CS 520: Markov Decision Process and Neural Networks

Project 3: Better Smarter Faster

Prepared by:

dm1639: Darshee Machhar
Rutgers University

pn266: Pankti Nanavati
Rutgers University

13th December, 2022

Contents

1 Introduction

1.1 Problem Statement- The Game

Like Project 2, within a circular graph, some nodes of which are occupied by a prey or a predator, an agent will try to capture the prey and, at the same time, evade the predator. In Project 2, we wanted to ensure the agent survives, and if it can evade the predator for sufficiently long, it will eventually catch the prey. We were not worried about the efficiency of the agent. In Project 3, we will try to do precisely that. We aim to build an efficient agent by curating utilities and training a neural network to learn said utility.

1.2 Problem Environment

The environment consists of a circular graph in which we add edges to make the graph more connected. The prey and predator are two separate entities that are completely independent of each other. They can reside in the same spot and not harm each other. When we spawn the agent, prey, and predator, we just have to ensure that the agent and prey or the agent and predator aren't spawned in the same spot.

1.3 The Prey

It moves randomly. It can move to any of its neighbors or continue to inhabit the same node with equal probability.

1.4 The Distracted Predator

The predator is more determined. It seeks out the shortest path to the agent. This predator can go to any of its neighbors uniformly at random with a 40% chance or go to the shortest path node with a 60% chance.

1.5 The States

The states are configurations of agent, prey, and predator positions.

Question: How many distinct states are possible in this environment?
Answer: Since there are 50 nodes in the graph. Each of our actors is free to be at those 50 nodes. So total combinations of their positions will be 50^3. Therefore the number of distinct states possible within our environment is 125,000.

2 U-Star

2.1 U* Design

We want to create a utility that captures the least possible number of steps it can take to catch the prey for an optimal and efficient agent. This utility will be a value for each state, which our agent can use to navigate the graph efficiently.

Question: What states s are easy to determine U* for?
Answer: Since our utility is the minimal number of steps to reach the prey. It is the easiest to calculate that value when we are close to the prey, more accurately, when we are in the prey's neighbouring nodes or when the agent has caught the prey. Therefore those states where the agent and prey occupy the same position and those states where the agent is next to the prey in the graph will be the states for which calculating the Utility will be the easiest.

Question: How does U*(s) relate to U* of other states and the actions the agent can take?
Answer: In order to calculate this utility, we have designed a Markov Decision Process-
For a finite state space S:
for every state $s \in S$, we have a finite action set A(s),
where taking action $a \in A(s)$ while in the state s will earn reward r_s, a transition to a new state s'
From state s, taking action, a will transition to state s with probability P_s, s'(a)
In every time step, based on the current state (starting at some initial state, U_o), the controller must decide which of the available actions to take by collecting the corresponding reward and transitioning to the next state.

Where,

State
S: All the possible combinations of agent, prey, and predator positions

Action
A(s): We will define the action or the agent's action as all the possible next movements of the agent, i.e., the neighbors of the agent's current position.

Reward
r_s, a: Since we have to curate a utility that captures the minimal rounds the agent is taking to catch the prey, we have assigned the reward of each move as 1, considering it as a cost (one step or round towards the prey) for the agent.

Initial Utility
U_o: We assign the initial Utility of all states as 1.
Only in cases where the agent and prey are in the same position do we assign

the reward as 0.
And when the agent and predator are in the same position or when the predator is in one of the neighbors of the agent's current position, we assign the reward as infinity.

Transition Probability
P_s, s'(a): The probability of moving from state s to s' upon taking action a depends on the movement of the prey and predator since states here are defined as a combination of the positions of all the three actors.

So we calculate the probability of both prey and predator in a manner akin to their moving behaviors.
The prey is equally likely to move to any of its neighbors or stay where it was. So, P(Prey at s' | Prey at s) will be 1/ (no of neighbors +1)

Unlike the prey, the predator cannot continue to be in the same position. Also, it is the distracted predator. Therefore P (Predator at s' | Predator at s) = 0.6 + 0.4* 1/ (no of neighbors) -> if s' predator position is the closest to the agent from state s.

Else
P (Predator at s' | Predator at s) = 0.4* 1/ (no of neighbors)

2.2 Value Iteration Algorithm

Question: Write a program to determine U* for every state s and each state and what action the algorithm should take. Describe your algorithm in detail.
Answer: We usually calculate U* using the below Bellman Equation using the discounting factor-beta.

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

But for our particular situation, we aren't calculating rewards many timesteps into the future; we are calculating the Utility by considering the reward to be taken in the very next step. So we don't use the discounting factor.

Also, since we are making a utility of the minimal steps to be taken by the agent to reach the prey, instead of finding the max value over actions, we find the minimum value over actions. That is, instead of argmax, we calculate the argmin.

