A Deep Dive into On-Policy Reinforcement Learning (SARSA)

This document mainly focuses on SARSA algorithm, how it works and how learning rate and discount factor effects its performance

Let's begin with what is machine learning

Machine learning is a process where we develop and improve algorithms to perform particular tasks or make some predictions using data. Here the algorithms learn from data and perform tasks for which they have been programmed for

There are different types in machine learning depending on the type of tasks that you want to perform

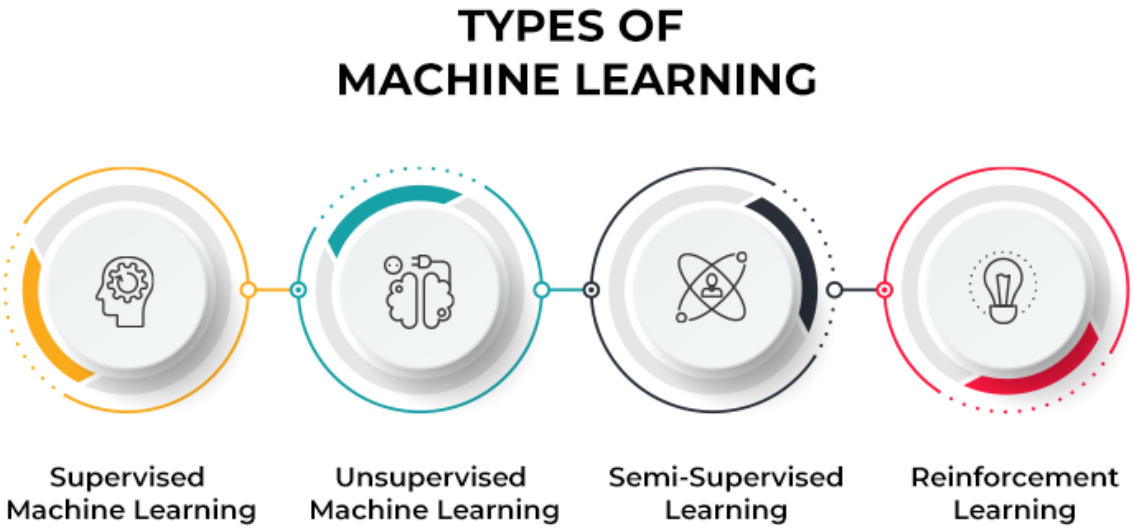

Figure 1 Shows different types of machine learning

In this tutorial we will be mainly focusing on Reinforcement learning

Reinforcement learning in simple words is learning from mistakes like a trial and error, here a machine makes decisions in order to achieve a specified goal using rewards and penalties paradigm, they learn

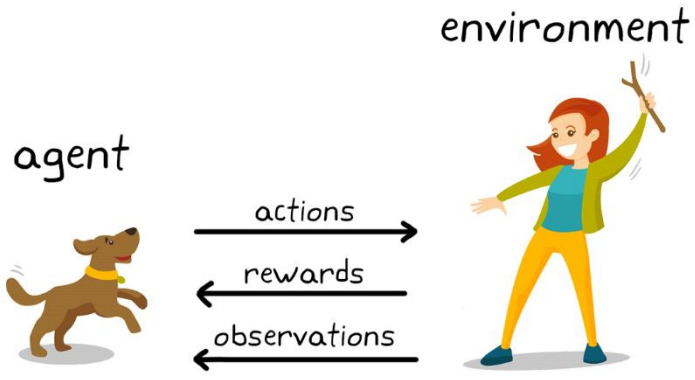

Figure 2 Reinforcement learning work flow example using dog and owner

from each step using reward/penalty they receive and takes their next step to achieve their ultimate goal or reward.

The algorithms are also capable of delayed gratification. The best overall strategy may require short-term sacrifices, so the best approach they discover may include some punishments or backtracking along the way.

Let's see few main components of reinforcement learning using real-world example:

A Cat owner decides to train his cat to use the litter box

Agent here is our cat which is trying to learn using litter box

Environment here represents a house that contains various objects, such as a sofa, a mat, and a litter box.

