CS 520 - Project 3

Better, Smarter, Faster

Produced by: Pranathi Vaddela

Net ID: PV250

Contents

1 Introduction

In project "Better, Smarter, Faster", we have same environment, a circle of 50 nodes, random edges and with three participants in given **environment**.

1. Agent (Wants to catch Prey)
2. Prey (Moves freely in the Environment)
3. Distracted Predator (Wants to catch Agent, in the distracted environment)

These three entities—the Predator, the Prey, and the Agent occupy the environment and can travel from node to node along the edges. The Agent and the Predator desire to capture Prey and the Agent, respectively. The Agent succeeds if it shares a node with the Prey. The Agent loses if he shares a node with the Predator. The three players move in rounds, starting with the Agent, then the Prey, and then the Predator.

I am taking **connected graph** of 50 nodes as the environment in this project and will analyze the Agent's success rates in different contexts. We are considering the following contexts for experimentation.

1. The Complete Information Setting (Agent always knows exactly where the Prey & Predator is)
   The Agent uses the Utility
2. The Partial Prey Information Setting (Agent is uncertain about Prey location)

2 Implementation from Project 2

2.1 Environment

I am using a connected graph of 50 nodes as my environment. The aim is to create a consistent environment across all the different contexts. So the rules for creating the environment for this project are as follows:

- Create a cycle
- Add random edges to make it more connected
  - Picking a random node with a degree less than 3
  - Add an edge between it and one node within 5 steps forward or backward along the primary loop. (So, node 10 might connect to node 7 or 15, but not node 16.)
  - Do this until no more edges can be added.

In this way, all nodes in the graph will have a degree of either 2 (because of cycle) or 3 (because of cycle & random edge).

2.2 Prey

Prey is a participant which moves freely in it's neighborhood in the given environment. Prey either wishes to move to one of its neighbors or choose to remain at its current location in all the settings that we are going to work on. Prey is represented by object of Prey class with encapsulating properties (location) and responsibilities (initialisation and moving).

2.3 Agent

Agent is also a participant trying to catch the Prey and avoids being caught by Predator. The whole project describes certain rules that dictate agent movement. Agent 1 and 2 are in the Complete Information Setting

2.4 Predator

The Predator here is the Distracted Predator, which will mostly try to move closer to the Agent with a probability of 0.6 and sometimes gets distracted and move to any random neighbor with probability 0.4.

Predator will mostly try to catch the agent by always trying to move closer to the agent. So the predator will compute the shortest path to the Agent at every step and will move in that direction. Predator is also represented by instance of it's encapsulated class Predator.

2.4.1 Shortest Path

For every game, we have to compute the shortest path between nodes several times. So I have implemented Floyd Marshall algorithm to compute the all pairs shortest path. I have pre-computed the shortest distance between nodes and stored it in a *self.distance* variable in Graph class.

2.5 Game

Game is a class which will co-ordinate the actions among the participants (Agent, Prey, Predator) in the given environment. Co-ordination in the given project will reflect the following:

1. Returning agent location to predator.
2. Returning predator location to agent in first two context settings, in which agent is certain about predator location.
3. Returning prey location to agent in contexts in which agent is certain about prey location.
4. Return neighbors of participant's location when asked by participant (agent, prey or predator).
5. Provide the information regarding the environment to participants when requested, such as degree for a node, shortest path between two specific nodes etc.

These functionalities are achieved by means of storing references of environment and all the participants in Game class. When an object created for environment, agent, prey and predator it will get registered in the Game object, and also store this Game object to request for information required at any given time.
Constructor will look like this for environment and all the participants ...

```
# Constructor for agent
def __init__(self, game: Game):
    self.game = game
    self.game.register_agent(self)
    ...
    ...
```

