RUTGERS

CS520

Project 3: Better, Smarter, Faster

Kunjan Vaghela (kv353)
Rutgers University

Manan Shukla (ms3481)
Rutgers University

Mitul Shah (ms3518)
Rutgers University

December 14, 2022

Contents

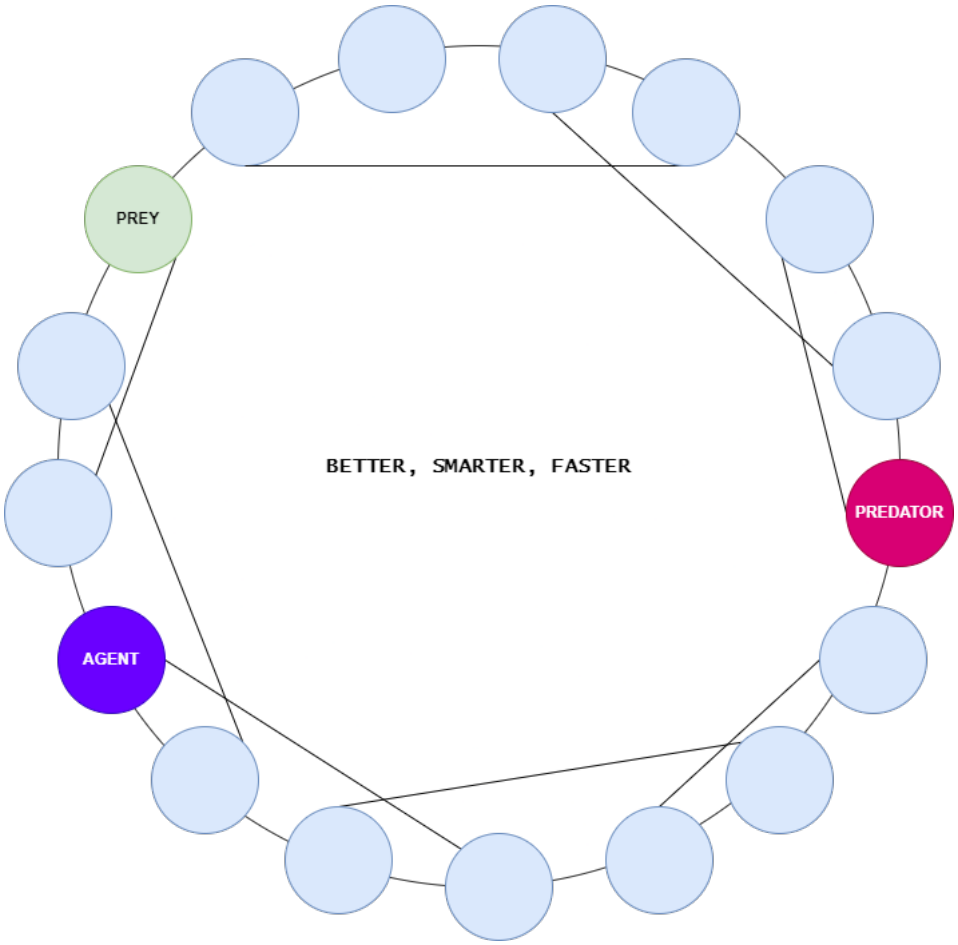
1

Introduction

1.1 Problem Statement

The environment is a Graph with 50 nodes connected circularly with randomly added additional edges/ paths between nodes. Random nodes are added with a constraint that a node can have a maximum degree of 3 with no repeating edges between nodes and can connect only with a node within 5 surrounding nodes (in each direction). There are 3 moving entities in the environment: the Prey, the Agent, and the Predator. Agent can't be spawned in the same cell as the Prey and the Predator. The Prey's goal is to evade the Agent indefinitely until the Agent fails (captured by Predator). The Agent's goal is to catch the Prey and be safe from Predator. For the Predator, it is to capture the Agent. Hence, the Agent has to hunt for the Prey and ensure he's safe from the predators' attack. It loses if the Agent is caught by the Predator (the Agent and the Predator are in the same node). Agent wins when it successfully catches the Prey without being captured by the Predator. The movements at each timestamp are random, which introduces uncertainty. The main thought behind this is Building a probabilistic model of an uncertain environment, which will then guide and inform decision-making. Agents won't be able to see where the Predator and Prey are and will have to make a probabilistic model and infer. Insights are to be drawn after collecting performance data depending upon the constraints applied to Agents and their implementation.