Actions are cat actions like using litter box or using sofa or using mat or floor

Reward, cat gets some treat if it uses litter box

Penalty, cat gets scoldings or no treat if it doesn't use litter box

Goal is to make cat use the litter box always

Initially cat uses different things and it learns that it gets a reward for using the litter box and over time it learns to only use the litter box, so here we can see that the owner successfully trained his cat to use litter box and probably they didn't even know that they are performing a reinforcement learning.

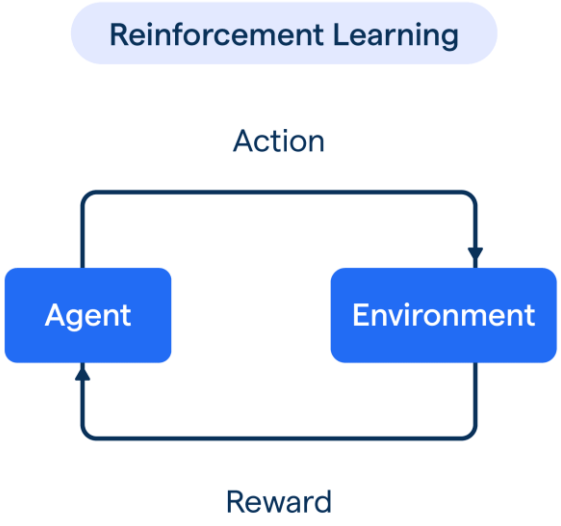

Figure 4 Reinforcement learning paradigm using its parameters

Figure 3 Gif of cat using litter box after training

But if you look up our tutorial name it says something like On-Policy Reinforcement Learning so what's this policy

A policy defines the behaviour of an agent as it determines what actions the agent should take in different states within an environment

There two policies in which we should be familiar with before learning about SARSA

ON Policy and OFF policy

OFF-Policy: The agent learns by observing or considering actions from a different policy than the one it is optimizing. For instance, the agent can use data from a past strategy or even mimic actions from others (e.g., Q-Learning)

On-Policy: The agent learns by following the same policy it is trying to optimize. For example, it uses the actions it chooses based on its current strategy (e.g., SARSA)

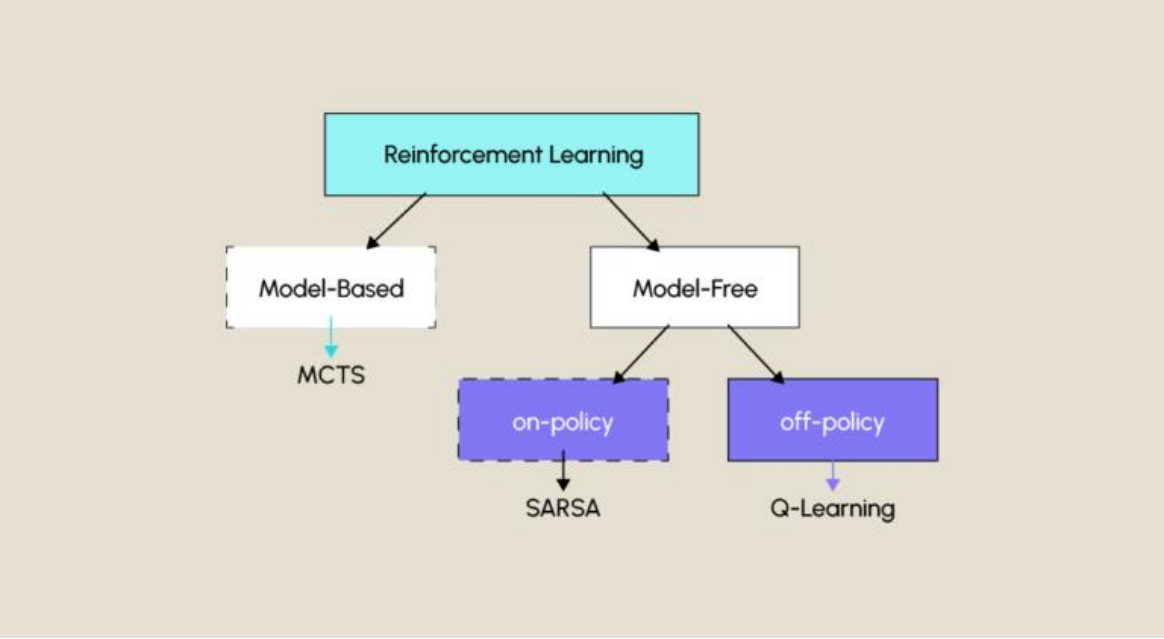

Figure 5 Chart showing different types of reinforcements learning algorithms

Now, let's deep dive into ON Policy Reinforcement learning - SARSA:

