factors that can change the accuracy and efficiency of a Q-Learning agent, and the simple truth is that there is no right answer for all programs. Each instance of this algorithm would perform best at its own tuned parameters, with consideration for overfitting. My goal with this project is really to expand on what I had confidently built before: explore the exploration of Q-Learning, and exploit the exploitive side of it. By changing the different variables that make Q-Learning so effective, we'll see how it affects accuracy, performance, and which would be more optimal for this instance of Robby.

METHODS

We are measuring the success of each individual element by two factors: accuracy, and efficiency in that order. We care about accuracy more than efficiency for a simple reason: what good is a quick algorithm if it doesn't work? We can determine these factors specifically by measuring two data points from Robby: What the total rewards were for which epoch of Robby. The higher the quality of rewards, the more confident we are that the Q-Learning is working accurately. The less epochs it takes to reach said high quality rewards, the more confident we become in Robby getting to a solution with less computational time.

There are many ways to represent this puzzle, but with over 100 tiles with random values each time, there's no certainty on what the board will look like and it rules out a few options. Thus, since Robby can only move one tile in one direction, I decided to make the state space representation similar to what is in its current view at the moment. Therefore, there are three possible items (cans, walls, or nothingness) in the five spaces near him (left, up, right, down, current). With these pieces of information, I decided to follow a style of ternary bits to represent the space, which can be composed by the following formula.

$$Space = \sum^{d} (3^{d}) * a[d]$$

Simplified, we have a string of numbers XXXXX, in which each digit will be multiplied by 3 to the power of the index then multiplied by its value. Because there are 3 possible results and 5 spaces, there are $3^5 = 243$ possible state spaces.

To begin, there will be a 'ground truth' of variables that are compared. The ground truth of this algorithm is the assignment parameters that were given, which had produced amazing results in the past version of this project. The parameters given are: each iteration will run through 5000 episodes of training and testing. Each episode will perform 200 steps, with an alpha value (the learning rate) of .2, and a gamma (the discount rate) of .9. As with most scientific experiments where you want controls, I'll be keeping the values provided the same, only the specific variable listed will be adjusted through the experiment.

In the formula mentioned earlier, there are two variables we can change to give Robby some more 'character'. The first value we'll mess with is the alpha, **α**, which changes the learning rate of the Q-Matrix. With a low alpha value, we'll update our Q-values very little, showing nothing is really learned. With a high alpha value, we'll prioritize learning, meaning learning occurs very quickly. Of course, there is no one "answer to rule them all", thus we'll be experimenting in increments from 0 to 1. With a testing of values 0, .1, .25, .5, .75, .9, and 1, we'll be covering an entire spectrum of potential learning. Using this, we'll be able to see how quickly Robby learns, and if it's effective.

Another potential variable to change is the gamma, ɣ. ɣ stands for the 'discount rate' or in other words, how much we should discount the advancements that the agent is making in the future. With a high discount rate, such as 1, we'll take ANY advancement anytime. However, with a low discount rate, such as .1, we'll indicate that the future rewards are less than current rewards, meaning we want the best results as soon as possible. There'll be testing values the same as the **α** above, with values of 0 to 1 stopping at the first tenth percentile on each side, and each quartile.