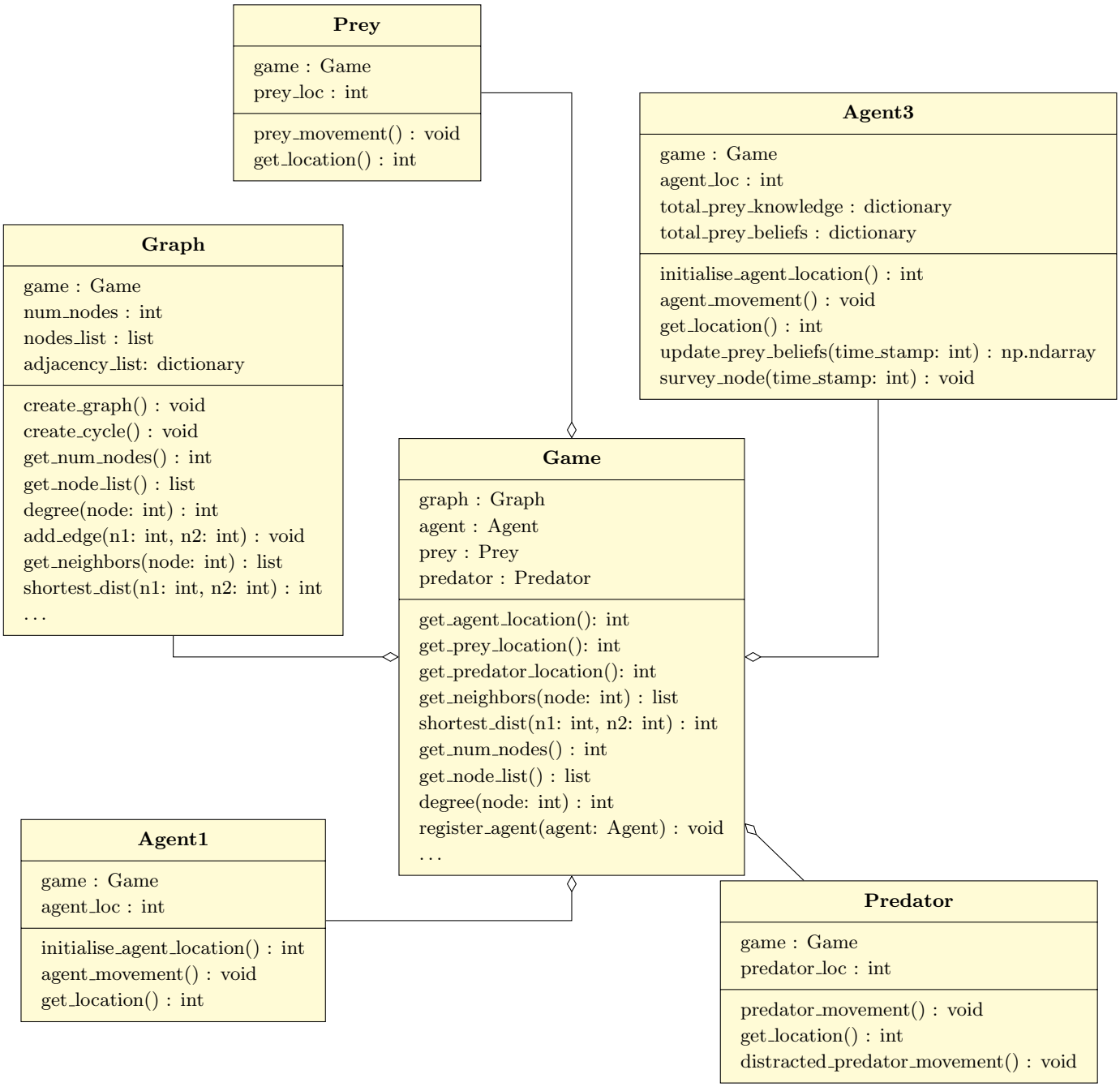
definition for game.register_agent() is given by

```
class Game:
    def register_agent(self, agent: Agent):
        self.agent = agent
```

In the similar fashion constructors for all the participants and environment created along with register methods in Game class.

Note:

So Before creating any object for agent, prey, predator and environment we should create Game object and pass the same Game object to all the participants and environment to get registered in game object.



2.6 Assumptions

- Prey and Predator can reside in the same node.

- Agent should initialise in node other than prey's and predator's initial location.

3 Implementation

MARKOV DECISION PROCESSES

This whole model can be formulated as a Markov Decision Process in the following way:

1. **Finite state space S, for every state** $s \in S$ are the positions of the players: Agent, Predator, and Prey at a given time k.

   In this project, States are taken as a tuple of (Agent position, Predator position, Prey position)

   > How many distinct states (configurations of predator agent prey) are possible in this environment?
   >
   > Since the environment is a graph of 50 nodes, and the agent, predator, and prey can be at any node, there are will, in total, 50*50*50 possible distinct states.

2. **Finite action set A(s), where taking action** $a \in A(s)$ is the Action space of the Agent throughout the game at each time step.

   The Action Space are defined by the Agent movements which we get from the Graph(edges that give the information about Agents Neighbours)

3. **Transition probabilities**, $P^a_{s,s'}$ is the probability for the player to jump from one state to the other.

   - The transition probabilities of the **Agent** is either **0 or 1**, as it just depends upon whether the agent will move to that node or not.
   - The transition probability of the **Prey** is the probability of the prey **to move to one of its neighbour**, that will be 1/(degree(prey) + 1)
   - The transition probability of the **Distracted Predator** to **move to one its neighbors.**

4. **Rewards / Cost**: The immediate cost taken when transitioning from one state to the other by the Agent.
   Agent when jumping from present state to the new state takes a potential cost ${C_s}^a$.

   ${C_s}^a$ - Cost to move to state s under Action *'a'*

   I have implemented the **minimization model - the Cost model**(instead of taking Reward).

   Here in the project, the immediate cost, ${C_s}^a$, is taken as 1.

In every time step, based on the current state (starting at some initial state), the agent will decide which of the available actions to take(i.e., which neighbors to move) to move towards the prey and avoid the predator, with a potential cost, and transition to the next state.

3.1 Utility U^*

At every time step, starting from state s, the 'value' of policy π is the immediate cost of the action taken under π, plus the expected future reward based on whatever state the system transitions to, discounted by a factor of $\beta \in (0, 1)$

What states s are easy to determine U^* for?

