Project Report - III

Better, Smarter, Faster

Sri Krishna Kaashyap Madabhushi
Net ID: sm2599

Lokesh Kodavati
Net ID: bk576

August 24, 2023

xc

1 Implementation

In this project, we want to build an agent that captures the prey as efficiently as possible.

1.1 Environment

Just like Project 2, for this project, we have represented our graph as an adjacency matrix. As a pre-computation step, we have executed the Floyd Warshall algorithm on our graph such that we have a distance matrix that stores distance from every pair of nodes. The following are the steps we used to generate the graph

1 Initially we defined an adjacency matrix of size *50 x 50*.

2 After creating the matrix, we iterated through every node adding an edge between all the adjacent nodes.

3 After this step has been performed, all the nodes in the graphs had the degree 2.

4 To further connect the graph, if any node has a degree 2, we have selected a random node in its [-5, +5] distance range and added an edge if the degree of both the nodes are 2.

5 We did this by iterating through every node.

How many distinct states (configurations of predator, agent, prey) are possible in this environment?

There are 50 possible locations for the prey, predator, and agent individually. Therefore, this environment has a total of 50*50*50 = 125000 distinct states.

What states s are easy to determine U∗ for?

All the terminal states are easy to determine. In our project, we have assigned a U* value of 0 for all the states in which the positions of the agent and prey are the same, indicating that the agent doesn't need to take any more steps to catch the prey. We have also initialized the states where the predator's and agents' positions are similar to infinity. We have also initialized the predator's neighbors to infinity in these states, indicating that the agent shouldn't move to both predators' position and to its neighbors.

How does U∗(s) relate to U∗ of other states and the actions the agent can take?

U*(s) can be interpreted as the maximum/minimum utility (in this case, the effort) required by the agent to win. i.e., as we move closer to the prey, we have a favorable outcome towards 0 and since dying in the hands of a predator is what we want to avoid, we denote that the agent would require an "infinitely" great effort to win as its impossible to win after dying. So in our case, the agent minimizes the effort i.e., moves towards prey and away from predator. The U* values of the other states denote how much effort the agent would need to win and the agent chooses the minimum of all those possible states to win.

2 The definition of U*

For a given state **S**, we define the U*(S) to be the minimum number of rounds it takes to catch the prey. Here, just like project 2, first the agent moves, then the prey, and finally, the predator.

3 Computing the U*

We need to compute the U* for every possible state for the prey, predator, and Agent. To compute this, we create a 3-dimensional matrix of size 50*50*50, where the first index indicates the agent's position, the second index indicates the prey's position and the third index indicates the predator's position. This matrix considers all the possible states of the agent, prey, and predator. Inorder to compute the U*, we would need to consider the parameters:

- State Space - The state space is the set of all possible states of the agent, prey, and predator.
- Action Space - The action space represents the graph and the actions that can be taken to move from one state to another.
- Transition Probability - It is defined as the probability of transitioning from one state to another state by taking an action.
- Reward - It is the reward of moving from one state to another state by taking an action.

In this project, we have used Value Iteration to compute the U* values and used the dynamic programming approach to solve it. Under this approach, we have initialized the terminal states in the U* matrix and have made some initial guesses about the U* values. We have assigned the Agent's and Prey's distances as the initial values so that our U* matrix converges faster. Now, we use the Bellman equation to update the estimates.

$$\boxed{U^*_{k+1}(s) = \max_{a \in A(s)} \left[r_{s,a} + \beta \sum_{s'} p^a_{s,s'} U^*_k(s') \right]}$$

Figure 1: Bellman Equation for U* Value Iteration

In our case, we minimize the Utility as we consider our reward in the form of cost. Now, as the time step k in the above equation tends to infinity, the values of the utility converge close to the true values of the utility. Therefore, when we reach a point where the change (or delta) is very low or negligible, we stop the iterations and consider those estimates to be the true values.

We have discussed the state space and the action space. Now, let's explore the transition probability and the rewards to fill up the equation and complete our calculation.