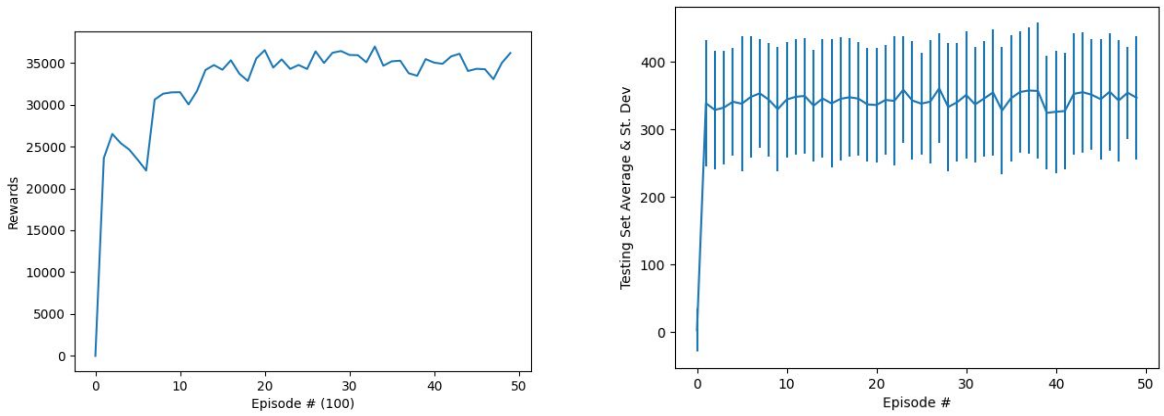
Rewards are another important, configurable factor into what makes a Q-Learning agent operate functionally. So, by punishing Robby for doing an incorrect move, we can see that it'll want to not just linger around, but rather move even more efficiently. To ensure each movement by Robby is an optimal one, I've introduced a penalty of .5 for each step Robby makes.

At the beginning, Robby doesn't really have much information as it begins its journey. So, the first few turns seem to be filled with chaos and error as it explores what the rewards are, and how it can make them. However, what if Robby was **filled** with curiosity, and took a little bit longer to 'mature'. In one of the experiments, Robby faced the same spectrum of changes with epsilon, from 0 to 1 with the first and last 10th percentile and each quartile. With this change, when Robby has low epsilon, it will lean more towards exploitation (with a little chance of exploration). On the flip side, there will be a strong 'curiosity' with Robby, and for the first few epochs it will continue to be purely exploratory. As we compare the epsilon values, we'll be able to see what a difference stochasity makes, and really assess what the 'sweet spot' is for exploration and exploitation.

Lastly, one of the last experiments is simply to see if this representation translates into a bigger board. We have a few limitations in a 10x10 grid: Robby only has 100 possible spaces. With 200 actions each turn, there's a chance Robby can get everything done. However, if Robby were to be in a bigger space, for say a 1000 x 1000 grid, how would it perform? We compare this to the 'ground truth', or the initial version of Robby to verify what the differences are and see how well this problem can scale.

RESULTS & DISCUSSION

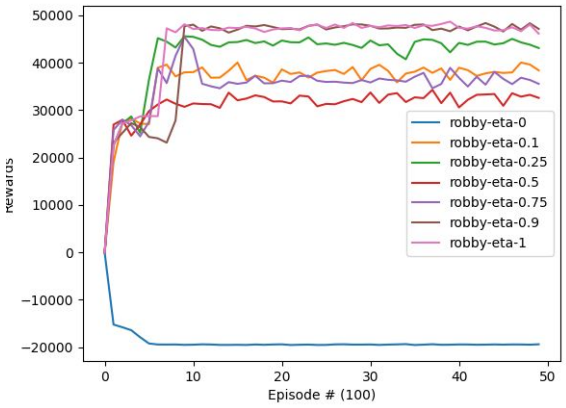
For starters, here is the Ground Truth of the experiment. In this graph, we know that the values are fine-tuned a la Rhodes. These values give the expected behavior of the Q-Learning agent and operate exactly as they need to be. As can be noticed in the left image, which is the training set, the amount of rewards rapidly grows until around 200-300 epochs, in which the agent begins familiarizing itself with the rules of the world. There are occasional dips, as can be observed in Epoch 4-8, but it will stabilize and become consistent after about 20 epochs. For the rest of the 30 epochs in its lifetime, it will hover around the total amount of rewards with some variation. In Robby's terms, he has found an algorithm that works consistently. The deviation occurs from unlucky environments Robby is spawned into, if he ever gets stuck, getting messed up by the stochastic elements, etc. However, in the average and deviation, we're able to see that it tends to hover the same for testing. Because the testing set is scaled down by 100 from the training set, we can see that the averages are the same in the training and testing data when collecting rewards.