1. Agent, Prey and Predator in same node
   $U^* = \infty$
2. Agent and Prey in same node
   $U^* = 0$
3. Agent and Predator in same node
   $U^* = \infty$
4. Agent's neighbor has prey and not predator
   $U^* = 1$
5. Agent's neighbor has predator and not prey
   $U^* = \infty$

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

The relation is defined in the Bellman Equations.
The Agent has an action space, that has the agent movements from the current location to all its neighbours. To calculate the $U^*(s)$ for the state, it has to consider U^* of all the possible transition states it can have after taking an action a.

U*(s‘) is the Utility of the transition state. So the Agent will calculate the Utility of the present state by calculating the U*(s‘) of all the other possible transitions states using the Expected Future Cost of the state.

Expected Utility

The Expected Utility (Expected Future Cost) is the total summation of all transitions to the possible transition states in the next step, which was given by the **transition probability of the prey**, **transition probability of the predator** and the **Utility of the transition state**.

The Expected Utility of a state is computed in the following way:

$$E[U(s)] = \sum_{s‘} P^a_{s,s‘} U^*_k(s‘)$$

where s‘ are all the possible transition states in the next time step.

When the agent transitions from one state to another one, the optimal Utility is calculated as the **sum** of the **immediate cost** it takes for the agent for an action and the **expected utility** of that transitioned state.

To get the optimal utility for the agent, we can use the Bellman's Equations, which will give an optimal utility function.

The Utility for every state is given as:

$$U^*(s) = \max_{a \in A(s)} [C^a_s + \beta \sum_s {}' p^a_{s,s'} U^*(s')]$$

Here I have taken β **= 0.7**.
I felt 0.7 was a good value for β as it will allow to **discount the future transition Utilities**.

However, since we here consider only the transition states for the next timestep, we can have a higher β ranging from 0.5 to 1. So I choose 0.7 value, which gives enough priority to the next transition state Utilities.

VALUE ITERATION

To solve the above Bellman's equation, we use the **Value Iteration Algorithm**.
Under the technique of Value Iteration, we improve the intial guess as following:

1. Construct an initial estimate or guess for U^*, i.e., select some value $U_o^*(s)$ for each state s S.
2. Iteratively improve this estimate

$$U_{k+1}^*(s) = \max_{a \in A(s)} [C_s^a + \beta \sum_s {}' p_{s,s'}^a U_k^*(s')]$$

3. Conclude the iteration when we get a successive approximation for U_k converging to U*.

$$\max_{s \in S} |U_k^*(s)) - U^*(s)| \to 0 \; as \; k \to \infty$$

For every state in the environment where the Agent, Prey and Predator are configured, we need to evaluate Utility of each state. The final Utilities that we get are the optimal Utilities, $U^*(s), for states.$

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

1. If the Agent, Predator and Prey are spawned in the same location of the graph
2. If the Agent and Predator are spawned in the same location

Find the state with the largest possible finite value of U*

With a random graph, I can see that the highest Utility (when $\beta = 0.7$) is 3.33159. The highest Utility for the same graph when $\beta = 1$ is 26.2. The state which had the highest Utility was (35, 40, 45) .

