# Reinforcement Learning - An implementation of REINFORCE and Actor-Critic algorithms for Pacman Game

Kavya Chandrika Vempalli, Rishitej Yadav Pallapothu, Thomas Tung, Harry DeCecco
School of Computing and Augmented Intelligence
Arizona State University
Tempe, Arizona, USA

***Abstract*—Reinforcement Learning is a branch of Artificial Intelligence, that trains the agent by rewarding on good actions. It enables agents to learn from the interaction with the environment. The goal is to arrive at an optimal policy. Reinforcement Learning has two types of training methods, Policy iteration and Value Iteration. In this document, we are interested in comparing and contrasting REINFORCE and Actor-Crtic algorithms that are implemented based on policy-based method.**

***Index Terms*—REINFORCE, Actor, Critic, Policy gradient.**

## I. Introduction

REINFORCEMENT learning (RL) is a type of machine learning (ML) approach where software learns to make decisions in order to reach optimal results. It mirrors the trial-and-error learning process observed in humans. Actions that lead towards the desired outcome are reinforced, while those that don't are disregarded. RL algorithms operate on a reward-and-punishment system, learning from the consequences of each action. RL proves to be a valuable tool in enabling artificial intelligence (AI) systems to attain optimal results in unfamiliar scenarios.

Key concepts in reinforcement learning include:

- The agent, which represents the ML algorithm or autonomous system.
- The environment, which constitutes the problem space with its attributes such as variables, boundaries, rules, and permissible actions.
- The action, which denotes the step taken by the RL agent to interact with the environment.
- The state, representing the condition of the environment at a specific moment.
- The reward, which indicates the positive, negative, or neutral value (i.e., reward or punishment) associated with an action.

Reinforcement Learning has two main methods namely, Policy-based and Value-based methods. Value-based Reinforcement Learning is where the agent determines the value for each action it has taken in each state . The agent estimates the value of each action or state in the environment, and based on it, the agent chooses its action. Value means to estimate the total amount of rewards the agent will get following a certain policy. The agent learns a value function at last. The value function that assigns the values to each action is called the state-value function or the value function. The second function that maps a Q value to each state-action pair is called the Q function. V-function: V-function is the most successful way to measure the expectation of returns. It represents an estimate of the agent's expected cumulative return once a defined state s is applied with the subsequent policy . The state-value function V(s) is a function of the expectation of returns once a state s as defined is used with the following policy . The action-value function is the Q function, which is the action value of the agent. It maps from state s and action a to an expectation of returns that vary from state s once action a is played and with the following policy : Q(s, a). Equally, the optimal version of action-value function is achievable from state s by taking action a and thereafter following the optimal policy. The ultimate policy value V* is computed as follows: where it involves taking a maximum across all possible actions that can be taken within each state. Specifically, the objective of the value-based Reinforcement Learning algorithms is to learn the ultimate value function through actual reinforcement based on the experiences of the agent while interacting with the environment.

Policy-based Reinforcement Learning is more concerned with learning a policy that directly maximizes the sum of rewards rather than the action-value function. Rather than learning the value function or the Q-function, policy-based methods learn the optimal policy. The policy-based method starts with an initial policy pi. The policy is then evaluated in the policy evaluation stage and the one with maximum reward given by argmax is chosen as the next policy in the policy improvement stage. The policy is frequently represented by a parameterized function in policy-based RL most often, represented as a neural network that takes as input the current environment's state and hints a probability over the possible actions for the agent at that time. But, in this project linear function approximation is used instead of neural networks with the weighted feature sum to update the weights.That means that the action selection process is stochastic, sampling is involved, and each possible action is sampled from the probability distribution. While in some cases, the policy-based methods have a few significant advantages over the value-based methods, such as a more natural handling of continuous action spaces or the ability to learn stochastic policies, which is often useful if the environment is stochastic or if the multiple

actions are optimal. However, this rule may also have some severe disadvantages, such as the high variance of the gradient estimates or the slower convergence rates.

## II. Technical Approach

The concept of linear function approximation is integral to approximating the Q-function in reinforcement learning scenarios. This method involves representing the Q-function through a linear combination of features and corresponding weights. Both features and weights play crucial roles and are stored for reference during the learning process[1].

A feature vector, denoted as f(s,a), consists of a collection of n * |A| different functions. Here, 'n' represents the number of state features, while '|A|' signifies the number of possible actions. Each function within the feature vector is designed to extract specific feature values associated with a given state-action pair (s,a). This structured approach facilitates the representation of state-action information in a manageable and organized manner. In conjunction with the feature vector, a weight vector denoted as 'w' is utilized. This weight vector has dimensions of n x |A|, aligning with the number of features and actions. Each element within the weight vector corresponds to a specific feature-action pair, determining the significance or contribution of that pair to the overall Q-value approximation [1].