Therefore our equation is as follows:

$$U^*(s) = \min_{a \in A(s)} \left[r_{s,a} + \sum_{s'} p^a_{s,s'} U^*(s') \right]$$

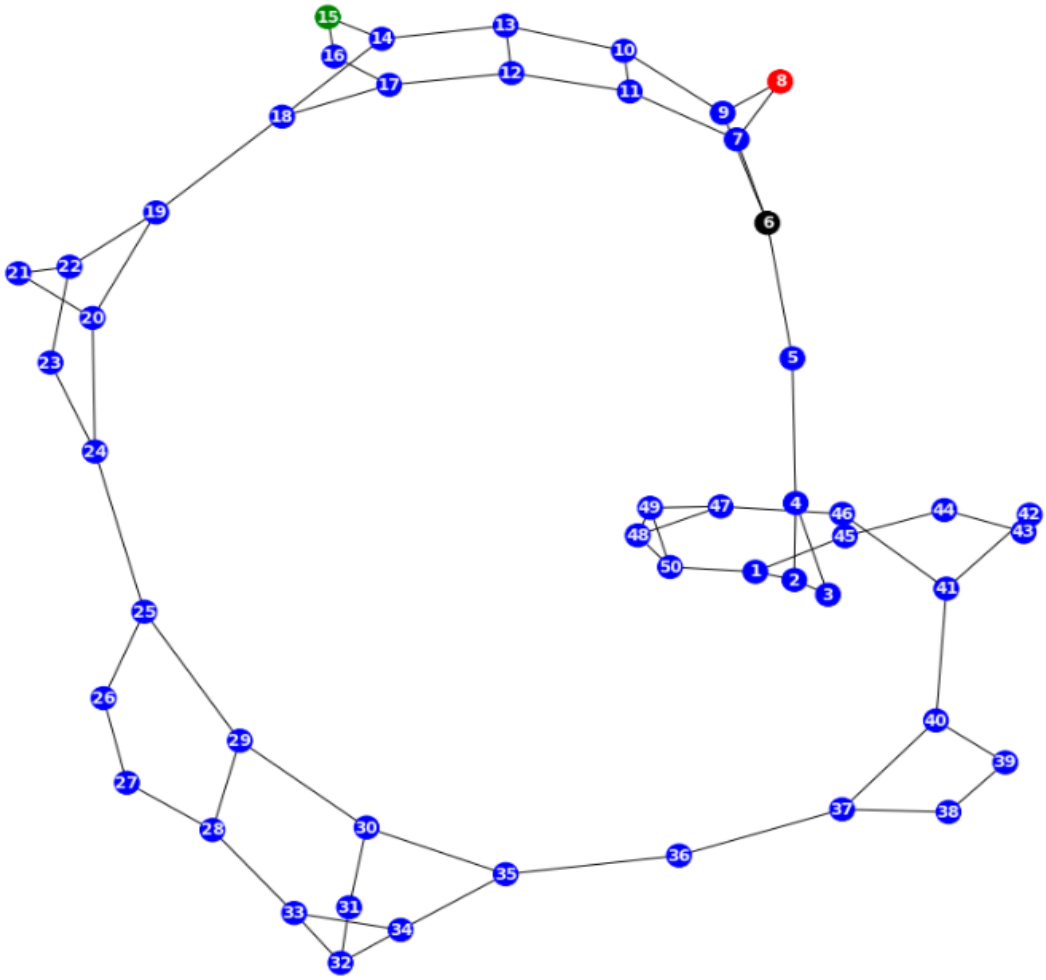
In order to solve our Bellman equations, we use Value Iteration.

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

This can be viewed as an iterated updated version of Bellman's equations above. The significance of this scheme is that the successive approximations U*k converge to U*.

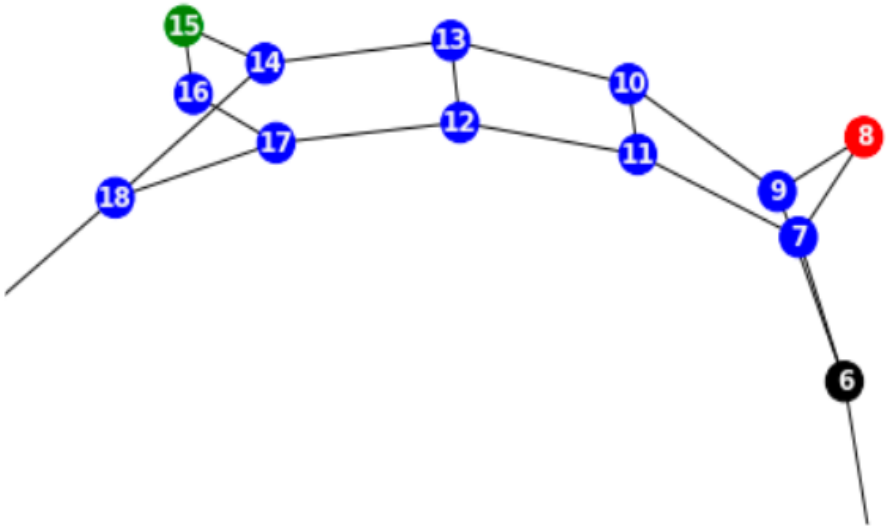
Question: Find the state with the largest possible finite value of U* and give a visualization of it.
Answer: The largest possible finite value of U* we got is 19.39.
The maximum number of steps to reach the prey is 19, and never more than that.
The state for which we get this value is (6,15,8).
That is, the agent is at node 6, the prey is at node 15, and the predator is at node 8.