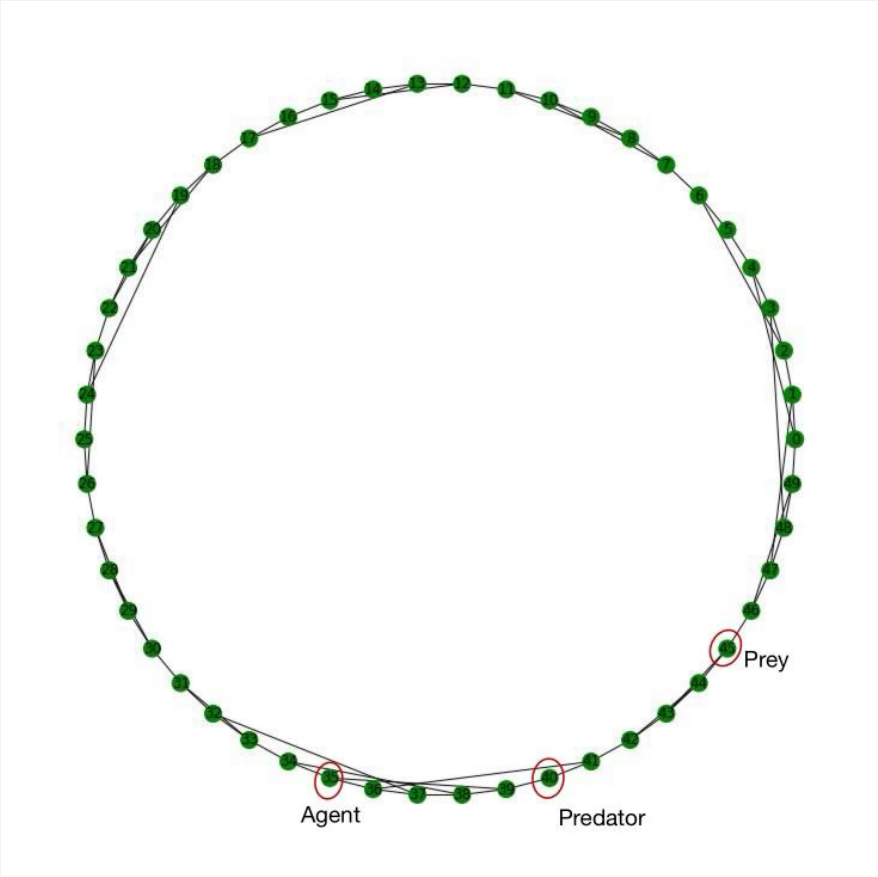


Figure 1: Highest Utility State

Here, we can observe that the Predator was in the the way of Agent while it tries to catch the Prey in the Shortest distance. So the Agent at this particular state has no other shorter way to reach the prey and had to take the longer way following the states having the minimal Utility. This is the reason why this state has the highest Utility value.

4 Complete Information Setting

In this setting, the Agent always knows exactly where the Predator is and where the Prey is.

4.1 U* Agent

As Agent is aware of both prey and predator location, it is straight forward implementation. Only concern is how agent is going to catch prey and escape from predator.

I am considering below events will place at given time sequentially

1. The action space for the Agent is defined by the number of neighbors the Agent has.
2. Agent 1 has knowledge of all the optimal Utilities U^* for all the states which are computed using Value Iteration Algorithm
3. So Agent 1 makes the next move based on the Optimal Utilities of the states having Agent's neighbors in the Agent location. Agent 1 takes action by considering the next state that has the minimum utility.
4. If Agent is present in state s= (AgentPosition, PredatorPosition, PreyPosition) at time k, then agent will check Utilities to decide for the state to transition at time $k+1$

   $\mathrm{U}^*(s1) = U^*(n1, PredatorPosition, PreyPosition)$

   $\mathrm{U}^*(s2) = U^*(n2, PredatorPosition, PreyPosition)$

   $\mathrm{U}^*(s3) = U^*(n3, PredatorPosition, PreyPosition)$

   where n1, n2, n3 are neighbors of Agent at time k
5. The Agent checks the minimal Utility among U*(s1), U*(s2), U*(s3) and moves to that state.
6. The Prey and Predator then move according to the rules of the game.
7. With the current state, the Agent again checks for the minimal Utility state (as done in Step 4) and continues moving to next state until it catches the prey.

4.1.1 U^* Agent results

	Agent	no_of_simulations	max_steps_allowed	Success rate	Failure rate	Hangs
0	U_agent	3000	100	0.9893	0.0103	1
1	U_agent	3000	150	0.9863	0.0137	0
2	U_agent	3000	200	0.9880	0.0120	0
3	U_agent	3000	250	0.9907	0.0093	0
4	U_agent	3000	300	0.9917	0.0083	0

Figure 2: U^* Agent results

1. U^* Agent has very high **success rate of 99 %**
2. The only failure case of U^* Agent is when the Predator is initialised at the same location as the Agent
3. If the Agent is not spawned at states which has Utility $\infty, then \mathrm{U}^*$ **Agent has 100 % success rate** at all cases.

4.1.2 Comparison of U* Agent and Agent 1 and 2 of Project 2

	U* Agent	Agent 1	Agent 2
(For 3000 simulations)			
Success Rate	99.1%	88.3%	97%
Average Steps taken to catch the Prey	14	46	49

Figure 3: U^* Agent Agent 1 and 2results

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?

- U* definitely takes the least steps to catch the prey because it has developed an algorithm which will help the agent to be able to make better choice while making next move by just checking the Utilities. This Agent is efficient and faster in catching the prey. Moving closer to the prey and not be caught by the predator is mathematically computed through the utility itself.
- Agent 1, takes longer as it has to catch the prey as well as the predator, and thus takes more steps than U*. However, for few situations, it is catches the prey in fewer steps that Agent 2.
- Agent 2 is mainly based on catching the prey using the probability of catching a prey in future. So, since the Agent 2 is based on probability, it sometimes end up having more moves to catch the prey that the Agent 1.

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

If the agent is at a location where it has predator at distance 2 from it, then Agent 1 would run away, but U* agent will move if the neighbour has the minimum utility, irrespective if there is a predator in the neighbour of that neighbour. Agent 2, however similar to U*, would have calculated the probabilities beforehand, and move according to that.
Example:

if the state = (1, 4, 7), the agent1 will try to move to node 50 or any other node that has larger distance from predator.

One of difference in movements of the agents come where there is a predator at the proximity. The agents usually have different movement at this point.

5 V Model - The Neural Network Model

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

The state is usually represented as tuple for normal agents.
But for my model, I have taken each entity of the state, i.e., the locations of the players and passed each of them as a separate feature for NN my model. The rest of the features that are used as inputs are the distances between the players, the vertices of the graph, which will give the information about the neighbours of a particular player position which are defined clearly in the Agents section.