The utilization of approximate Q-functions within the framework of reinforcement learning typically involves two primary steps: Initialization and Update. These steps are pivotal in setting up and refining the linear function approximation model. During the Initialization phase, all weight values are set to an initial state, commonly initialized to 0. This initialization step establishes a starting point for the learning process, ensuring consistency and coherence in subsequent updates. The Update phase is where the actual learning and refinement occur. In this phase, the weights of the linear function approximation model are adjusted based on observed experiences and feedback from the environment. Through iterative updates, the model adapts to changing circumstances and refines its estimations of Q-values for different state-action pairs [1].

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots$$

The REINFORCE Algorithm is a policy gradient algorithm that leverages a parameterized policy, usually implemented as a neural network or using linear function approximation, to directly maximize the expected return .The REINFORCE Algorithm computes the estimated gradient of the expected return concerning the pursuit's policies and then updates this gradient's representation. Consequently, the algorithm samples trajectories at every time step to generate a probabilistic representation of states pursued, actions taken, and the total rewards recorded in the course of the iterations. Using the generated information at every phase of iteration, the REINFORCE employs the normalized gradients of the action log probabilities and total rewards to influence the direction and stride of the policy. The REINFORCE Algorithm is an on-policy approach where the updated state of the policy is a direct result of the gathered data [3].

**Algorithm 1** Reinforce: Monte-Carlo Policy Gradient Algorithm

```
Initialize θ (initial policy)
for each episode do
    Initialize empty episode list
    for each time step until episode ends do
        Observe current state S
        Choose action A using policy with current θ
        Take action A, observe reward R and new state S'
        Store S, A, R in the episode list
    end for
    Calculate return G as the sum of rewards from the current step to the end of the episode
    for each step of the episode do
        Calculate the gradient of the log policy with respect to θ
        Update θ in the direction of the gradient, scaled by step size α and the return G
    end for
end for
```

The Actor-Critic algorithm merges policy-based and value-based learning. The actor and critic are the main players in this method. More theoretically, the actor is an approximator in the form of a policy, while the critic is an approximator in the form of a state value. Specifically, the actor here is usually a parameterized policy . It's a function defined by a set of tuneable parameter values that generates action based on the given state. The critic in this method's implementation is used to estimate the state or the state action and to return a reward to the actor to reduce that difference. The functions that approximate the critic seek to maximize the likelihood or logarithm of the probability of the return function. The goal of learning is for the actor to maximize the expected return over time based on the policy and its own state, as influenced by the critic's estimate of state-action values. On the other hand, the goal of learning in the critic is to approximate value function and to learn to predict value [3].

## III. Results and Analysis

After developing these algorithms and testing them on the Pac-Man domain there are a variety of ways that these algorithms differ especially when discussing different environments and their strengths and weaknesses.

When discussing convergence we can make observations based on Fig. 1 which is the average reward after 100 runs on each episode from 1 to 1000 for each algorithm. An important note is that this compares the convergence when looking at the medium classic layout only. Each algorithm starts at a much lower value for the first few episodes and then has a drastic increase in reward after a few iterations. We can clearly see that all of these algorithms have different values that they converge to, with Actor-Critic being the best. However this is not always the case. If we look to Fig. 2 we see a much different story. Here we can observe that each of these algorithms perform very similarly and in turn converge very similarly. This is because of the simple nature of the small

**Algorithm 2** One-step Actor-Critic

Initialize $\theta$, $w$
**for** Every time step **do**
  Initialize $S$
  $I = 1$
  **while** $S$ is not a terminal state **do**
    $A = \text{policy}(S, \theta)$
    Take action $A$, observe $S'$, $R$
    **if** $S'$ is terminal **then**
      $\delta = R - \text{value}(S, w)$
    **else**
      $\delta = R + \text{discount\_factor} \times \text{value}(S', w) - \text{value}(S, w)$
    **end if**
    $w = w + \text{alpha\_w} \times \delta \times \text{gradient}(\text{value}(S, w))$
    $\theta = \theta + \text{alpha\_theta} \times I \times \delta \times \text{gradient}(\log(\text{policy}(A|S, \theta)))$
    $I = \text{discount\_factor} \times I$
    $S = S'$
  **end while**
**end for**