The visualization for that is given below:



Black Node = Agent
Green Node = Prey
Red Node = Predator

Zooming in on the positions to make it clearer:

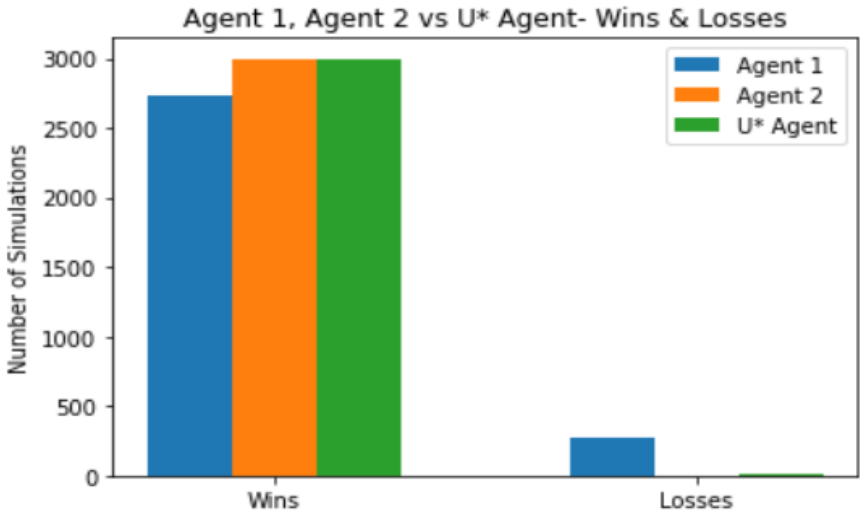


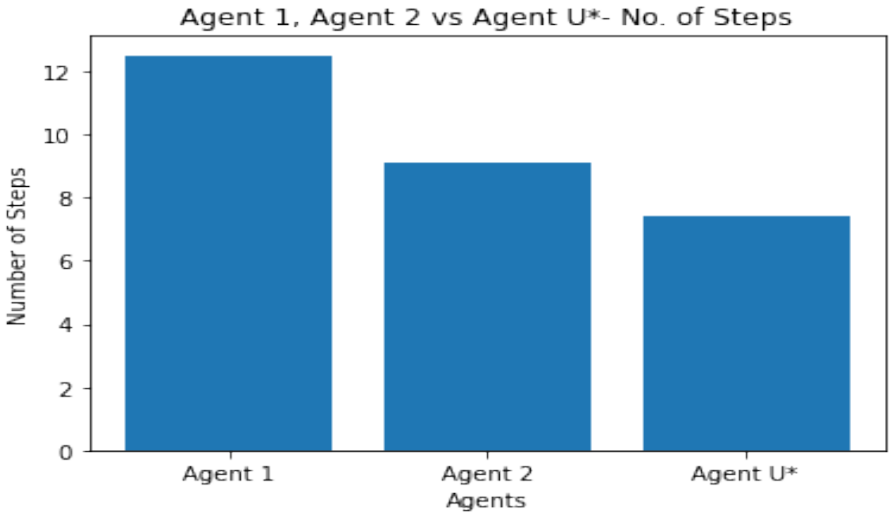
2.3 U* Agent

Question: Simulate the performance of an agent based on U* and compare its performance (in terms of steps to capture the prey) to Agent 1 and Agent 2 in Project 2. How do they compare?
Answer:

- How often the Predator catches the Agent?
  Answer- 6 times out of 3000 times (0.20%) the predator catches the agent i.e., the number of times the agent loses the game
- How often the Agent catches the Prey?
  Answer- 2994 times out of 3000 times (99.80%) the agent catches the prey i.e., the number of times the agent wins the game.
- The average number of steps
  Answer- 7.40

The histogram and the bar chart given below compares the performance in terms of wins and losses of Agent 1, Agent 2 and U* Agent and the number of steps to capture the prey.





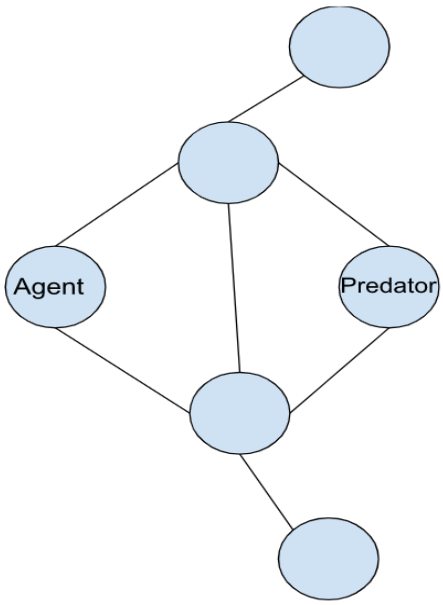
Wins- Agent 1= 2730 (91%); Agent 2= 2997 (99.9%); U* Agent= 2994 (99.8%)
Number of steps to capture the prey- Agent 1= 12.50; Agent 2= 9.13; U* Agent= 7.40
Hence, we can conclude that U* Agent and Agent 2 performance is comparable while the U* Agent takes lesser number of average steps to catch the prey.
Question: Are there any starting states for which the agent will not be able

to capture the prey? What causes this failure?
Answer: We observed 2 scenarios in which our agent was dying and hence not being able to catch the prey.

Sceanrio 1: Predator and Agent spawned at the same position.

In this scenario, the agent dies right at the very beginning of the game.

Scenario 2: Predator is bound to get the agent in next step.



Though rare, we did come across this scenario, no matter where the agent goes next, the predator can catch it.

Question: Are there states where the U* agent and Agent 1 make different choices? The U* agent and Agent 2? Visualize such a state, if one exists, and explain why the U* agent makes its choice.
Answer: A Scenario where Agent 1 and U* make different choices:
Agent 1 has to choose between decreasing the distance between prey and increasing that between itself and the predator, and it does so in consequent steps. It will pursue the prey first and if the predator gets too close, it will evade it.

It used one extra step to make the optimal move, whereas the U* agent would take a move that simultaneously takes the agent closer to the prey optimally whilst also making sure the agent doesn't die.

A scenario where Agent 2 and U* make different choices:
Similarly, Agent 2, also utilizes one extra step. Agent 2 prioritizes survivability, so it will take steps away from the predator to evade it before it takes a step to pursue the prey, hence taking more rounds to end the game.
As mentioned before, the U* agent would take an optimal move and not waste steps.