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

I have chosen Neural Networks as a model here because Neural Networks can approximate a wide variety of functions very well when given an appropriate combination value. I took a 3 layered Neural Network and used activation, weight initialization and loss functions
I am training it by creating an input data set which is drawn from the data during the game simulations which is explained in detail further.

5.1 Implementation of V Model using the Neural Network Model

I am developing the Neural Network Model to predict the values of Utilities. I have used Neural Networks as a model here because Neural Networks can approximate a wide variety of functions very well when given an appropriate combination value.

So here I have taken the input features from the graph(the data which is required), feed it to Neural Network and got the output with the Utility values.

5.1.1 Data Set Creation

Neural Network needs to be trained with the dataset that has enough data for the Neurons to perform computations to predict the output.

We need to keep in mind several factors while creating the data set for the input of the Neural Networks, like the **size of the data** and the data which we are taking as **features for the input data**.

Here, I am creating the data set while the Agent moves at every time step, and that data that is stored during the simulations is given as the input for the Neural Network.

5.1.2 STRUCTURE OF NEURAL NETWORK

Designing the Architecture includes defining the activation functions, loss functions, Parameter initialisation functions, computations in the forward pass and the backward propagation.

1. **Choosing the Activation Functions** Initially while building a neural network, it is important to find the appropriate activation function. Since, we want to get the single Utility values for a given state, this comes under Regression Problem. Linear Activation function will work for the output layer.

   Similarly there are several other activation functions that are used in the hidden layers in the project.

   (a) **Leaky Relu** : Used leaky relu because several input values which are less than 0, will be multiplied with a fixed scalar. This function has mean activation close to 0, which speeds up training.

(b) **Tanh** : Tanh helps in the backpropagation process. Gives output between -1 and 1. Tanh gives better results in Multi layer Neural networks

(c) **Sigmoid** : Sigmoid function is useful when we deal with probabilities as it gives output between 0 and 1.

(d) **Linear** : It is simply used to get the output using simple linear functions. These functions are also used as gradient activation functions which are used during the Backpropagation process.

2. **INITIALISING THE WEIGHTS AND BIASES**
   Main task of building a neural network is the initialization of parameters to achieve optimisation in least time and have faster convergence to minimum.

   To avoid vanishing and exploding gradients issue, I have defined several weight initialisation methods.

   There are few parameter initialisation methods defined in the project:

   - Random Initialization - Randomly initialising the weights
   - He initialization - The weights are multiplied with $\sqrt{\frac{2}{size^{[l-1]}}}$
   - Xavier initialization - Same as He initialisation, but only used with tanh activation function.

3. Forward Pass : Includes all the mathematical computations of the input sent that occur at the neurons using the weights, biases and activation functions

4. Loss Functions

   - Mean Squared Error
   - Root Mean Squared Error
   - Mean Absolute Error

5. Model Fitting This depends upon how we split the data into training and testing datasets. It also defines the size of batches in which the training is done in NN.

Is overfitting an issue here?

Since we are taking a particular graph and working on it, and since we will be using same model again for predicting the Utilities for the same graph,Overfitting will not be an issue here. However, it is wise to have a split in the data as training data and test data, atleast in the ratio 9:1 to have a better model.

How accurate is V?

The V model I have designed is pretty accurate as it is trained with 90 % of the data. The Mean Squared Error for the model was 12.265.

5.2 Inputs used for Neural Network in the Complete Information Setting.

To train the Neural Network to predict the Utility of a given state, i.e.,
State ≡ (Agent-loc, Predator-loc, prey-loc)

I have taken the following features, which gives sufficient information to the Neural Network to get the picture of the graph and positions of participants in the game at a given point of time.

1. Agent-location (1)
2. Predator-location (1)
3. Prey-location (1)
4. Agent to prey distance (1)
5. Agent to predator distance (1)
6. Prey to Predator distance (1)
7. All the vertices of the edges of the Graph (146)

We are considering all the 152 attributes and make it a **Datapoint** for the input data.

DATAPOINT :
[Agent-location, Predator-location, Prey-location, Agent to prey distance, Agent to predator distance, Prey to Predator distance, vertices of the edges of the Graph]

So several data points are created at every time step, as the agent moves.
e_1, e_2, ... $e_k \rightarrow$ k edges in the given graph
$e_1(n_1)$, $e_1(n_2)$ are vertices involved in edge e_1
$e_2(n_1)$, $e_2(n_2)$ are vertices involved in edge e_2
.
.
.

The above data has been generated after converging Utilities while calculating the U* of the states using Value Iteration Algorithm.

FILTERING THE DATA

Data has been filtered out based on Utility values i.e., states which are having Utility values as **float('inf')** are stored in **look-up table** and the corresponding states are not used for training purpose in the NN as the infinity values won't help in converging the NN weights.

Out of all the data points (125,000). I have used X_n data points for training and validation.

For training and testing purposes, I have taken the **test and training split as 1:9 ratio.** I have passed (99781, 152) size data set as input for the neural network as Training data set and I have taken (11087,152)

5.3 Architecture used for Neural Network

The main problem we are trying to solve here is the **Regression problem** using the finite domain of X, i.e., the states. A simple Neural Network will work fine for this problem and yield good results.