3.1 Transition Probabilities of Prey

Since the prey's movement is random and can move to one of its neighbors, including staying at the current cell, there is a total of *d+1* moves the prey can take where d is the degree of a node. Therefore, the prey's transition probability is defined as *1 / d+1*.

3.2 Transition Probabilities of Predator

We are using the distracted predator in this project. The idea of a distracted predator is that it moves towards the agent with **60%** probability and gets distracted with **40%** probability. If multiple nodes take us closer to the agent, we choose a random node among them. Therefore, we first calculate the nodes that take us closer to the agent. Let this set be m. If the neighbor is part of this set, then the transition probability is:

$$0.6 * \frac{1}{size(m)} + 0.4 * \frac{1}{n+1}$$

Figure 2: Probability Transition if a neighbour is part of shortest set

In the above equation, n is the degree of the current predator node. If the neighbor is not part of the shortest set, then it is the distracted setting and the transition probability is $0.4 * \frac{1}{n+1}$ where n is the degree of the current predator node.

3.3 Reward/Cost

Our Reward can be considered in a way as the cost. We have assigned an infinity reward for all the states where the predator and the agent's nodes are the same. We also assign an infinity reward for the nodes next to the neighbors. This indicates that we can never catch the prey if we translate into these states. Imagine we pick a state next to the predator; since the prey and predator move after the agent, it is highly likely that the predator will kill the agent. Now, we assign a reward of 0 for all the states where the prey and the agent are in the same node, indicating we don't need to take any more steps to catch the prey, and for all the nonterminal states, we assign a reward of 1.

3.4 Implementation

Now, we have all the variables we need to compute U*. We now simply iterate over all the states, and for each state, we iterate through all actions and store the minimum of all the actions for a state. We repeat this process until the maximum change in the delta values is below a threshold value. At the end of this step, we have a matrix that accounts for all the states and the utility values for each state.

4 Agent U*

At this point, we have successfully computed the U* values for all the states. We spawn the agent, prey, and predator at random positions. At each time step, we find the agent's neighbors and check the utility values of all the states that are formed using the agent's neighbors. We now move to the node with the minimum utility value. And end the game when we get into the prey's node.

4.1 Performance of Agent U*

We have run the agent 100 times on a graph, and the success rate for our Agent 1 is mostly 100%. This indicates that the U* values that are being computed are optimal. The average number of steps it took to catch the prey in U* is 12.87.

Are there any starting states for which the agent will not be able to capture the prey? What causes this failure?

The states where the agent and predator spawn in the same node are the states in which the agent will not be able to capture the prey. Apart from that, all the states where the agent spawns next to the predator are the starting states for which the agent will not be able to capture the prey. Since the Predator move with 0.4 random movements, even if the agent chooses the move with good utility, there is a chance that the predator might randomly kill the agent.

4.2 Maximum Possible U* Value

The Maximum Possible U* value for our graph is 3.0763972361086975. This is for the state where the agent is at 24, the prey is at 31, and the predator is at 28. This is a situation where the agent, to reach the prey, has to cross paths with the predator, and since the agent wouldn't take those risky steps, that state is denoted as the one with the most effort(utility).