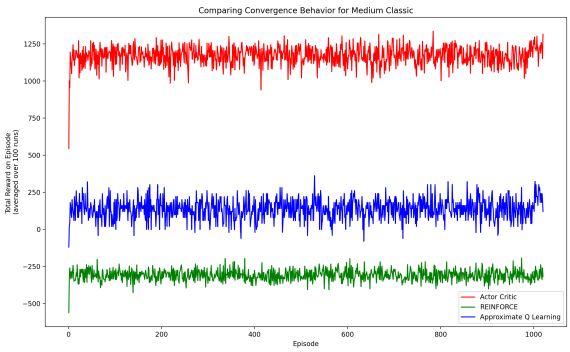


Fig. 1. shows the performance of Actor-Critic, REINFORCE, and Approximate Q-Learning on Medium Classic layout

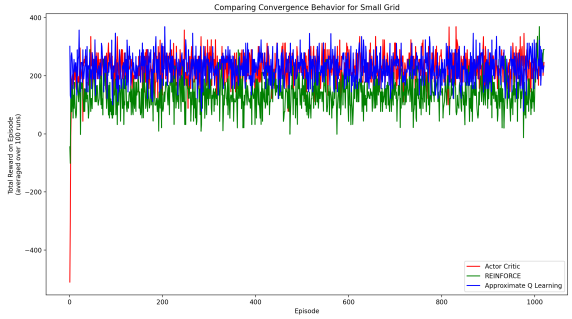


Fig. 2. shows the performance of Actor-Critic, REINFORCE, and Approximate Q-Learning on Small Grid layout

| | layout_11_1 | layout_12_1 | layout_15_1 | layout_15_2 | layout_15_3 | layout_16_1 | layout_18_1 | layout_18_2 | layout_18_3 | layout_19_1 |
|---|---|---|---|---|---|---|---|---|---|---|
| AC | 598.4 | 550.8 | -485.1 | 675.7 | 678.1 | 663.5 | -501 | 807.4 | 850 | 740.9 |
| R | -335.6 | -453.3 | -499.7 | -443.1 | 659.3 | 772.2 | -501 | -384.6 | -433.4 | -446.4 |
| Q | 599.8 | 669.9 | 462.3 | 777.6 | 802.2 | 775.3 | -501 | 686.9 | 864 | 836.1 |

Fig. 3. shows the average score of 10 runs after 500 episodes for each generated layout

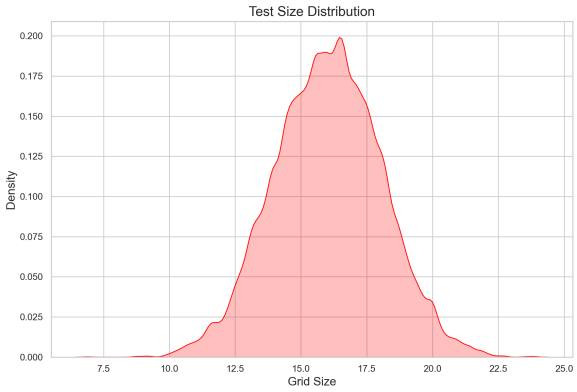


Fig. 4. shows distribution of sizes of the randomized layouts to prove normality

grid. There is not a lot of strategy involved here and each of these algorithms is able to find that strategy at a very quick rate.

For both medium and small layouts these converges produce a large amount of variance. This is because of the nature of the Pac-Man domain. In these games there is a huge reward for either completing the game with a win or a lose. This allows for large variance as these different values has a large difference between them which then leads to averages of episodes being much different based on the number of wins that were recorded.

These convergence patterns leads us to the fact that given different environments each of these algorithms will perform very differently and some environments are more suited to each algorithm like how Actor-Critic is more suited to the medium classic layout.

Our data used for our 2-sample T-Test is an average score between 10 runs after 500 episodes of training on 10 different randomly generated layouts with all three algorithms. This is seen in Fig. 3. What is notable about this data is we can clearly see that 500 episodes is not enough time for the REINFORCE algorithm to be able to create an optimal policy as seven out of nine average scores are negative. Also note that we are only considering nine scores ass layout_18_1 was a randomly generated layout that placed 2 ghosts surrounding Pac-Man which made the game end after the first move.

To perform a 2-sample T-Test both samples need to be normalized. However, because of the nature of the Pac-Man domain as well as the different environments that it is tested on there is no way to get a normalized sample of scores. We have tried performing Box-Cox Transformation as well as Log Transformation and both have failed to get our samples to be normalized. Even though the samples of this 2-sample T-Test are not normalized we will still run the tests because of the normality of the layouts that we are testing on as scene in Fig. 4. This will lead to not as concrete results in our overall assessment however it will still provide us with valuable feedback to observe the benefits of one algorithm

over another.