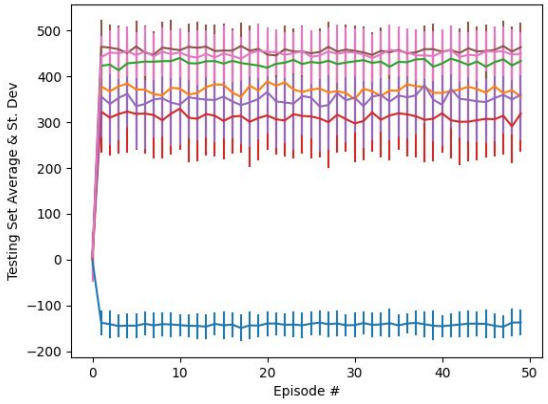


Experimenting with **α**

To clarify, **α** is the 'learning rate' of the Q-Learning agent. In the following text, alpha, **α**, eta, and η are all synonymous, each referring to the 'learning rate'. With the graphs below, we can see the different ranges of results and how they compared. In the training section, which is the image on the left, we're able to see that a learning rate of .9 and 1 performed the best. However, in the testing results, we're able to see that on average, an alpha of .9 performed much better consistently than the learning rate of 1. While there's no large difference in the numbers, we're able to see that having a little bit of acceptance for the previous material helped out, perhaps in the choices that Robby had made. On the opposite side, we see that having NO learning curve resulted in… terrible decisions. With an alpha value of 0, there are extremely poor rewards in both the training and testing data set, just as expected. Essentially, Robby is making the same mistakes, never learning, and never updating its own values. There is a stark difference

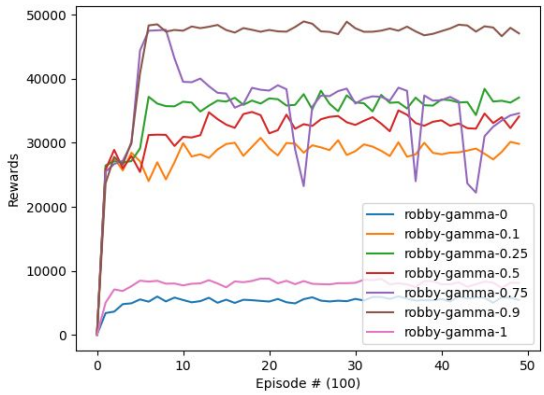
when the learning rate jumps to .1, and we're able to see a significant increase in improvement. It's obvious to see that the rest of the learning rates just fill in the gaps in between, neither posing any significant improvement nor mistakes.