SARSA stands for State action reward state action, which represents learning process of the algorithm

In this algorithm, Agents the current state (S), an action (A) is taken and the agent gets a reward (R), and ends up in the next state (S1), and takes action (A1) in S1, that's why its named SARSA (S, A, R, S1, A1)

As we know it works using trial and error method the agent in a state takes an action and gets rewards and makes its next move based on the new state and reward it received, this flows until the agent reaches the optimal result which is its maximum reward and as it is an on-policy technique it works iteratively with the initial policy, refining it based on the rewards obtained.

In an on-policy algorithm, the behaviour policy is the same as the target policy. Behaviour policy refers to the policy the agent follows to explore the environment. Target policy means the policy the agent follows to reach the target.

Including the components that we saw in the cat example we need to understand few more to see how SARSA algorithm works

Q Values: The estimated cumulative reward awarded to the agent for acting in a particular state Q(s,a)

Q Table: The Q Table keeps track of the Q Values of the states and actions to take the next optimal decision

Epsilon greedy: We use this strategy in SARSA to balance the algorithm for **exploration and exploitation**

Exploration is a process where agent tries new actions to learn more about the environment

Exploitation is where the agent chooses the next step with its knowledge that it has learnt already and takes the best step possible

Epsilon refers to the probability of choosing to explore, exploits most of the time with a small chance of exploring. While exploring it selects the random next action but while exploiting it selects the action with the highest Q value

We can choose at what rate our algorithm should explore, with probability ϵ it chooses to exploit and with probability 1- ϵ it chooses to explore the environment

If ϵ=0.1, The agent will:

Choose the greedy action (with the highest Q-value) 90% of the time.

Choose a random action 10% of the time.

We should be careful while selecting our epsilon value smaller value may lead to agent exploring the environment very little which may lead to sub optimal learning on the other hand larger epsilon value can make agent explore lot which delays learning as it spends more time trying random actions. So, there's always a trade-off between exploration and exploitation

Discount factor (γ): This indicates the importance of future rewards in the learning process, lower value mainly focusses on the immediate results and higher value gives more importance to future rewards, it is a crucial parameter that determines the present value of future rewards.

learning rate (α): It determines how much the agent updates its Q-values (state-action value estimates) during each step. The learning rate determines how much weight the new information has with the current estimate. A smaller α ensures stable and incremental updates and a larger α makes the updates faster but might lead to instability.

Now as we know the all-important factors and components let see how they come in order with SARSA update rule, which is the rule on what basis the Q values are updated in algorithm with changing states and actions in the learning process

$$Q(s,a) = Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$$

Here it updates the Q-value of the current state-action pair by considering the immediate reward plus the discounted Q-value of the next state-action pair.

- **Q(s,a):** The Q Value of the present state-action pair
- **Alpha(α):** The learning rate of the algorithm
- **r:** The reward expected at the state
- **Gamma(γ):** The discount factor which is used to provide weightage to the future rewards
- **Q(s',a'):** The Q Value of the next state-action pair

Now let's understand SARSA algorithm in more clear way by implementing it in a simple environment

Here we use SARSA algorithm to train agent to navigate the grid from its starting position to the goal.

We create a 5*5 grid which will be our environment where agent interacts and learns, its main goal is to reach the (5,5) position while starting from (0,0) by navigating through the grid

For more complexity we will add some pits in the grid where agent can't take a path through those pits

```
import numpy as np
import random

# Parameters required for SARSA algorithm

alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
epsilon = 0.1  # Exploration rate
num_episodes = 1000 # Number of episodes
grid_size = 5  # Size of the grid

actions = ['up', 'down', 'left', 'right'] # Actions that agent can take

# Map actions to movements

action_map = {
    'up': (-1, 0),
    'down': (1, 0),
    'left': (0, -1),
    'right': (0, 1)
}

# Lets define our environment here

pits = [(2,2), (3,1),(1,4),(4,2)]  # Pits in the grid that agent cant pass through them
goal = (grid_size - 1, grid_size - 1)  # Goal position which our agent should reach
```