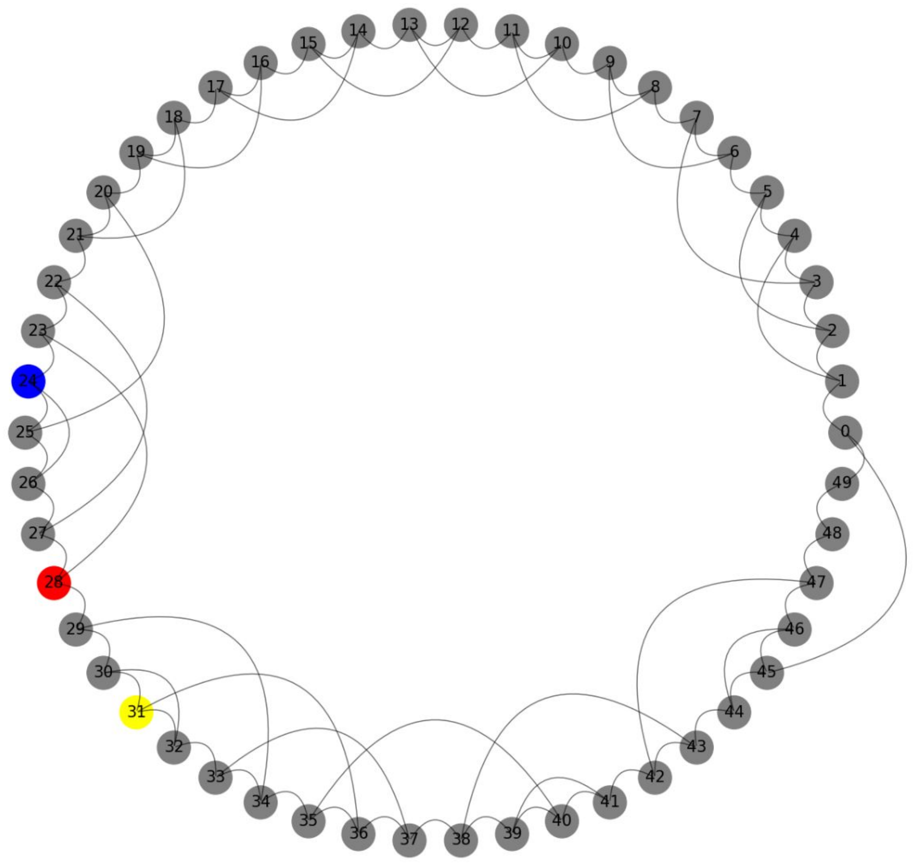

Figure 3: The Representation of the state which has the maximum U* value in the Graph

4.3 Performance Comparison in terms of steps to capture the prey

On average, Agent 1 takes 28.56 steps to capture the prey. Agent 2 on the other hand takes 24.33. Since our Agent U* is most optimal and with the discount factor, the reward that occurs further away are less valued; the overall steps Agent U* takes to catch the prey is least and its at 12.87.

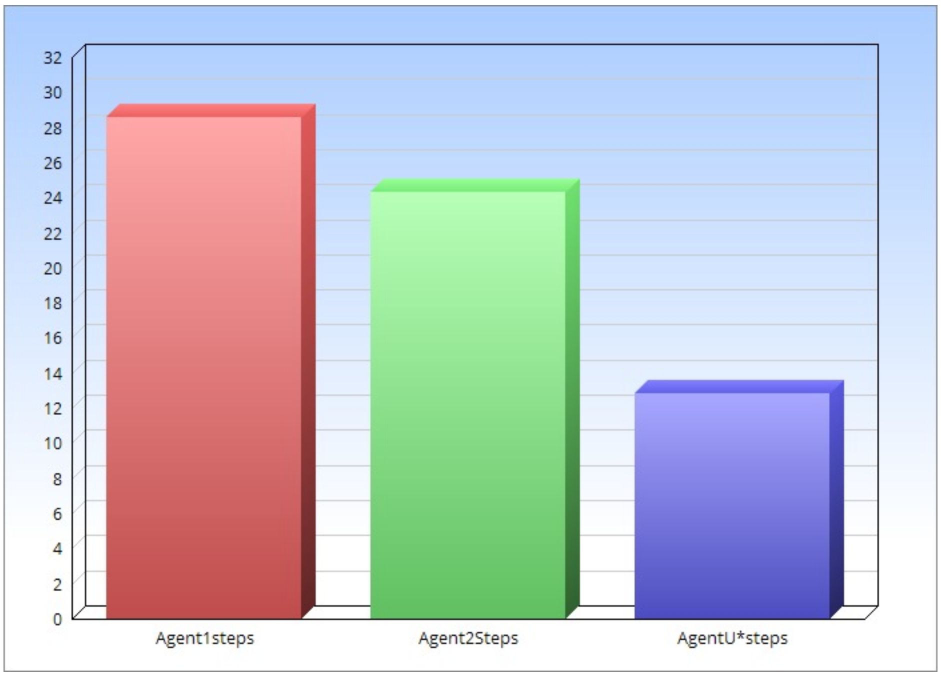

Figure 4: The comparison of Agent 1, 2 and U* in terms of number of steps

4.4 Agent 1, Agent 2 and U* Choice Analysis

Agent 1 moves on the fixed premise of rules; it will examine each available neighbor and select from them in the following order (breaking ties at random).