Problem Representation:



1.2 Problem Environment

The environment is a Graph with 50 nodes connected circularly. These nodes have a maximum degree of 3. Each node, apart from having 2 edges contributing to the circle formation, can have a random additional edge with any node within 5 nodes in each direction (clockwise or anti-clockwise). This addition helps make the environment space challenging with the arbitrary action of the moving entities. The 3 objects in the graph space can move randomly from one node to another if there is an edge between them to traverse. At first, the Agent moves, then the prey, followed by Predator.

1.2.1 The Prey:

Surviving is the sole purpose of Prey. When it is Prey's turn, the probability of traversing to any neighboring nodes is equal. For example, if there are 3 nodes, then the chance the Prey will go to any one of the nodes is 1/3 (33%).

1.2.2 The Predator:

The predator's involvement is similar to that of Ghosts from 'Project 1- Ghosts in a Maze'. The predator hunts for the Agent and wins when it successfully captures the Agent (reaches the Agent's node). What's implicit is that the predator is unsuccessful if the Agent completes its task of capturing the prey. Predator's movement is inspired by its goal of capturing the Agent, and randomness is introduced. It calculates the minimum distance from each of its neighboring nodes to Agent. Then has a 60% probability of following that route and a 40% probability of choosing any of its neighbors at random and proceeding with it.

1.2.3 The Agents:

As mentioned earlier, for each decision taken, the Agent has to meet 2 goals viz.

- Capturing the Prey.
- Staying away from the Predator (Staying Safe).

6 Agents are classified depending upon their movement, which is dictated by the strategy implemented; they are:

  - U* Agent
  - V Model - Agent
  - $U_{partial}$
  - $V_{partial}$
  - $V_{partial}$ 2.0
  - Bonus

1.2.4 The Environment Information Settings:

- The location of the Predator is always known to the Agent.
- Information about the location of Prey varies from Agent to Agent.
- Refer the table below for the Prey and Predator Position awareness summary.

Environment	Prey Position	Predator Position
U*	✓	✓
V Model	✓	✓
$U_{partial}$	✗	✓
$V_{partial}$	✗	✓
$U_{partial}$ 2.0	✗	✓
Bonus	✗	✓

1.2.5 Parameters

The Maze generated is based on the following parameters:

- Number of Nodes: Number of vertices in the environment graph [50]
- Degree: Maximum number of edges a node can have [3]
- Graph Layout : Circular [1st Node connects to 2nd node, 2nd node with 3rd node, so on and 50th Node connects back to 1st Node]

Environment Summary and Implementation used from 'Circle of Life'.

2

Implementation

2.1 The Environment

For implementing the Undirected Circular Layout Graph Environment, we have used an Adjacency Matrix implementation with 50X50 dimensions. We know that the Adjacency Matrix of undirected graphs forms a symmetric adjacency matrix. Here the index of rows and columns indicates the number of nodes. If the element's value in the matrix is 1, it indicates an edge between the nodes represented by the index of the adjacency matrix. For example, if the element adjacencyMatrix[12][16] has a value of 1, it means that there exists an undirected edge from Node 12 to Node 16. We had to implement the graph in a circular manner, meaning that the first node should be connected with the second, second with the third, and so on, to the end, where the last 50th node connects back to the first node. On top of this, we had to add more edges until the degree of each node was at max 3. Also, adding an edge randomly with the constraint can only connect the node with another node in the range of 5 nodes.

Environment Example for reference:

$$\text{Graph} : \begin{bmatrix} [0] & [N1] & [N2] & [N3] & [N4] & [N5] & [N6] & [N7] & [N8] \\ [N1] & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ [N2] & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ [N3] & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ [N4] & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ [N5] & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ [N6] & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ [N7] & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ [N8] & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Graph G: [V, E]
Vertices: N1,N2,N3,N4,N5,N6,N7,N8
Edges: [N1,N2], [N2,N3], [N3,N4], [N4,N5], [N5,N6], [N6,N7],
. [N7,N8], [N8,N1], [N1,N6], [N2,N4], [N3,N8], [N5,N7].