TABLE I
2-SAMPLE T-TEST ON EACH PAIR OF ALGORITHMS

| Algorithms | T-statistic | P-value | Hypothesis Status |
|---|---|---|---|
| Actor-Critic, REINFORCE | 2.976 | 0.008 | Reject |
| Actor-Critic, Q-Learning | -0.678 | 0.506 | Fail to Reject |
| REINFORCE, Q-Learning | -4.005 | 0.001 | Reject |

Based on the values given in our table we can clearly see that when comparing REINFORCE with both Actor-Critic and Approximate Q-Learning there is clearly enough evidence to reject the null hypothesis and deduce that REINFORCE is clearly the worse algorithm in this case. It average score is much lower then both Actor-Critic and Approximate Q-Learning.

REINFORCE algorithm is a very strong learning model as unlike Approximate Q-learning it still takes the opportunity to learn no matter what episode it is at. Constantly tuning itself until it can traverse. However, this leads to REINFORCE being very costly while training. Since it still takes inefficient paths even after the nth episode, the step size and how much training can drastically effect the efficiency of REINFORCE in certain environments. There are many scenarios we observed when making an analysis on the training of REINFORCE where the agent would continuously cycle between two locations because it does not see any benefit from moving from its original position. This leads to REINFORCE being a slow learner that can have a lot of variance compared to other learning models. So while REINFORCE is a poor agent when only given 500 episodes it can eventually become a very reliable agent given more training.

Looking at the other two algorithms and their resulting T-Test we see that Approximate Q-Learning when compared to Actor-Critic, has a much higher P-value and therefor we fail to reject the null hypothesis meaning that these algorithms, in these specific conditions, cannot discern a clear winner and therefore both provide lots of different benefits.

Like stated before the normality of the samples is negative and therefore these tests can not provide us with as concrete of an answer that we would like. However, because of the normality of the test layouts as well as the visual inspection of the performance of each of the algorithms and our P-values being well above and below the threshold of our alpha set at 0.05, there is evidence to suggest that these results are conclusive.

Computation time was also a lesser observation noted when performing these analysis on the agents in different environments. When looking at layouts with a larger environments we see a gradual shift in computation time when looking at REINFORCE and Actor-Critic as a larger environment leads to longer games time needed to win. This is the same increase we see in algorithms like Approximate Q-Learning.

Throughout all of these statistical tests it makes it very apparent that each of these algorithms are suited for different circumstances. We have seen that REINFORCE needs many episodes in order to produce an optimal policy and we have also scene how quickly actor-critic can learn an optimal policy given less episodes. We also have taken note that with smaller environment sizes performances tend to converge together as optimal policies become a lot easier to find with fewer episodes. Our t-tests support this hypothesis as they state Actor-Critic is the clear favored algorithm when compared to REINFORCE based on only 500 episodes of training.

## IV. CONCLUSION

We have understood that the REINFORCE only takes the policy gradient method, while actor-critic combines the advantages of both value-based and policy-based methods. In our analysis we have clearly outlined the different benefits to each of the algorithms and how with a lower training size Actor-Critic handily outshines REINFORCE. Our results clearly indicate that actor-critic outperforms REINFORCE. We have also seen that given different environments the performance of each of these algorithms can change dramatically. An assumption is the alpha value for our algorithms is one of the major factors when it comes to performance in different environments but in order to derive this conclusion more research would need to be done in a different setting.

## REFERENCES


[1] Zhang, H., Mao, Y., Wang, B., He, S., Xu, Y., & Ji, X. (Year). In-Sample Actor Critic for Offline Reinforcement Learning. *Journal Name*, *Volume*(Issue), Page Range. Tsinghua University, Dalian University of Technology.

[2] Lu, J., Wu, X., Cao, S., Wang, X., & Yu, H. (Year). An Implementation of Actor-Critic Algorithm on Spiking Neural Network Using Temporal Coding Method. *Journal Name*, *Volume*(Issue), Page Range. College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, China.

[3] Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (Year). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Journal Name*, *Volume*(Issue), Page Range. AT&T Labs Research, 180 Park Avenue, Florham Park, NJ 07932.

[4] , Bhatnagar, S. (Year). The Reinforce Policy Gradient Algorithm Revisited. *Journal Name*, *Volume*(Issue), Page Range. Department of Computer Science and Automation, Indian Institute of Science, Bengaluru 560012, India.

[5] *Text book* Sutton, R. S., & Barto, A. G. (2014, 2015). Reinforcement Learning: An Introduction (second edition)