- Neighbors that are closer to the Prey and farther from the Predator.
- Neighbors that are closer to the Prey and not closer to the Predator.
- Neighbors that are not farther from the Prey and farther from the Predator.
- Neighbors that are not farther from the Prey and not closer to the Predator.
- Neighbors that are farther from the Predator.
- Neighbors that are not closer to the Predator.
- Sit still and pray.

Agent 2 improves its moves comparatively by having a heuristic dependent on its distance from the predator and prey. This can be considered as the immediate reward for the particular move. The agent takes the move with the better value of the heuristic, which rewards more when the agent is distant from the predator and close to the prey.

Agent U star further improves this by taking into account the non-terminal rewards and infinite future into consideration, therefore rewarding each state with a score for the agent to choose, which would enable it to make a move that would allow it to consider the far future while making a current move.

For instance, consider a state where the agent is at 37, the prey is at 35, the predator is at 42

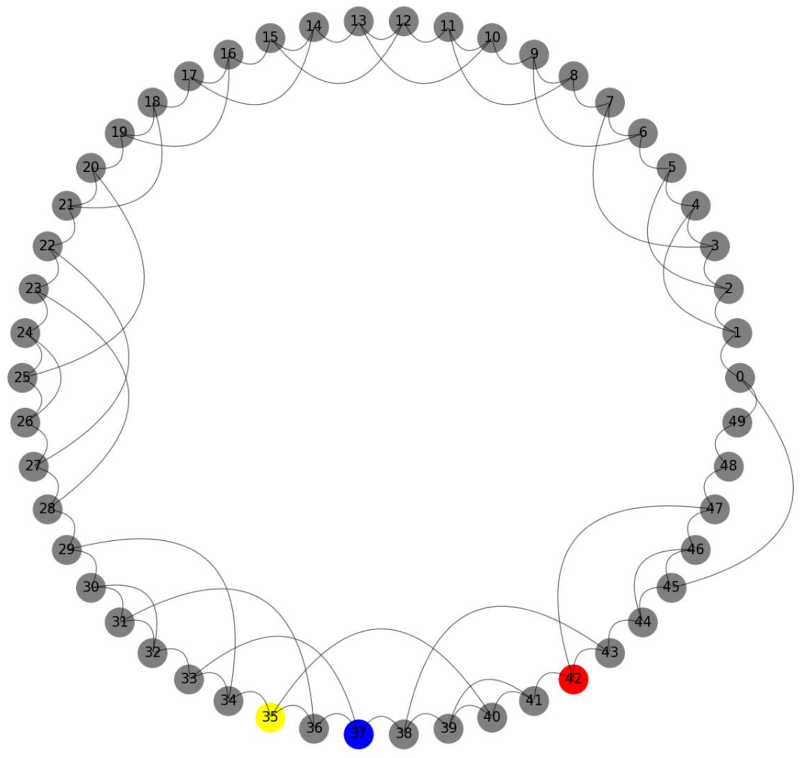

Figure 5: The Sample state where Agent 1, Agent 2, and Agent U-star took different moves

Using the rules mentioned earlier, agent1 moves to 36.

Since Agent 2 prioritizes moving away as far as possible from predator and closer to prey as the best way to survive and considers such a move has the most immediate reward, it moves to 33

In the case of Agent Ustar, though, the move to 38 is near the predator and away from the prey. In the long run, it holds a better overall reward (Utility Value) than the rest of the states. Therefore it prefers to go to that state.