3 V Model

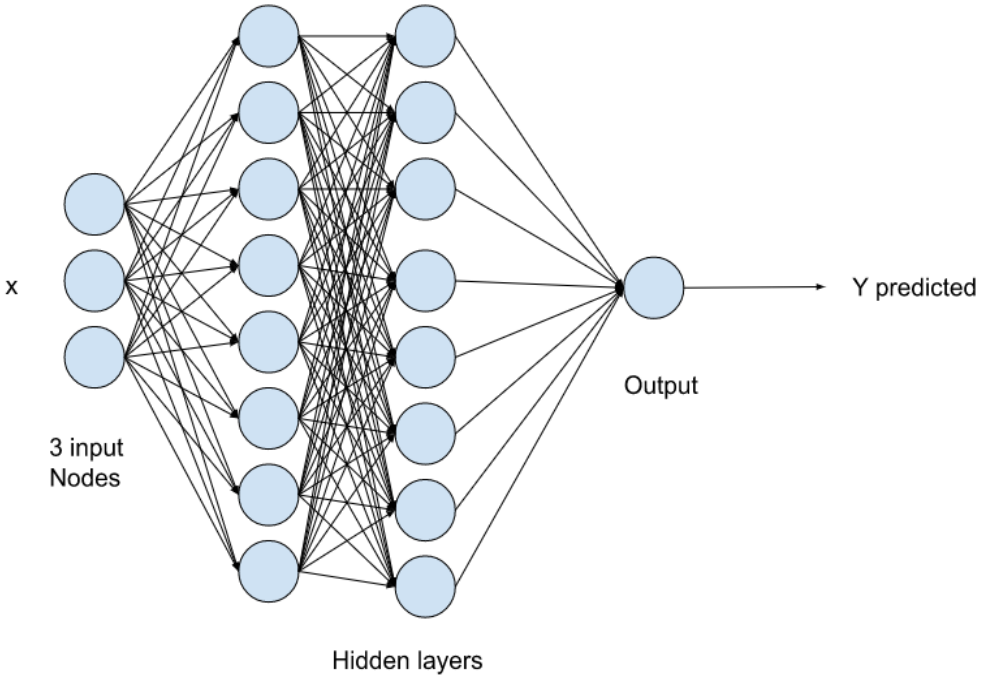
3.1 Input Data and Features

Question: How do you represent the states s as input for your model? What kind of features might be relevant?
Answer: We have calculated a Utility based on the states. Therefore the input for our model is precisely that. The agent's, prey's and predator's position is fed into the model as input. The states itself encapsulate the relative positioning of the three actors, hence it speaks to the distances between them. Also while calculating the utility, the reward we have considered is 1 for each action. So adding reward as another feature/input didn't seem useful as it would not be adding any information.

Input			Output
Agent Position	Prey Position	Predator Position	Utility

3.2 Neural Network Architecture

Question: What kind of model are you taking V to be?
Answer: We have made a neural network that using the Leaky ReLu activation function. The architecture of our ANN looks like this:



3.3 Training Accuracy

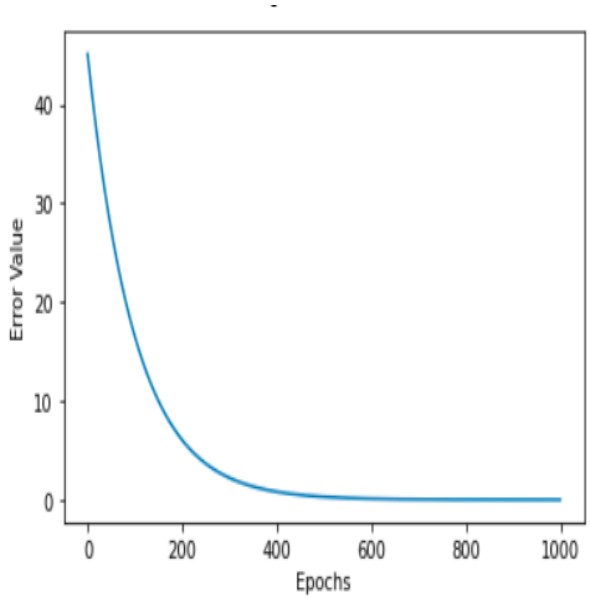
Question: How do you train it?
Answer: These are the steps we followed to train our neural network.

- First an ANN will require a random weight initialization
- Split the dataset into batches
- Calculate the forward pass (what would be the output with the current weights)
- Compare the calculated output to the expected output (loss)
- Adjust the weights (using the learning rate increment or decrement) according to the backward pass (backward gradient propagation).
- Repeat Step 3 with modified weights.

Question: Is over-fitting an issue here?
Answer: Overfitting was not an issue here. Since we have the exhaustive state space with us from the start and we split it into a training and testing set, we always generalize our model's abilities to a certain degree.

Moreover, because we already have the entire data, even if we were to give the whole batch of data to the training, our model would end up memorizing the utilities. It would still not be overfitting, because it would never encounter a different state space.