The Neural Network used here is having **3 hidden layers**, each having **4 neurons** and the last output layer with one neuron (since we are just predicting a single value, i.e., the Utility)

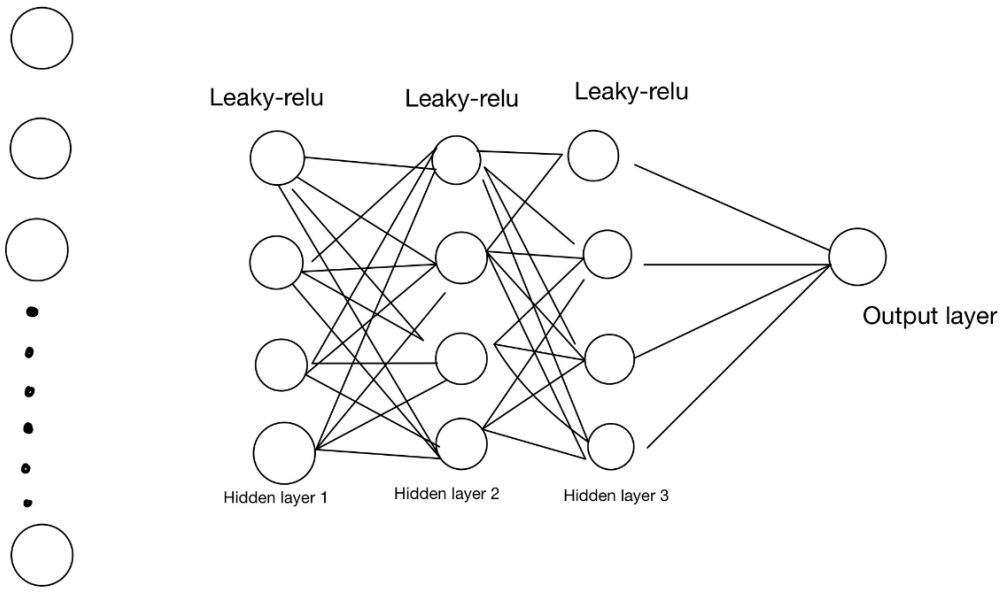


Figure 4: NN Model for U* Agent

5.3.1 Activation Functions Used

- **Leaky Relu** : For 3 hidden layers
- **Linear** : For the output layer

ERROR METRICS

- Loss Function = Root Mean Squared Error
- Training Loss = 0.4602447667161656
- Validation Loss = 0.3362941811892751

Learning Rate = 0.01

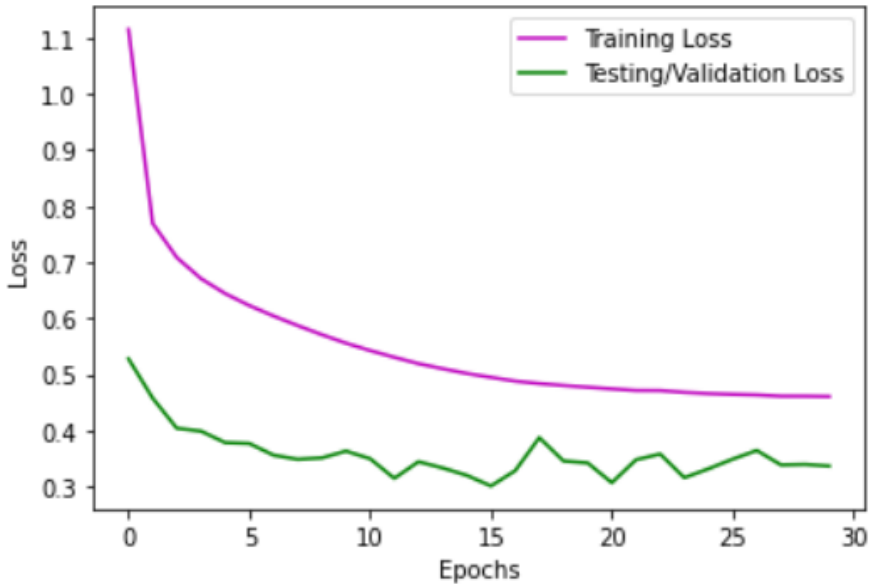


Figure 5: U^* Agent Error Convergence

No. of Features	152
Epochs for Training NN	30
Training Data Set	99781
Testing Data Set	11087
Activation Functions	Leaky-Relu
Error Function	Root Mean Square Error

Figure 6:

5.3.2 Comparison between U* and V

			U_agent	AgentNN
	no_of_simulations	max_steps_allowed	Success rate	Success rate
0	3000	100	0.9893	0.9810
1	3000	150	0.9863	0.9867
2	3000	200	0.9880	0.9877
3	3000	250	0.9907	0.9890
4	3000	300	0.9917	0.9893

Figure 7: U^* Agent and V agent

How does its performance stack against the U* agent?