5 Agent U* Partial

At this point, we have computed utilities and a belief array of the prey's position. Just like project 2, at every time step, we scout a node that has the highest probability. After scouting a node, we update the belief array to account for the new information about the prey. After updating the belief array, we compute a neighbor's utility by considering all the nodes in which the prey can be and its probability. It can be easily computed using:

$$U_{\text{partial}}(s_{\text{agent}}, s_{\text{predator}}, \underline{p}) = \sum_{s_{\text{prey}}} p_{s_{\text{prey}}} U^*(s_{\text{agent}}, s_{\text{predator}}, s_{\text{prey}}).$$

Figure 6: Formula to compute the utility in a Partial Prey Setting

5.1 Performance of Agent U* Partial

When we run our U* partial agent on a graph for 100 iterations, we get a success rate between 98 and 100 %. In most cases, it's performing as efficiently as our Agent U*, indicating the complete convergence of the Utility values. The average number of steps is 20.34.

5.2 Agent 3, Agent 4 and Agent U* Partial Performance Comparison

The average number of steps Agent 3 takes to catch the prey is 37.12. Since we have introduced a heuristic in our Agent 4, the number of steps is reduced to 32.89. Finally, our U* Partial is the most optimal with the number of steps being 20.34. Yes, U*Partial agent is the most optimal agent for this information environment because this takes the entire states and the best optimal future reward into consideration and wins, while also making fewer moves.

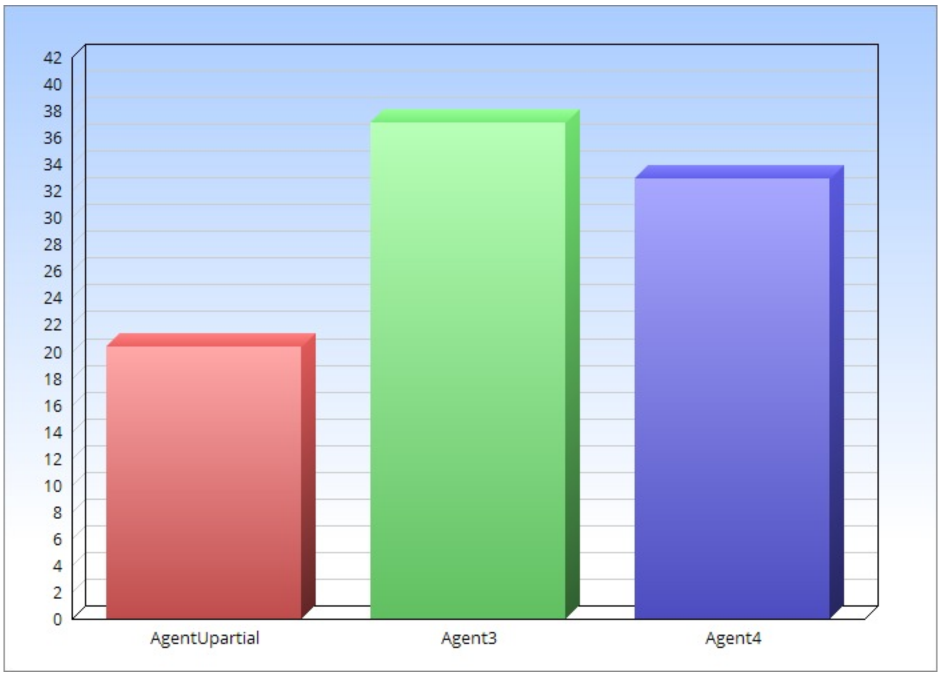

Figure 7: Performance Comparison of Agent 3, 4 and U* Partial in terms of Number of Steps

6 Model V

After computing and storing all the U* values, our aim now is to use the existing information and train a model to predict the utility values. We have considered the following features as our dataset to compute the utility values and used a *5 x 10 x 10 x 1* neural network architecture to predict the values.