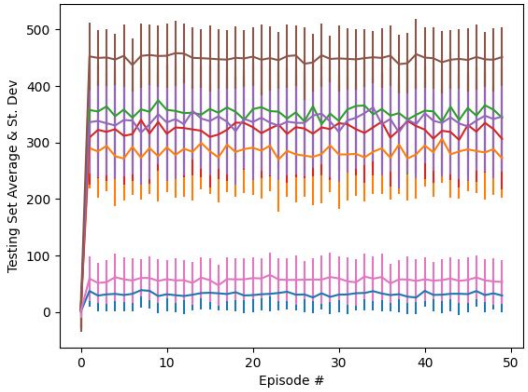




Experimenting with γ

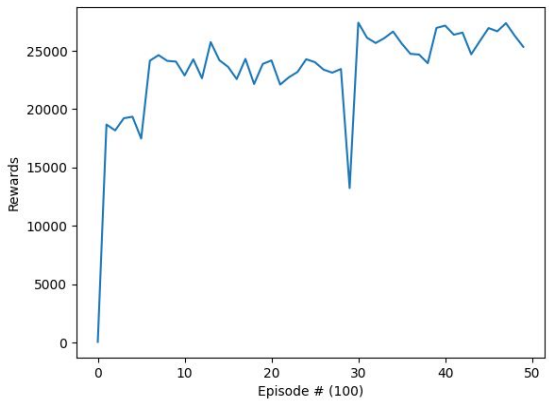
Gamma, the 'discount rate' of Robby, prioritizes that Robby finds the right decisions quicker. The longer that Robby takes to get to its decision, the less of a reward will it get. In the graphs below, it's clear to see there are sweet and sour spots. To begin, a gamma value of 0 and 1 do terribly. It's noticeable that the rewards either being completely discounted cause for very little rewards, yet the rewards being fully accounted for will also cause some discrepancy. To be honest, I'm curious to know why a γ value of 1 causes some problems, but I am willing to bet that it will take its time to get to the solution, causing a long timeline for anything to happen. As soon as there are discounts, results will perform much more consistently, with the exception of a γ value of .75 (which is all over the place). The lesser rewards seem to perform, well, less. The graphs reflect total points, yet with discounts, the lesser discounts will give less total rewards. This is something I honestly didn't consider until just now, yet I realize that there is another way to analyze the rest of the graphs. Thus, the graphs can be analyzed by their consistency, and how soon they reach the values they tend to hover around. A γ value of .9 seems to perform the best, having only minor bumps in its training average, and a very consistent slope in its testing set. γ values for .25, .5, and .1 all have a line they tend to hover around, yet they have a significant more amount of deviation.

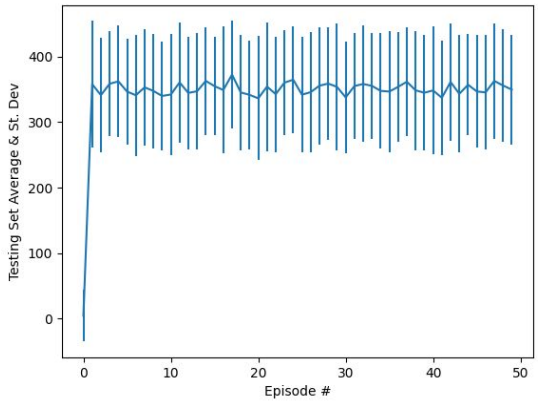




Experimenting with Negative Rewards

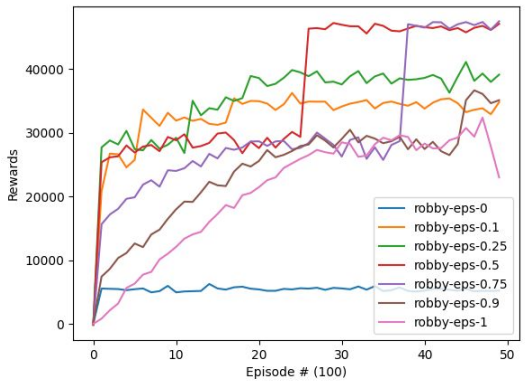
When there are negative rewards, Robby tends to perform a little bit differently in the training. Robby faces many more fluctuations, presumably when it gets stuck. If it finds itself in a space with no cans, it has no other option but to explore outside of its local scope. Which could cause significant dips if there are no cans in the nearby area. There is a big dip around epoch 29, but it stabilizes and becomes even higher in epoch 30. The testing data set itself is super stable and consistent, hovering around 350 rewards. It doesn't seem that the harsh punishments in the testing environment performed any better than the ground truth, so it's hard to say whether the punishment was sufficient to improve performance. There are two options here. Perhaps Robby has already optimized its strategies when there are cans and when there aren't, and the state would have to be represented in a different way for this tactic to become more effective. Alternatively, there could be a stronger punishment enforced.