Figure 6 Code cell for defining SARSA parameters and environment

We will first define the parameters that we have learned before in this tutorial as these are important components of SARSA algorithm and we define the SARSA environment and what actions the agent can take in the environment

```python
def take_action(state, action): # Movement of the agent in grid environment
    x, y = state
    dx, dy = action_map[action]
    x = max(0, min(grid_size - 1, x + dx))
    y = max(0, min(grid_size - 1, y + dy))
    next_state = (x, y)

    # Rewards
    if next_state in pits:
        reward = -1  # Negative reward for falling into a pit
        next_state = state  # The agent stays in its current state
    elif next_state == goal:
        reward = 1  # Positive reward for reaching the goal
    else:
        reward = -0.1  # Small negative reward for regular steps

    return next_state, reward

# Initialize Q-table to store Q values

Q = {(x, y): {a: 0 for a in actions} for x in range(grid_size) for y in range(grid_size)}
```

Figure 7 Code cell to specify SARSA agent flow/movement in environment and rewards

Then we shall how agent moves in the environment here the current state of the agent in grid is specified by pair (x,y) x represents the row index and y is our column index

Action is what action agent is taking at that state like moving up, down, right or left as we defined these actions already

The action_map maps the action (up/down/right/left) to a movement vector (dx, dy) and x,y are updated based on the action chosen

And then we specify the rewards of the agent as feedback to the agent on its learnt step

I specified strong negative reward if the agent takes path through the pit so that it can avoid it and the maximum reward for reaching the goal. It also gets a small penalty for taking regular steps to encourage it to learn different paths and choose most optimal path and a Q table has been initialised to store estimated cumulative rewards awarded to the agent for acting in a particular state

As now we have set up the environment, required parameters and the Q table we will now try to implement SARSA

```
# Now lets implement SARSA Algorithm

paths = []  # Store the agent's path per episode
rewards_per_episode = []  # Cumulative rewards for each episode

for episode in range(num_episodes):
    state = (0, 0)  # The Starting point of the agent
    path = [state]  # Track the agent's path
    total_reward = 0  # Initialize the total reward for this episode
    action = random.choice(actions) if random.random() < epsilon else max(Q[state], key=Q[state].get)

    while state != goal:  # Until reaching the goal
        next_state, reward = take_action(state, action)
        next_action = random.choice(actions) if random.random() < epsilon else max(Q[next_state], key=Q[next_state].get)

        # Update Q-value
        Q[state][action] += alpha * (
            reward + gamma * Q[next_state][next_action] - Q[state][action]
        )

        # Transition
        state = next_state
        action = next_action
        path.append(state)
        total_reward += reward

    paths.append(path)
    rewards_per_episode.append(total_reward)  # Track total reward for this episode
```

Figure 8 Code cell for implementing SARSA algorithm

The agent here tries to complete each episode trying to reach the goal, the path always starts from the starting point of the agent which we defined as position (0,0), the agent at first uses epsilon greedy to take its first action and then follows to select the action with highest Q value otherwise selecting the greedy action once in a while depending on our epsilon value. (Exploration vs Exploitation)

Agent follows this until it reaches its goal by keeping moving to next state with the action it took and reward it got for the action

The SARSA formula updates the Q value for each state-action Q(s,a) which will improve the agent's decision making.

The difference (reward + gamma * Q[next_state][next_action] - Q[state][action]) is the **temporal difference error**, representing how much the current Q-value should be adjusted.

This continues until the agent reaches its goal and then comes the next episode, after the several episodes agents learns the optimal path to reach its goal.