- Agent's position
- Prey's position
- Predators Position

- Distance between Agent and Prey
- Distance between Agent and Predator

We have all the information required from the utility matrix and the Floyd Warshal distance matrix and have 125,000 features.

6.1 Model Architecture

We have used fully connected neural networks to predict the utility values of the states. Since we have five inputs, our input layer has five nodes, one for each of the inputs. We now use two hidden layers with ten nodes each. And finally, an output layer with one node. We have used a linear activation function (ReLU) as our activation function throughout the network. We have split the data in the ratio of 80:20 as our training data and the test data. Therefore, we have 100,000 entries in our training data and 25,000 in our test data. We have also followed batch training, where we fed our entire data into the architecture as a batch. Therefore, the weights matrix at each level is as follows:

- Weights vector at the input layer has the dimensions of *5 x 10*, and the bias vector has the dimensions of *1 x 10*.
- When we pass our input of dimensions *100000 x 5* and perform the dot product with the W1 matrix, we get a *100000 x 10* matrix.
- Now, this *100000 x 10* matrix is passed into the second layer whose dimensions is *10 x 10* that returns a matrix of *100000 x 10*.
- This *100000 x 10* matrix is passed into the third layer whose dimensions is *10 x 10* that again returns a matrix of *100000 x 10*.
- And finally, this matrix is passed into a *100000 x 1* with a single node to return a vector of *100000 x 1*, and an error is calculated with the actual values of the dataset.
- Based on the error calculated, we back-propagate the error and adjust the weights of the hidden layers.
- We repeat the above steps until we finish all the epochs. We have trained the data for 10000 epochs.

Except for the output layer, we have used the ReLU activation function throughout the architecture and have analyzed the network using the Mean Squared Error metric.

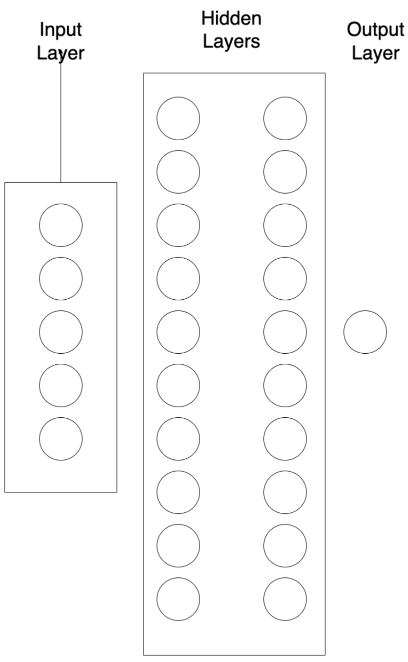

Figure 8: Image representing our architecture

How do you represent the states S as input for your model? What kind of features might be relevant?

We represent the state "S" as three separate columns, the agent's position, the prey's position, and the predator's position. We also shuffled the data randomly a couple of times before feeding the data into the network. Apart from the state space, the distance from the agent to the prey and the distance from the agent to the predator are also important features where the model could derive a relationship between the distances.

What kind of model are you taking V to be? How do you train it?

We are considering V to be a regression-based neural network model where we have a set of input values and need to predict a real number. We have also observed that all our utility values are real numbers less than ten. Therefore, we have replaced the infinity value with 100, so all our error calculations are capped. We have followed the above steps using this dataset to train the model.

Is overfitting an issue here?

Overfitting is not an issue in this case because we have 125,000 states and different utility values for each state. Even after splitting the data into the train-test split, we have sufficient data points for the model to avoid overfitting.

How accurate is V ?

We ran Agent V on the graph and the average success rate we achieved when we ran on an average was 97%.

7 Agent V*

We now have a model that predicts a utility value for a given state. We provide the agent's position, the prey's position, the predator's position, the distance between the agent and prey, and the distance between the agent and predator as the inputs, a *1 x 5*, vector, and we finally get a single value that gives us a predicted utility. Like the U* agent, we iterate and move to the node with the least expected value.