Question: How accurate is V?
Answer: The model converges until the MSE is **1.87**
This is what the convergence looks like:

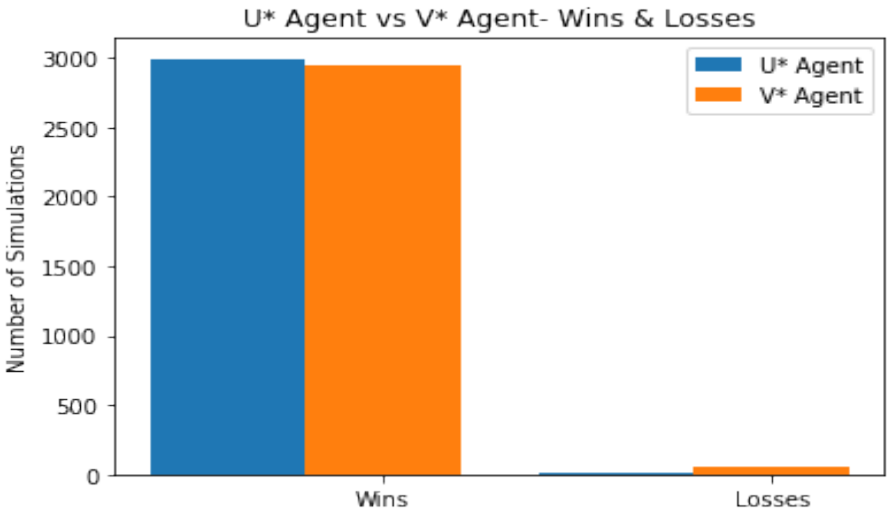


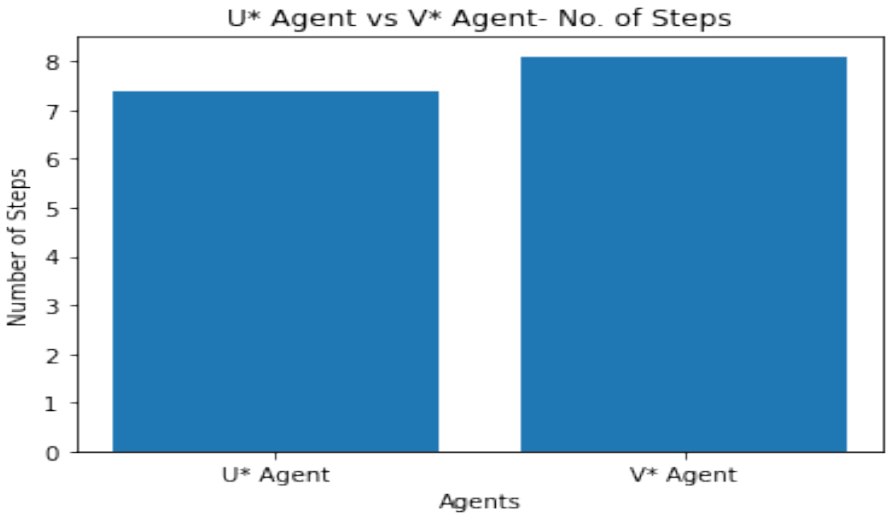
3.4 V Agent

Question: How does its performance stack against the U* agent?
Answer:

- How often the Predator catches the Agent?
  Answer- 54 times out of 3000 times (1.80%) the predator catches the agent i.e., the number of times the agent loses the game
- How often the Agent catches the Prey?
  Answer- 2946 times out of 3000 times (98.20%) the agent catches the prey i.e., the number of times the agent wins the game.
- The average number of steps
  Answer- 8.10

The histogram and the bar chart given below compares the performance in terms of wins and losses of U* Agent and V* Agent and the number of steps to capture the prey.





Wins- U* Agent= 2994 (99.8%); V* Agent= 2946 (98.20%)
Number of steps to capture the prey- U* Agent= 7.40; V* Agent= 8.10
Hence, we can conclude that the performance of both the agents is comparable.

4 U Partial

4.1 Estimation

The state space of U* was the combinations of the agent, prey and predator positions. In a partial information environment, we don't always know where the prey/predator lies.

For $U_{Partial}$, we had to consider the partial information environment where we don't know where the prey is, but the agent maintains a belief state of probabilities to keep track of the prey's most likely positions.

Therefore the state-space of $U_{Partial}$ is the agent's position, the predator's position, and the belief state for the prey's position. Since there could be infinitely many belief states for each agent-predator position combination, it is very tough to calculate the utility of this situation accurately. So we turn to estimations instead, using the given update equation.

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

4.2 Transition Probabilities

The transition probabilities of the prey are calculated similar to project 2.
We modify the belief states, 3 times per round of the game:

- When the agent surveys a node,
- When the agent moves into a new node,
- When prey moves, we propagate its initial probabilities.

When we Survey/ agent moves:
For Current Belief State-
When we survey a node and find the prey, the probability of that node becomes 1, and the rest of the nodes have a probability of 0.
Because the prey can not be in more than one node at any particular timestep.
Say we survey node 2 and find the prey:
Current Belief State = [1: 0, 2: 1, 3: 0, 4: 0, 5: 0]

Upon surveying if we don't find the prey, we have to make the probability

of occupying the prey 0 and the rest of the probabilities will get updated.

We update the probabilities using conditional probability.
Let A be the probability of the prey being at node A.
Let B be the probability of the prey being at node B;

therefore, the probability of the prey being at node a, given that we surveyed b and did not find the prey there, can be represented as P(A | not B)

According to the conditional probability rule: P(x | y) = [P (x). P (y | x)]/ P (y)
We get P(A | not B) = P(A). P(not B|A)/P(not B) =P(A). P(not B | A)/(1-P(B))
Also, P(not B | A) is always going to be 1 because the prey can only be at one node at a time.
Therefore the probability of it not being at B, given that it is at A, is always true, always 1.