Both the agents have almost equal with high success rate of 99This show the power of a good algorithm. The U^* Agent works with the same success rate as the Neural Network Model Agent.
So both the agents are equally efficient with high success rate.

6 Partial Prey Information Setting

In this scenario, the Agent is always aware of the location of the Predator but does not necessarily know the location of the Prey. Each time the Agent advances, it can first select a node to survey (anywhere in the graph) to see if the Prey is present. **Every time the Agent enters a node and the Prey isn't there, the Agent learns where the Prey isn't.** In this situation, the Agent must keep track of a belief state for the Prey's location, which is a set of probabilities for the Prey's presence at each node. **These probability need to be revised every time the Agent gains new information about the Prey. These probability must be updated whenever the Prey is known to move.**

6.1 $U_{partial}$ *Agent*

Since Agent 3 doesn't know the exact location of the prey, it develops a belief system to figure out the exact location of prey eventually. Here, the Agent 3 is designed in such a way that it will built the belief system ***beliefs***, and update it according to the actions taken by the agent (Same as the Agent 3 in Project 2).

Since we don't know the prey's exact location, we use the prey's beliefs in the states instead of prey location. The state is represented by

State ≡ (Agent-loc, Predator-loc, p) , where p is the prey beliefs at that time.

It is impossible to find Utility for every state, as there will be infinitely many states with the prey beliefs.So the agent will calculate the expected Utility for every possible next move and consider the minimum Utility among these Utilities to make the next move.

We calculate the expected Utility in the following way:

Since the Agent has no idea about the exact prey location, it will consider all the possible positions of the prey(All the 50 nodes) and calculate the Utility for all those state by using the **prey belief of that state** as the Transitional Probability for the Prey and **the Optimal Utility U* of that state**.

The expected Utility can be formulated mathematically in the following way:

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

So the Agent will calculate the Expected Utility for all the Agent Neighbors that it can jump to, and calculate the expected Utilities for all the neighbor states. The Agent will then pick the state which has the minimal Utility and make transition to it.

I am considering below actions will occur at given time T_t sequentially for **Partial Prey Information Setting** :

1. Agent surveys a node according to rules at T_t
2. Agent updates ***beliefs*** using the result of survey.
3. Agent calculates the expected Utility using the $beliefs$ and the **U*** of the states.
4. Agent moves to a new location which has the minimal Expected Utility.
5. Agent updates $\boldsymbol{beliefs_t}$ using it's new location at T_t
6. Prey moves at T_t
7. Agent updates the beliefs according to the transition model at T_t
8. Predator moves at T_t

Simulate an agent based on Upartial in the partial prey info environment case from Project 2 using the values of U* from above How does it compare to Agent 3 and 4 from Project 2? Do you think this partial information agent is optimal?

$U_{partial}$ has a very good success rate of 98.5 %, which is higher than Agent 3, Agent 4.
$U_{partial}$ is also able to find the exact prey location better than the Agent 3 and 4.

	Steps:	Upartial Agent	Agent 3	Agent 4
Success Rate	50	89.93	76.70	84.43
	100	97.10	82.80	91.07
	150	98.67	83.90	92.63
	200	98.67	81.97	92.1
	250	98.57	84.17	93.7
	300	98.67	85.21	93

Figure 8: Agent Upartial vs 3 vs 4

	Steps:	Upartial Agent	Agent 3	Agent 4
% of finding the prey	50	6.60	5.67	5.97
	100	6.89	5.80	6.12
	150	7.21	5.64	5.87
	200	6.98	5.79	5.84
	250	6.98	5.71	6.04
	300	7.42	5.73	5.79

Figure 9: Agent Upartial vs 3 vs 4

6.2 Neural Network for Partial Prey Setting

In Partial Information Setting, the state can be represented as:

State ≡ (Agent-loc, Predator-loc, $\underline{p}$) where p is the prey beliefs at that time.

How do you represent the states s-agent s-predator p as input for your model? What kind of features might be relevant?

I have taken s-agent, s-predator and p as separate features to the model. I have used the prey beliefs and also the node that has maximum belief of having prey.

To train the Neural Network for $V_{partial}$, I have given the following features as the input to the Neural Network.

1. Agent-location (1)
2. Predator-location (1)
3. Prey-location with the highest prey belief (1)(breaking ties at random)
4. Agent to prey distance (1)
5. Agent to predator distance (1)
6. Prey to Predator distance (1)
7. Prey beliefs (50)
8. All the vertices of the edges of the Graph (146)

All these attributes are used to make a Data-point for the input features.

[Agent-location, Predator-location, Prey-location, Agent to prey distance, Agent to predator distance, Prey to Predator distance, p - Prey beliefs vertices of the edges of the Graph]

There are in total 202 features used in the input data.
The training set of (2254615, 202) is fed to the neural network.

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

I am taking a Neural Network model with

- 3 hidden layers, an input and output layer
- each hidden layer having 4 neurons each
- First Hidden layer using tanh function
- The rest 2 hidden layers using leaky-relu
- output layer using the linear activation function
- Loss function : Mean Square Error

Data used for training the Neural Network, all the input features that are mentioned above, is collected while simulating the Agent for Partial Prey Information Setting, where the Agent uses the Optimal Utilites, i.e., the Utilties obtained after converging (U*(s)), and with the above architecture, it will train the data.

