Assignment 3
CS 451: Computational Intelligence

Ali Asghar Yousuf | Muhammad Murtaza

April 5, 2023

1 Reinforcement Learning Agent

1.1 Problem Statement

The Gridworld problem is a classic reinforcement learning problem where an agent operates in a two-dimensional grid-like environment, and its goal is to navigate to a goal state while avoiding obstacles and maximizing rewards. The agent has a set of actions available at each grid cell, which can move it to a neighboring cell. The environment is stochastic, meaning that the agent's actions may not always have the intended effect, and the agent receives a reward or penalty depending on the action taken and the resulting state. The problem statement for Gridworld in reinforcement learning would be to find an optimal policy for the agent to navigate the environment, maximize rewards, and reach the goal state while minimizing the number of steps taken and avoiding obstacles.

1.2 Solution Approach

We have make use of Reinforcement Learning to optimize this problem making use of reward based learning. We will learn the value function using via episodic learning. The actions will be selected by the Boltzmann distribution.

1.2.1 Initialization of the Grid

The first step is to initialize a grid of size n x n where n is an integer. Red terminals (obstacles) are placed around the grid and Green Terminals (rewards) are placed around the grid. γ and α are initially set to 0.8. You can change it to test for different runs.

1.2.2 Agent Position

Agent Position is selected in the grid randomly wherever there is any empty grid cell. In each episode, the agent starts with a new position.

1.2.3 Temporal Difference Learning

In our algorithm, we made use of TD to update the q values in each iteration. In TD learning, the agent learns directly from its experience in the environment by updating the value estimates of each state-action pair after each time step. This allows the agent to learn online and adapt to changes in the environment in real-time.

By using the TD equation to update the value estimates of the previous state, the agent can learn the optimal policy through trial and error. Over time, the agent learns to associate certain states and actions with higher rewards, and it uses this knowledge to make decisions that maximize the total reward over time.

Following equation was used to update the values:

$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma(V(S_t + 1)) - V(S_t))$

When the agent moves from one state to another, the value for that cell is updated using the equation above.

1.2.4 Boltzmann Distribution

$$P(a|s) = \frac{e^{Q(s,a)/k}}{\sum_b e^{Q(s,b)/k}} \tag{1}$$

In the Gridworld Reinforcement Learning problem, the Boltzmann distribution plays a crucial role in selecting actions for the agent. The Boltzmann distribution is a probabilistic policy that selects actions based on their estimated values and a temperature parameter. The temperature parameter controls the degree of exploration versus exploitation, where a higher temperature results in more exploration and a lower temperature results in more exploitation.

The Boltzmann distribution provides a way to balance exploration and exploitation during the learning process. Initially, the agent explores the environment by randomly selecting actions, and the temperature parameter is set high to encourage exploration. As the agent learns more about the environment, the temperature parameter is gradually reduced to favor exploitation of the learned information.

1.3 Results

For the following parameters we observed the following results:

- Grid Size = 10×10
- Episodes = 100
- $\alpha = 0.8$
- $\gamma = 0.8$

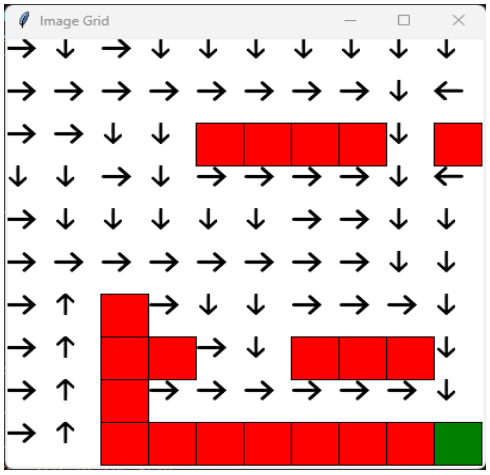

Figure 1: First Run

- Grid Size = 25×25
- Episodes = 500
- $\alpha = 0.8$
- $\gamma = 0.8$

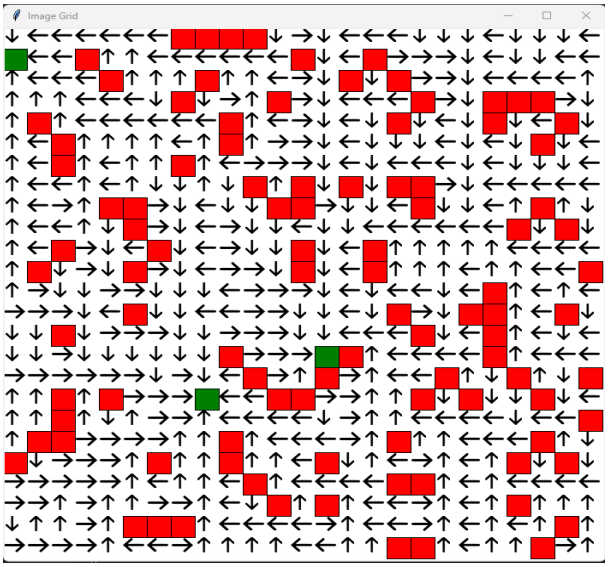

Figure 2: Second Run

- Grid Size = 15×15
- Episodes = 250
- $\alpha = 0.2$
- $\gamma = 0.6$

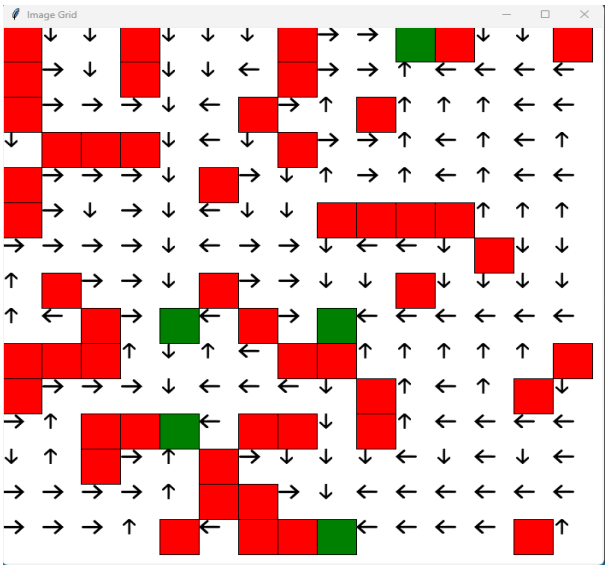

Figure 3: Third Run

To run the code, kindly refer to the `main.py` file. Here you can change the grid size. When changing the grid size, please change the number of red terminals and green terminals according to it.

Note: Sometimes the algorithm gets trapped in opposite states selecting each other. If this happens the episode will get stuck. The algorithm needs to be run again in this case.

2 Self Organizing Maps

2.1 Problem Statement

The Self Organizing Maps (SOM) is a type of Artificial Neural Network (ANN) that is used for unsupervised learning of high-dimensional data. It is a type of competitive learning, where the network learns by competing with its own copies to produce a set of outputs that best match the input data. The SOM is a two-layer neural network that consists of an input layer and a competitive layer. The input layer consists of a set of neurons that represent the input data, and the competitive layer consists of a set of neurons that represent the output data. The SOM is trained by presenting the network with a set of input patterns, and the network learns to produce an output pattern that best matches the input pattern.

2.2 Solution Approach

2.2.1 Data Preprocessing

Before applying the SOM algorithm, it is important to preprocess the data. This involves normalizing the data and removing any outliers or missing values. This step ensures that the SOM can effectively cluster the data.

2.2.2 Initialization

The SOM algorithm requires an initial set of weights to start with. These weights are randomly initialized and represent the initial positions of the neurons in the SOM.

2.2.3 Neighborhood Function

The SOM algorithm uses a neighborhood function to determine the extent to which the weights of neighboring neurons are updated during training. The neighborhood function decreases over time as the algorithm converges towards a stable state.

2.2.4 Learning Rate

The learning rate determines how much the weights of the neurons are updated during training. The learning rate decreases over time as the algorithm converges towards a stable state.

2.2.5 Training

During training, the SOM algorithm iteratively updates the weights of the neurons to minimize the distance between the input data and the weights. This process continues until the weights converge to a stable state.

2.2.6 Visualization

Once the SOM has been trained, it can be visualized using a 2D or 3D map. The map represents the clusters of data in a visually intuitive way.

2.2.7 Interpretation

Finally, the clusters can be interpreted and analyzed to gain insights into the underlying patterns and relationships in the data.