Let’s visualise the agents learned path from the environment, you can see below how agent learned to reach the goal avoiding the pits the grid

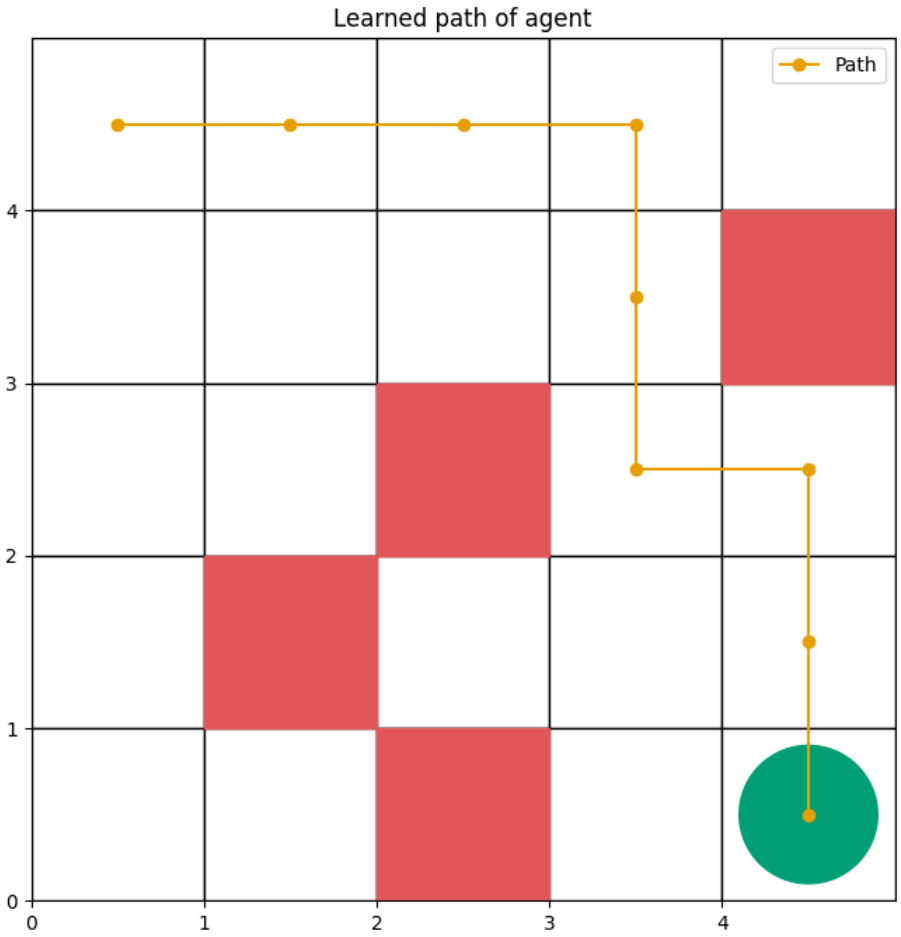

Figure 9 Plot showing agent's learnt path after training using SARSA

Let’s visualise how agent’s cumulative rewards improve over time showing the learnings of the agent