7.1 Performance of Agent V*

Our Agent V* has an average success rate of 97%. The average number of steps it takes to catch the prey is 16.78

8 Model Partial V

After developing the partial U* setting, we now develop a model to predict the utility values in the partial prey information setting. We use a similar neural network architecture in this information set but we have a different set of inputs. Instead of the prey's absolute position, we have a list of probabilities indicating the probability of the prey being in that location.

8.1 Data Generation

To build the model for this environment, we get the data from U* partial information setting. In the U* partial prey information environment, we first scout a node, and based on the findings of the scout, we update our belief array. After updating the belief array, we iterate through the belief array and compute the U* value. During this step, we have stored the computed U* values along with the states that include the agent's position, the prey's belief array, and the predator's position along with the distance between the agent and the predator. We have run multiple simulations of U* partial and collected around 50000 data points along with the expected utility.

8.2 Model Training

Now, after obtaining the data, we have a 53-sized vector as an input that signifies the agent's position, the prey's belief array, the predator's position, the distance between the agent and the predator. We have designed a Neural Network architecture of *53 x 106 x 53 x 1*. We have split the data into training and testing sets and trained the model using the steps mentioned in model V.

How do you represent the states Sagent, Spredator, p as input for your model? What kind of features might be relevant?

We represent the position of the agent, predator, and the belief array of the prey as individual columns. These take up 52 columns in our input data. The most important feature being considered is the distance between the agent and the predator. Therefore, we have a total of 53-sized NumPy array in our input data for the model.

What kind of model are you taking Vpartial to be? How do you train it?

We take the V-Partial model to be a regression-based deep neural networks model where we have a set of input values and we need to predict a real number. The detailed steps of how the model is trained and how the data is obtained are explained above.

Is overfitting an issue here? What can you do about it?

Yes, Overfitting is an issue here. This is because there are only *50 x 50* unique combinations of agent and predator positions. And with a slight change in the belief array of the predator, there is a significant change in the utility values. Therefore, it is an issue, and we can overcome it by using a regularization technique like a dropout. Dropout is a technique used to reduce overfitting in neural networks by preventing complex co-adaptations on training data. Here, the utility values change significantly with a minor change in the prey's belief array. In cases like these, drop out is a perfect technique to avoid overfitting.

How accurate is Vpartial? How can you judge this?

V partial is less accurate. We believe that loss is not the right metric to compute the accuracy of the model because this model has overfitting in it and there are comparatively less states in the Vpartial as compared to V.

Is Vpartial more or less accurate than simply substituting V into equation

V Partial is less accurate than simply substituting the V into the equation because the Vpartial model has overfitting in it. When we substitute V in place of Vpartial, the problem almost converges to finding the V which has a higher accuracy than Vpartial.

8.3 V* Partial Performance

The success rate of V* partial is around 88% while the number of steps it takes to catch the prey is 26.45. U* partial performs way better than the V* Partial because the values of U*Partial are far more close to the actual values and use the true values as compared to the V* partial.

9 Bonus 1

After running the above model based on the belief array of the probability, we found that there are conditions of overfitting in the model. This is because there are only *50 x 50* unique combinations of the Agent and the predator's positions. Therefore, instead of taking the belief array as the input to our model, where the utility value changes significantly for a slight change in the probability of the belief array, we assume that the prey is located in the node with the highest probability after scouting breaking ties at random and consider that as our prey's position. When we do this, we reduce the strong dependency of our output variable on the input parameters where previously a small change in the probability resulted in a significant change in the output; reducing variance. Similar to model V, when we consider this node as our prey's position, we have a lot more unique datapoints and their corresponding utility values. Now, we also take the distance from the agent to this node as an extra attribute in our model. When we make these changes and consider these, it is observed that this model is closely getting converged to model V, thereby improving our success rate of the partial prey information agent. Since the predator is always visible, the number of steps taken to catch prey is high in this setting but overall, the success rate has improved as compared to the partial prey environment.

This has improved our success rate by around 5%, bringing our average success rate to around 94% but has increased the number of steps from 26.45 to about 32.14.

10 Bonus 2