Therefore, P(A | not B) = P(A)/ (1-P(B))

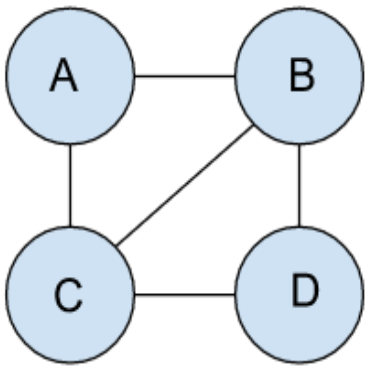
For Example
4 Nodes: A, B, C, D
P(in A) = 0.4
P(in B) = 0.3
P(in C) = 0.2
P(in D) = 0.1
We survey B, prey isn't there
P(in A | prey not in B)
= P(in A) P(prey not in B | in A) / P(prey not in B)
= 0.4 * 1 / (1 - P(in B))
= 0.4/ (1- 0.3)
= 0.4 / 0.7
P(in A / prey not in B) = 0.4 / 0.7
P(in B | prey not in B) = 0
P(in C | prey not in B) = 0.2 / 0.7
P(in D | prey not in B) = 0.1 / 0.7

Similarly, when our agent moves to its new position, and the game doesn't end, i.e., there is no prey at that node the agent moved to. We can make similar changes again to the current belief state.

Now we must observe that all of these calculations were for the current state, but when the agent must choose the most probable prey position to move towards it, he can not use the current state as is because the prey will move in the next time step. Hence we maintain the future belief state. For this, we modify probabilities according to the prey's transitions, as explained in the next section.

When we transition:
The prey can move to any of its neighboring nodes or continue occupying the current node with equal probability. That needs to be reflected when we propagate the probabilities of the current belief state to the future belief state.

We need to find the P(A at T= t+1) depending on the probabilities of its neighbors at T= t
Say B, C, and D are the neighbors of A in this way:



And these are their probabilities as we'd calculated above:
P(in A) = 4/7
P(in B) = 0
P(in C) = 2/7
P(in D) = 1/7

We make use of Marginalization here -
P(Prey in A Next) = P(Prey in A Next, Prey in A Now) +P(Prey in A Next, Prey in B Now) +P(Prey in A Next, Prey in C Now) +P(Prey in A Next, Prey in D Now)

Therefore, we can apply conditional factoring to this and get:
P(Prey in A Next) = P(Prey in A Now) * P(Prey in A Next | Prey in A Now) P(Prey in B Now) * P(Prey in A Next | Prey in B Now) P(Prey in C Now) * P(Prey in A Next | Prey in C Now) P(Prey in D Now) * P(Prey in A Next | Prey in D Now)

Where P(Prey in A Next | Prey in A Now) is basically referring to the one divided by degree of node A plus one [1/(degree + 1)], if A has 2 neighbors, the prey is (100/3) = 33% of its current probability likely to be at its neighboring node or at the same position in the next timestep.

If a node is not connected to the node in question, like node D in the example above: P(Prey in A Next | Prey in D Now) will be 0. Because node D's probabilities won't propagate to node A.

Therefore, P(Prey in A Next)
= (4/7) (1/3) + (0) (1/4) + (2/7) (1/4) + (1/7)
= 4/21 • 1/14
= 0.261.

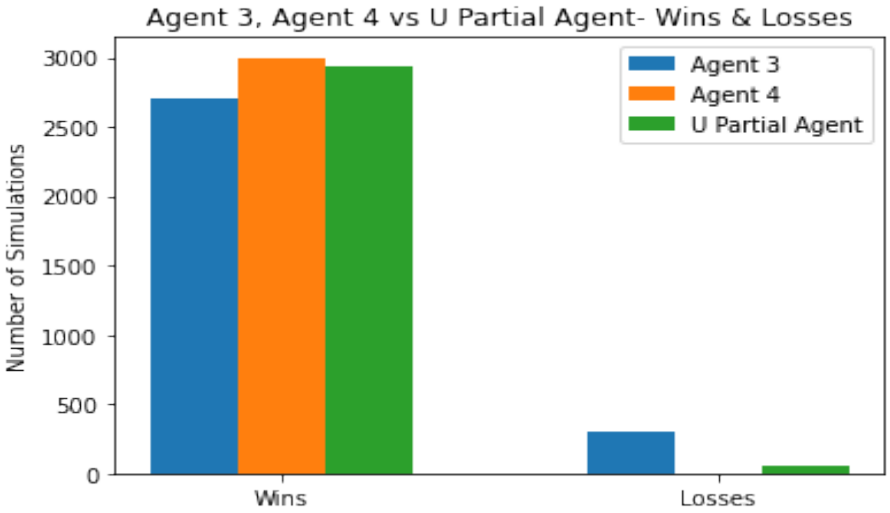
This is how we update our current belief state and future belief state to help the agent have a comprehensive understanding of its environment.