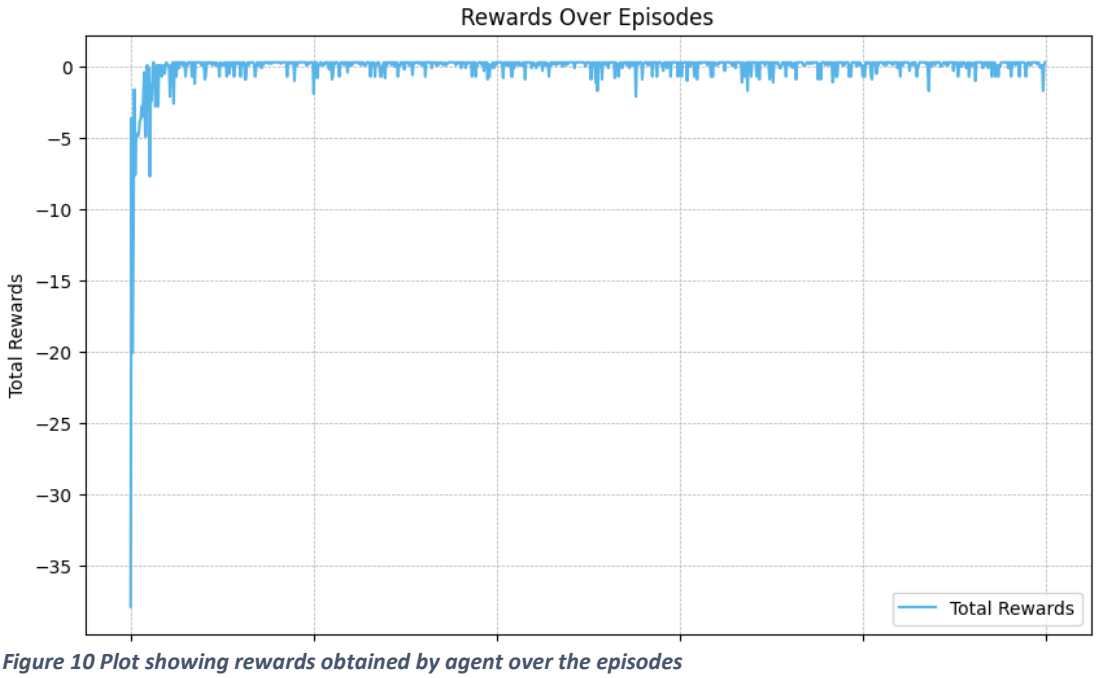

Figure 10 Plot showing rewards obtained by agent over the episodes

I will upload the code for these visualisations along with SARSA implementation in a Jupyter file so you can clearly see how this has been done and make changes for trying out different things

We also talked about how the learning rate and discount factor can affect SARSA functioning we can see those clearly in the plots below as I tried the algorithm with different values to compare their rewards.

