SAI PRANITH BHAGAVATULA (24AT61R03)

REINFORCEMENT LEARNING

TERM PROJECT PHASE – I

Submitted by *Sai Pranith Bhagavatula* to *Dr. Aritra Hazra*

TABLE OF CONTENTS

PROBLEM

STATEMENT

To create an agent using Reinforcement Learning algorithms that runs around in a 10x10 grid world and survive from the chasing tagger for as long as possible.

CONSTRAINTS

a. The Tagger is fast. It can move two boxes at a time-step in 4-directions – Up, Down, Left, Right);
b. The Runner is slow, but agile. It can move in 8-directions, but one box at a time-step.
c. The Runner is said to be caught when both the Tagger and Runner land on the same box (grid).

REQUIREMENTS

Grid World: This is a grid of size 10x10.

Tagger: It is initially at the bottom-right corner of the grid world. The behaviour of the Tagger is simple. It calculates the Euclidean distance between the Tagger and the Runner at any given time and makes a move in the respective direction.

Runner: It is initially at the top-left corner of the grid world. The behaviour of the Runner is dependent on the RL algorithm applied. In this project, the Runner incorporates Value Iteration and Temporal Difference learning algorithms to observe and learn from the environment, the best state to be in at a time '*t*'.

Environment: The environment is hardcoded to run for a specific number of steps (100 steps), where the Runner is rewarded +1 for survival at every step and -100 (minus hundred) when terminated. An episode is said to be terminated when both the Tagger and the Runner land on the same grid, in which case, the environment gets reset (with the number of remaining steps unchanged). The Runner and the Tagger start over from their respective original states.

IMPLEMENTATION

grid_world_env.py demonstrates the running of the environment without any learning from the agent, i.e. the Runner. In this setup, the Runner chooses its actions based on a random sample incorporating no learning algorithms.

TAGGER LOGIC

The tagger logic is defined by a greedy strategy to reach the runner's current position.

For example, let us assume that the Tagger's Position is [x1, y1]; and the Runner's Position is [x2, y2].

Tagger's logic compares the absolute differences between x1 and x2 with y1 and y2 to see which is greater, and chooses to move along the axis of the greater difference.

Further, the difference being **negative** determines the tagger's direction as right (if x-axis) and down (if y-axis), and the difference being **positive** determines that the tagger has to move left (if x-axis) and up (if y-axis).

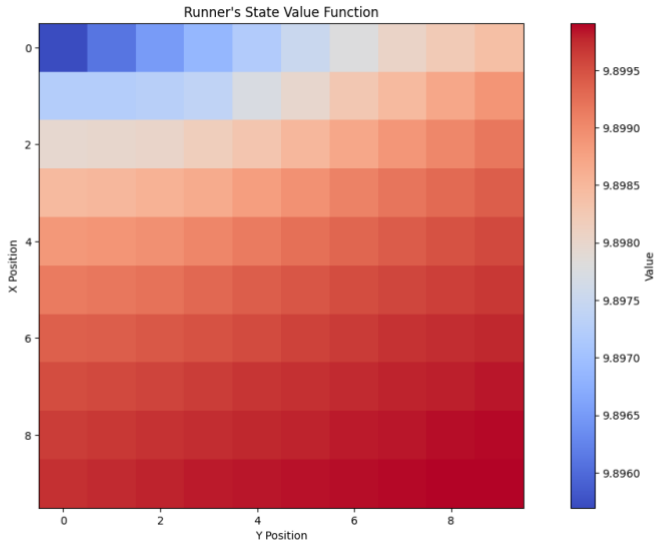
VALUE ITERATION

grid_world_vi.py aims to implement a Dynamic Programming approach of a learning algorithm, where the agent (Runner) adapts to the environment, by iteratively updating the value of each state based on **Bellman Equation** until the values converge.

In short, using this algorithm, the agent chooses the actions which would guarantee maximized rewards in the long run.

BELLMAN EQUATION

Below is the heatmap of value functions of the states thus generated:



Observation

After the convergence to the true-values, it can be observed that the value function is heavily flowing towards the **bottom-right** corner. By running the simulation, it can be further observed that the runner is exploiting the "*strategy*" of toggling between two corner-most grids to maximize the survival, and hence the reward points.

EXPLORATION V. EXPLOITATION

In Reinforcement Learning, one of the major trade-off problems is *Exploration versus Exploitation*.

- When an algorithm arrives at a sub-optimal solution, the agent choosing to exploit that sub-optimal solution to maximize the rewards is called as Exploitation.
- This is opposed to when the agent is willing to make some sacrifices momentarily and explore the environment a bit more to find the optimal solution; known as Exploration.

SOLUTIONS

In order to make the agent not exploit the local optimum (the toggling between two corner grids) we could try the following solutions:

- ***Penalize Stagnation*:** The code **grid_world_no_stag.py** simulates an environment where the Runner is additionally given negative rewards for stagnating; i.e., the latest history of the Runner's grids is recorded and the Runner is penalized if the same grid is chosen immediately again. This should theoretically discourage the agent to move out of the corners that are currently being exploited.

  Observation: After implementing the above idea, it can be observed that this idea has little practical impact, since the agent still favours the corners and accepts the negative rewards. This happens because the agent sees that the long term goal is to survive on the grid (infinitely) which surpasses his goal to avoid corners.
- ***Make the Tagger Smarter*:** As the problem statement says that the Tagger can run two steps at most, the Tagger can take advantage of this constraint and run two steps when the Runner is far and one step when the Runner is closer, which would chase the runner even more greedily.

  Observation: After implementing the above idea (**grid_world_smart_tag.py**), it can be observed that the Tagger is chasing the Runner extremely fast and not letting the agent learn the environment. This can be tackled by:
  - Adding a bit of randomness to the tagger's movements (this can be seen implemented in the code **grid_world_better_tag.py**); and
  - By increasing the number of grids.*

*__Bonus Question__

The increase in number of grids, even up to N=30 (or more in supported systems) would not only work, but also help the algorithm converge to its true-values faster.

TEMPORAL DIFFERENCE (TD) LEARNING USING SARSA IN GRID WORLD

Temporal Difference (TD) learning is a model-free reinforcement learning method that updates value estimates using the difference between estimated rewards and actual rewards from interacting with the environment. Here, we implemented SARSA (State-Action-Reward-State-Action), an on-policy TD control algorithm to train the Runner agent.

State or (s): The runner's position in the grid.

Action or (a): The runner's movement (in any of the 8 directions).

Reward or (r): The feedback received when the runner is caught by the tagger (negative reward) or survives (positive reward).

Q-value or (Q(s, a)): An estimate of the expected cumulative reward for taking action aaa in state sss.

Explanation

SARSA uses the current state-action pair and the next state-action pair for updates, making it an on-policy TD method. The agent learns to survive by continually adjusting Q-values, refining its policy over many episodes by balancing exploration and exploitation. This method trains the runner to avoid the tagger by learning from its interactions with the environment.

Submitted by *Sai Pranith Bhagavatula* to *Dr. Aritra Hazra*

FINAL OBSERVATION

In our Grid World project of Fast Tagger V-1, we can clearly observe that the TD algorithm works much better over VI as VI seemingly exploits the local optimum, while TD learns and adapts to the environment.