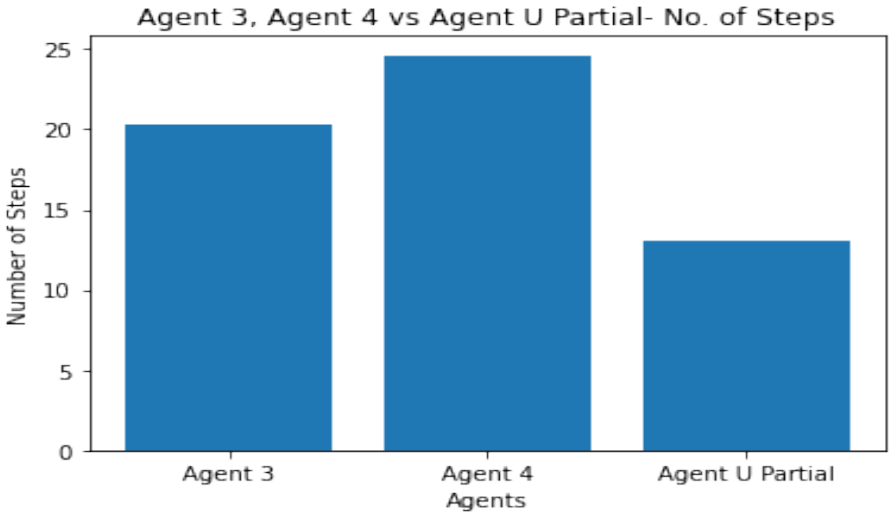
4.3 U Partial Agent

Question: Simulate an agent based on $U_{partial}$ in the partial prey info environment case from Project 2, using the values of U* from above. How does it compare to Agents 3 and 4 from Project 2?
Answer:

- How often the Predator catches the Agent?
  Answer- 60 times out of 3000 times (2.00%) the predator catches the agent i.e., the number of times the agent loses the game
- How often the Agent catches the Prey?
  Answer- 2940 times out of 3000 times (98.00%) the agent catches the prey i.e., the number of times the agent wins the game.
- The average number of steps to catch the prey.
  Answer- 13

The histogram and bar chart given below compares the performance in terms of wins and losses of Agent 3, Agent 4 and U* Partial Agent and the number of steps to capture the prey.





Wins- Agent 3= 2700 (90%); Agent 4= 2997 (99.9%); U* Partial Agent= 2940 (98%)
Number of steps to capture the prey- Agent 3= 20.29; Agent 4= 24.60; U* Partial Agent= 13
Hence, we conclude that Agent 4 has higher success rate while the U Patial Agent takes lesser average number of steps to catch the prey.

Question: Do you think this partial information agent is optimal?
Answer: No this partial information agent is not optimal. We are only finding an estimate of the true utility and hence it comes close to being very efficient but it is not the most optimal.

5 V Partial

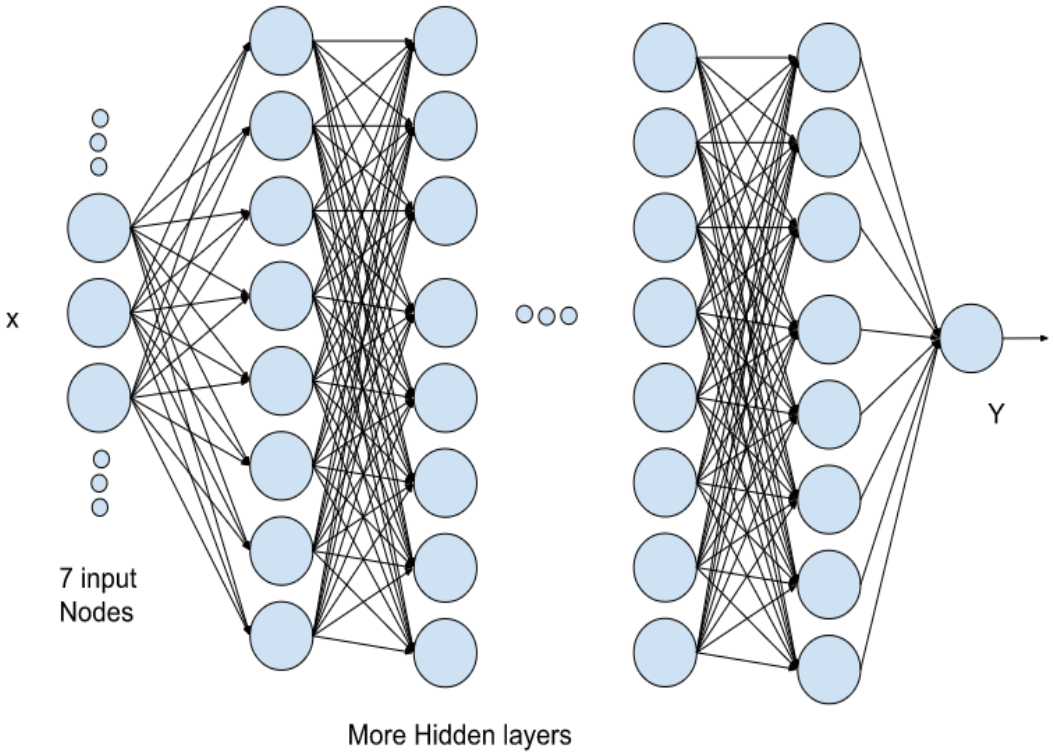
5.1 Input Data and Features

Question: How do you represent the states, agent, predator, p as input for your model?
Question: What kind of features might be relevant?
Answer: We have taken the agent's position and predator's position as the input features. Additionally, instead of taking the entire belief state for the prey in that state, we have taken statistical inferences of the belief state.
These include

- Max probability
- Min Probability
- Mean Probability
- Mode of the probabilities
- Argmax of the prey probability (the most likely node where the prey can be)

5.2 Neural Network Architecture

Question: What kind of model are you taking V partial to be?
Answer: We have made a neural network that uses the Leaky ReLu activation function. The architecture of our ANN looks like this:



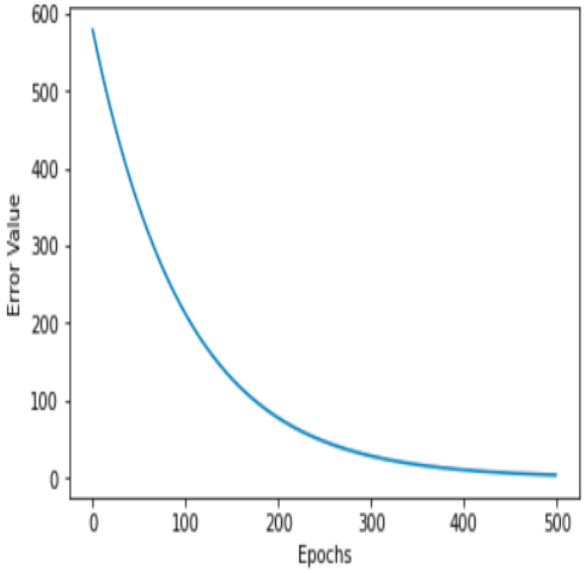
5.3 Training Accuracy

Question: How do you train it?
Answer: We train it similarly to Model V.
We initialized the weights, did forward propagation, calculated the loss, and then did a backward propagation to modify the weights for the next forward propagation.

What we did differently, compared to the previous V star model, is that we added some noise to our input data, generated some belief states at random (still keeping in mind that the sum of probabilities will be one), and then trained the model.

Question: Is over-fitting an issue here? What can you do about it?
Answer: Since the belief states that what could occur in the real game are infinite, the data we have to train the model is limited. The model can get over-fitted to a few scenarios.
Hence we added the noise to the input data to handle the over-fitting issue.

Question: How accurate is V partial? How can you judge this?
Answer: Our V Partial converges to an error (MSE) of 3.83
This is what the convergence graph looks like



Question: Is V Partial more or less accurate than simply substituting V into equation (1)?
Answer: According to our observations, VPartial was almost as accurate as substituting V into equation 1. Their performances were comparable.

5.4 V Partial Agent

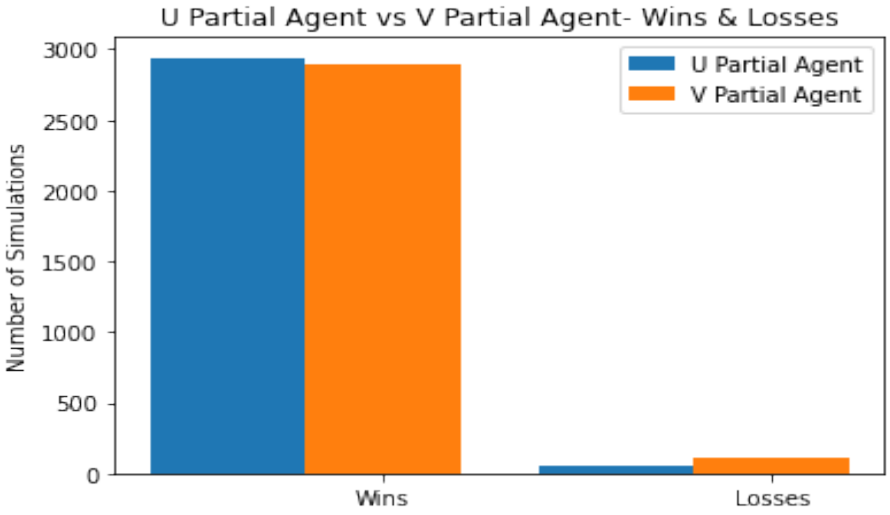
Question: Simulate an agent-based on V-partial. How does it stack against the U-partial agent?
Answer:

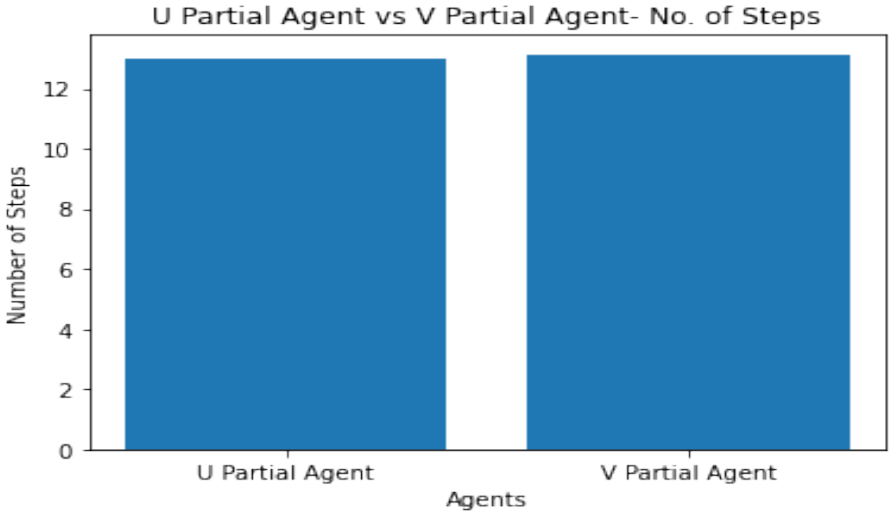
- How often the Predator catches the Agent?
  Answer- 110 times out of 3000 times (3.67%) the predator catches the agent i.e., the number of times the agent loses the game

- How often the Agent catches the Prey?
  Answer- 2890 times out of 3000 times (96.33%) the agent catches the prey i.e., the number of times the agent wins the game.

- The average number of steps
  Answer- 13.13

The histogram and the bar chart given below compares the performance in terms of wins and losses of U* Partial Agent and V* Partial Agent and the number of steps to capture the prey.





Wins- U* Partial Agent= 2940 (98%); V* Partial Agent= 2890 (96.33%)
Number of steps to capture the prey- U* Partial Agent= 13; V* Partial Agent= 13.13
Hence, we can conclude that the performance of both the agents is comparable.