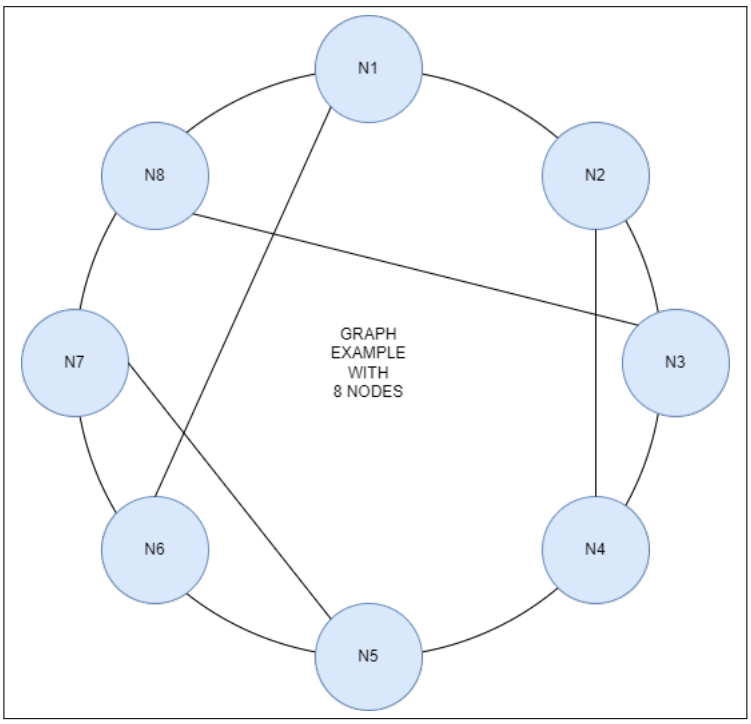


Figure 2.1: Graph Representation

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

After looking at the constraints, we can have at most 50 choices of Agent, 50 of Prey, and 50 of Predator. The maximum number of possible distinct states would be 50* 50* 50 = 125000 states.

2.2 Probability:

2.2.1 Prey Updation Equation:

$P(in\, X_{t+1})$ [Probability that Prey is in node X at the time 't+1']

$= P(in\, X_{t+1} \wedge in X_t) + P(in\, X_{t+1} \wedge \neg in\, X_t)$

$= P(in\, X_t)$ * $P(in\, X_{t+1} \mid in\, X_t) + P(\neg in\, X_t)$ * $P(in\, X_{t+1} \mid \neg in\, X_t)$

$= P(in\, X_t)$ * (1/k+1) + $\sum_{i=1}^{k}$[P(in i_t) * $P(in\, X_{t+1} \mid in\, i_t)$]

$= [P(in\ \ X_{t+1})$ / (k+1)] + $\sum_{i=1}^{k}$[P(in i_t) / $(m_i + 1)$]

[Where m_i *is number of neighbors of* i_t *AND* k *is number of neighbors of* X]

Distribute Probability from cell:

$$P(in\, X_t \mid Entity\, not\, present\, in\, i_t)$$

[Effect on Probability Belief when the entity is not found in a given cell (either by Survey or by Agent movement)]

$$= P(in\, X_t \wedge Entity\, not\, present\, in\, i_t)/P(Entity\, not\, present\, in\, i_t)$$

$$= [P(in\, X_t)\ *\ P(Entity\, not\, present\, in\, i_t \mid in\, X_t)]/P(Entity:\ not\, present\, in\, i_t)$$

$$= [P(in\, X_t) * 1]/[1 - P(in\, i_t)]$$

$$= [P(in\, X_t)]/[1 - P(in\, i_t)]$$

2.3 Markov Decision Process (MDP) Configuration:

MDP Property	Our Use
States	We defined a state as a tuple of (Agent Position, Prey Position, Predator Position).
	In total, we have 50*50*50 states, that is 125000 states in total.
	Since a node can have a maximum of 3 connections (3 neighbors), there are 48 neighboring nodes possible for transition from any given state based on the following:
	1) Agent can stay in the same node or can move to another neighbor node, thus in total 4 actions possible
	2) Prey can stay in the same node or can move to another neighbor node, thus in total 4 actions possible
	3) Predator can move to another neighbor node, thus having 3 actions possible.
	By permutations, there are 48 states possible to transition from any given state (4 * 4 * 3).
Transitional Probabilities	It is possible to go from one state to another state with some given probability. The probability of ending up in another adjacent state is defined by:
	P(getting to state y from x) = P(Prey moving from state x to state y) * P(Predator moving from state x to state y)
	Furthermore, from any given state, there are in total 48 transitional states possible. This can be calculated by taking following movement probabilities into consideration:
	1) Agent Movement - Agent can move to one of the next nodes or can stay in the same node based on the policy defined.
	2) Prey Movement - Prey can move to one of the next nodes or can stay in the same node with probability 0.25.
	3) Predator Movement - Predator can move to the next node with shortest path to Agent by 0.6 probability and by remaining 0.4 to one of the random neighboring nodes.
Actions	From any given state, Agent can move to another state by taking an action. Taking which action to take from a given state is defined by the policies defined.
	One action can lead the agent to 12 states (4 due to Prey's movement * 3 due to Predator's movement).
	In order to determine which action to take, Agent will take the action which will transition Agent towards a next cell with highest Utility value.
Reward	There are 3 rewards defined:
	1) Reward of Agent with Prey = 0
	2) Reward of Agent with Predator (and when Predator is in next node from Agent) = -inf
	3) Reward of other states = -1
	These rewards are propogated over the graph node to calculate the (optimal) U* value. Reward of each action taken is also defined by 'Reward of other states'.