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

Overfitting is definitely an issue in the general cases. Here, since we are working on a single graph, overfitting does not harm much. But it will effect the model, when it is used to test new data from new graph. To overcome, we can first have a proper train-test data split ratio. Secondly, we can provide abundant data or data points to train the model. This will train the model over vast data points.

No. of Features	202
Epochs for Training NN	10
Training Data Set	2254615
Testing Data Set	250513
Activation Functions	Tanh, Leaky-Relu
Error Function	Mean Square Error

Figure 10:
Upartial Information

ERROR METRICS

- Loss Function = Mean Squared Error
- Training Loss = 0.0866406879640861
- Validation Loss = 0.07909306206967785

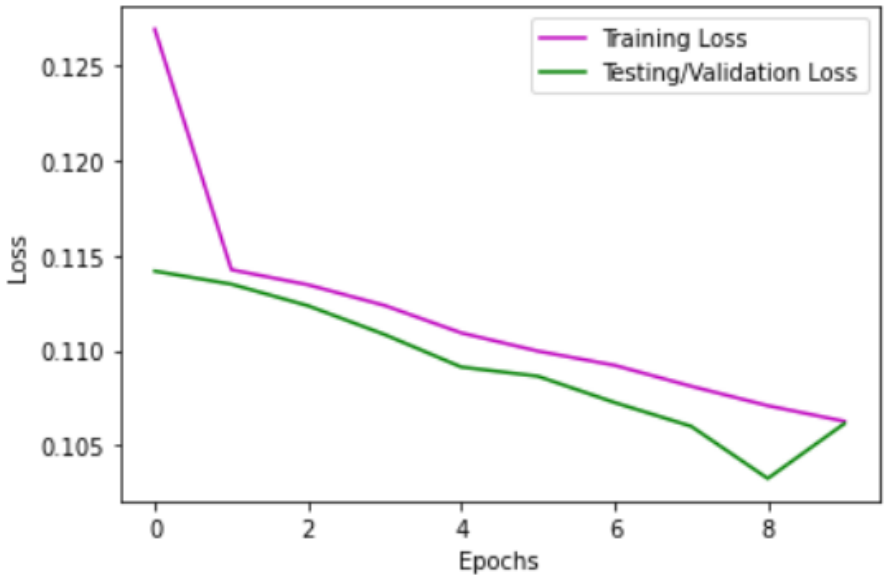


Figure 11:
MSE Error Graph

As the probabilities have low values, they are scaled (multiplied by 100) while passing through the Neural Network.

How accurate is Vpartial? How can you judge this?

$V_{partial}$ is around 98% accurate, as it has enough data to get trained and also enough data to test. The Mean Squared Error for this model was around (the Training Loss is 0.086) and (the Validation Loss is 0.079) after 10 epochs. So based on the MSE, I base that the accuracy of V$^{p}artial$

Is Vpartial more or less accurate than simply substituting V into equation (1)?

V partial is more accurate than just substituting it in equation 1.
Vpartial is a complete Neural Network model which has several functions to predict the Utility values. The Equation 1 is expected Utility values that depends upon just the prey beliefs and the transition probabilities.
Moreover on substituting it confirms the same that I have mentioned.

	Agent	no_of_simulations	max_steps_allowed	Success rate	Failure rate	Hangs	% knowing prey location	% knowing predator location
0	AgentPartialNN	3000	50	71.87	7.60	615	6.60	100.0
1	AgentPartialNN	3000	100	85.87	11.27	86	6.89	100.0
2	AgentPartialNN	3000	150	88.40	11.23	11	7.21	100.0
3	AgentPartialNN	3000	200	87.90	12.03	2	6.98	100.0
4	AgentPartialNN	3000	250	88.73	11.27	0	6.98	100.0
5	AgentPartialNN	3000	300	86.33	13.67	0	7.42	100.0

Figure 12:
Vpartial Agent

$V_{partial}$ is approximately closer to the success rate of the Agent 4 in the project 2.

7 Bonus 1

To calculate the actual utility of the $U^partial$ agent, we can use the Expected Future Outcome which we used for calculating for the $U^{partial}$ using the prey beliefs.

Again applying the Value Iteration Algorithm, and take the expected Utility which was used in the Partial Prey Beliefs.

$$U^*(s_{agent},\ s_{predator},\ s_{prey}) = \max_{a \in A(s)} [C_s^a + \beta \sum_s{}' \sum_{s_{prey}} p_{s_{prey}} U^*(s_{agent},\ s_{predator},\ s_{prey})]$$

1. The Agent before making the next movement, will consider all the possible states to transition and check the one having minimum Utility.
2. For each the neighbour, it will calculate all the probabilities of a prey being in the transition state multiplied with the Utility of that transition state.
3. After that, the agent with the probabilities calculates for all the possible transition states that the agent can transit in the next time step.

In this way the agent can calculate the possible actual Utilities of the states.