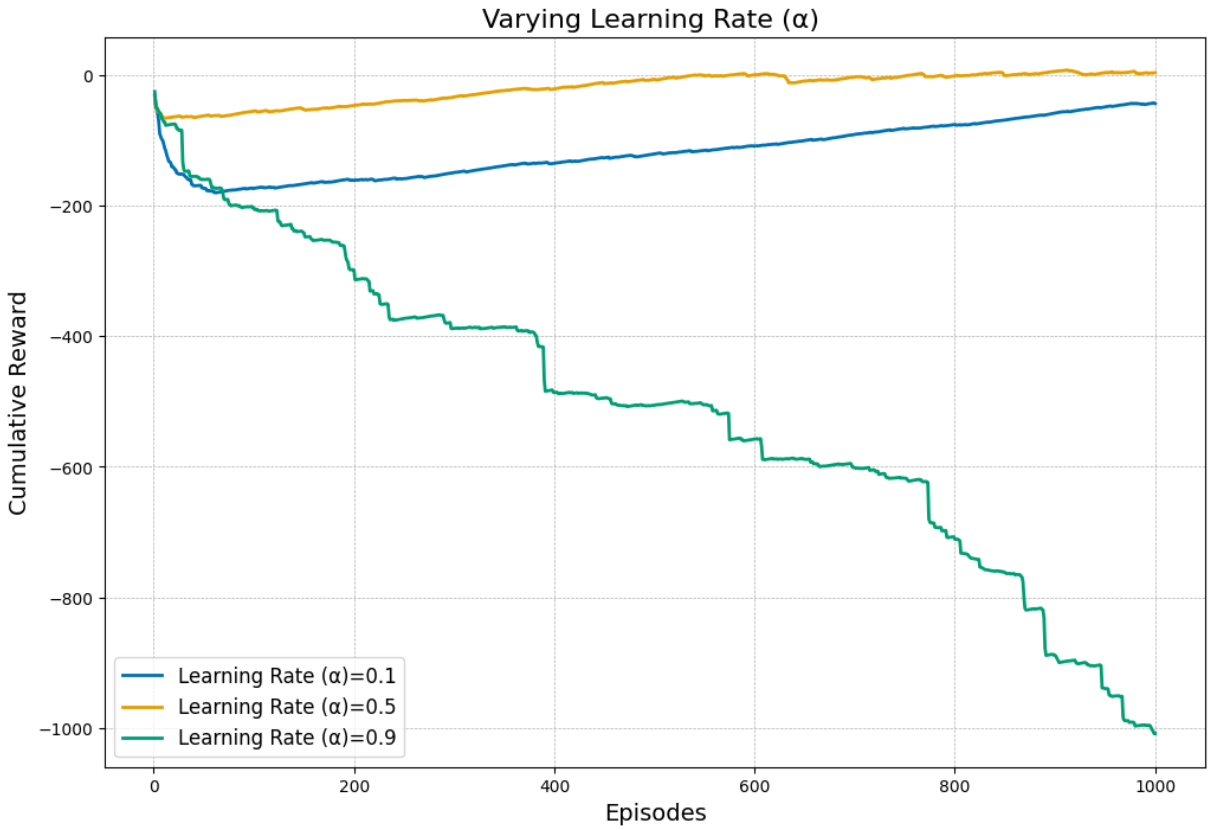

Figure 11 Plot showing how different learning rates effect the SARSA algorithm

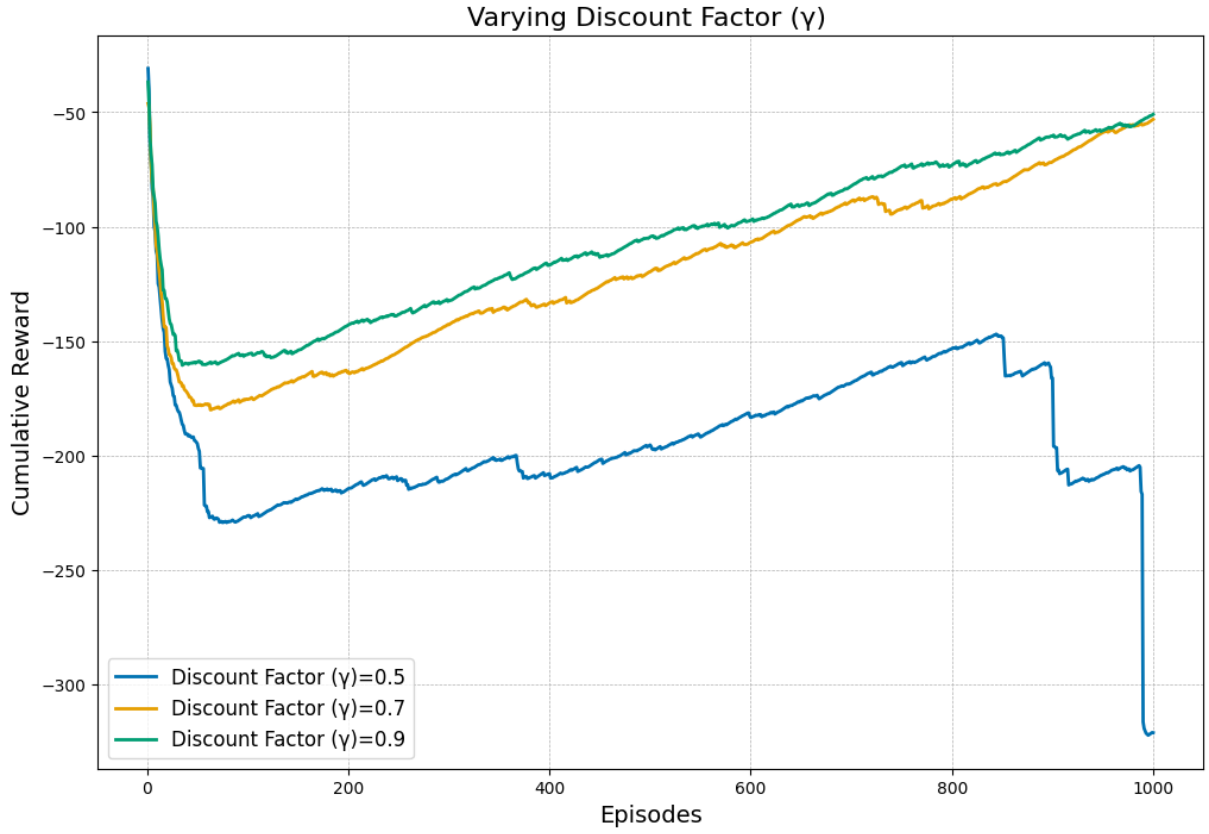

Figure 12 Plot showing how different discount factors affect the SARSA algorithm

Here you can observe that higher learning rate can make the agent performance unstable over time and how lower discount factor can make agent to not reach its optimal path which has high reward. So, selecting the values of these parameters according to your data and environment is very important

There are many reinforcement learning environments which are available that you can work on, most popular one is OpenAI Gym which provides different reinforcement learning environments

For Example, Taxi problem in OpenAI Gym where taxi navigates by picking up customers and dropping them at destination. You can work on these environments with what you have learned in this tutorial to explore different things in SARSA. Also, try tweaking parameters such as learning rate, discount factor to observe different behaviours.