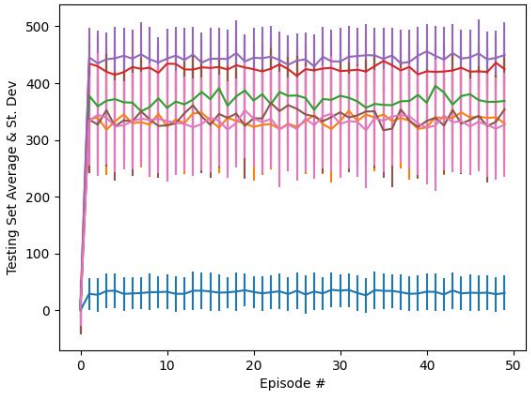




Experimenting with **ε**

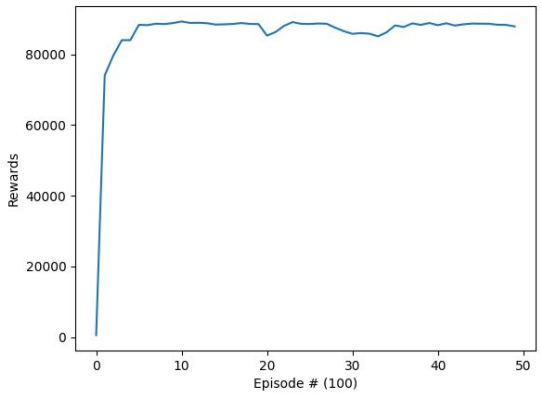
One of the final spectrum-based variables was **ε**. Epsilon (**ε**) causes stochasticity in the agent, causing random actions at any point. There is a timer to cool it down, so every 100 epochs it will go down by .1. Personally, I predicted that results would be significantly random, cooling down to their local maxima, but the results seem to somewhat confirm the theory. As can be seen, each piece is random until it reaches its set epsilon value in epochs, when it cools down to 0, and will stay around the same base. Epsilon value of .5 and .75 performed significantly better than the other results, but epsilon value of .5 reached the maxima much faster. As expected, Epsilon value of 0 reached a maxima that never grew, and it seems that it got stuck at a 'hill' or some sort of local maxima. On the opposite end, epsilon value of 1 appears to be completely random, yet as it cools off it has a significant growth. This is as expected from the decreasingly random behavior installed into Robby. On the testing data set, Epsilon value of .75 and .5 performed, again, super well. It could be that extended period that left it open to explore a new possibility, one that often stumped the rest of the epsilon values.

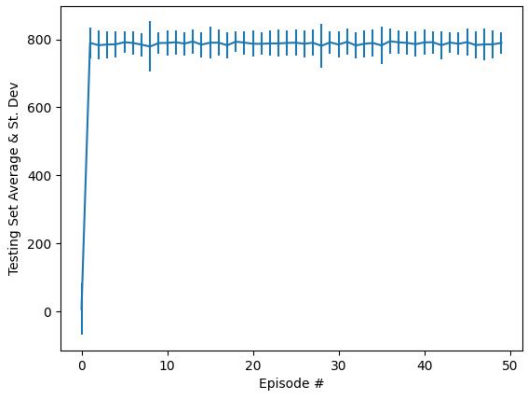




Experimenting with a Large Board

One of the last problems with this challenge was to see if the limited state space using only neighbors would translate into a large board, where you have more possible moves than you have guaranteed actions. Therefore, with a board size of 1000x1000, it's clear to see there is no difference. Robby is able to achieve more points, yet it has very little deviation in terms of getting those points. This occurs in both the training and the testing dataset. One possibility is that the walls called in a few problems with other data points, and with having a lesser chance of getting affected by walls, there are now more possibilities to explore cans.





CONCLUSION

By changing the different variables that make Q-Learning so effective, we'll see how it affects accuracy, performance, and which would be more optimal for this instance of Robby. To confirm, by changing the different variables that make a Q-Learning operational, Robby has variables that allow it to be more accurate and efficient. In this specific problem, having a higher learning rate will help keep reward performance high. Having a gamma result specifically of .9 produces steady results, but other gamma values seem to produce some inconsistencies. There seems to be no change in Robby's behavior when there are negative results, leading to a belief that the system is already optimized with high enough rewards. Experimenting with epsilon also found some new, improved results compared to the ground truth. By having another 15 epochs of stochastic behavior, the system was able to increase the average points by over 100. Lastly, these behaviors do translate well, and in an environment that is 10000 times better, Robby is able to perform just as well, if not better.

All in all, the tweaks to Robby did have a sweet spot. For optimal performance, Robby should have an α of 0.9, a γ of 0.9, an ε of 0.5. Using these values, Robby was able to outscale the ground truth originally listed in the assignment, and find a maximum score higher than the others.