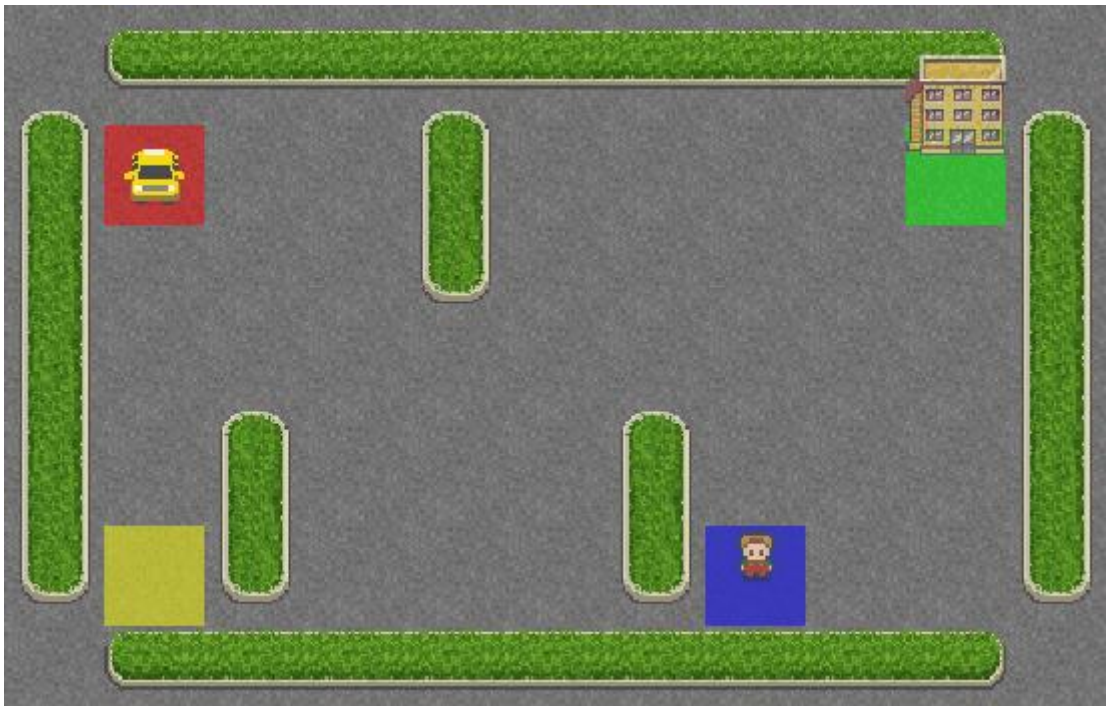

Figure 13 Taxi Problem example from Open AI Gym environment

SARSA is one of the very most important algorithms in reinforcement learning as it's an On-policy algorithm it can easily adapt to any changing environment, if there are any changes made to the environment SARSA will adapt the policy according to it changes with real time learning.

Few Applications of SARSA:

Game Playing (Train agents to play video gamers)

Finance and Trading (SARSA can assist in dynamic asset allocation by learning optimal investment strategies based on market conditions)

Healthcare (SARSA is used to develop adaptive strategies for medical treatments, adjusting to patient responses)

Education (SARSA helps in adaptive learning platforms, creating personalized study plans based on student performance)

CLICK HERE FOR MY GITHUB REPOSITORY

References and further readings:

1. Dong, H., Zhao, D., Huang, D., Yan, K. and Ren, W., 2024, May. SARSA Reinforcement Learning Based Path Planning of Unmanned Aerial Vehicles. In *2024 IEEE 13th Data Driven Control and Learning Systems Conference (DDCLS)* (pp. 1099-1105). IEEE

2. Qiang, W. and Zhongli, Z., 2011, August. Reinforcement learning model, algorithms and its application. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)* (pp. 1143-1146). IEEE

3. https://www.datacamp.com/tutorial/sarsa-reinforcement-learning-algorithm-in-python

4. https://builtin.com/machine-learning/sarsa

5. Reinforcement Learning: An Introduction Richard S. Sutton and Andrew G. Barto