2.4 Agents

2.4.1 U* Agent:

Path Planning and Algorithm:

- Environment initiated with agent, prey, and predator spawned in one of the nodes.
- Repeat the below steps until the conclusion:

1. Initialize States
2. Create Environment

3. Calculate Transition Probabilities in between each adjacent state based on Predator and Prey Transitional Probabilities
4. Transitional Probability from State x to State y = P(Prey Movement from x to y) * P(Predator Movement from x to y)
   (a) Get all the next states of a state.
   (b) Get the transitional probabilities of prey and predator movement to these neighboring states from the current state.
   (c) Assign transition probability to the next state based on the formula above.
   (d) Repeat this for each possible state.
5. Calculate the Optimal Utility of the states.
   (a) For each state, calculate utility at time t using the given formula
   (b) Check if the error between a state's utility value at time t and at time t-1 converges with 0.000001 error. If no, repeat 4.a
6. Run the agent with the policy of selecting the next action as one of the next actions available to the agent from the current node, with the maximum utility value calculated in the above step.
   (a) Agent moves according to the policy determined.
   (b) Prey and Predator move according to their movement behavior.
   (c) Check if Agent won or lost; continue the agent run if not.
7. Check if Agent wins.

Question. What states s are easy to determine U* for?

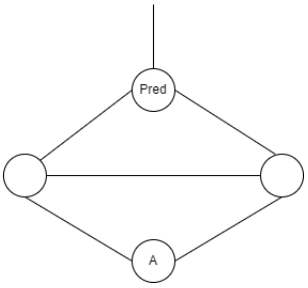
- The states where Prey and Agent are in the same position will have U* as 0. (As the Minimal expected number of rounds to catch the prey is 0) In the states where Predator and Agent are in the same position, it will have U* as '-inf'. This is because the agent will never go to the predator state and, thus, should be taken '-inf'.
- U*, as per definition and formula, will range from 0 to -inf for all these states. A negative of the U* will give the minimum expected number of rounds to catch the prey. The states with Agent as one step away from the prey on taking any (of the possible) actions should also have the utility of the -1. (As the Minimal expected number of rounds to catch the prey is 1)

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

- $U_t(s) = \max_{a \epsilon Action(s)} \sum_{s'} T(s,a,s') * [R(s,a,s') + \mathrm{U}_{(t-1)}(s')]$

- $U_t(s) = R(s) + \max_{a \epsilon Action(s)} \sum_{s'} P^a_{ss'} U_{t-1}(s')$

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



(a)

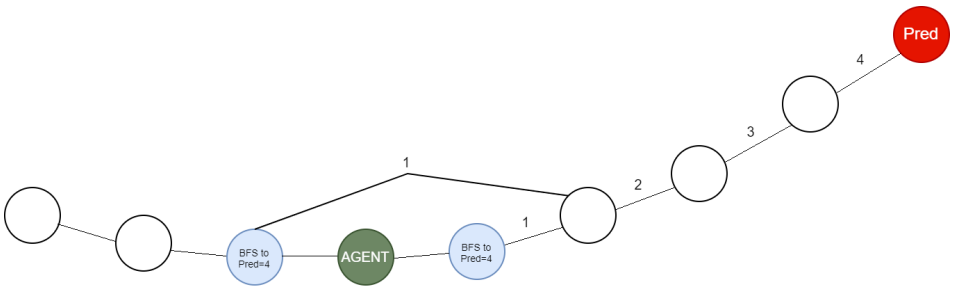
Figure 2.2: (a) Agent 4 Success Rate

Assume Prey is more than 2 to 3 nodes away from Agent. When an agent is spawned very close to the predator, and if the distance between the predator and the agent is 1 at t and t+1, then the agent will be killed by the predator, no matter where the Agent goes. The same is shown in the pictorial representation attached above.

Question. Find the state with the largest possible finite value of U*, and give a visualization of it.

- The state with maximum largest finite value of Utility is the state Where: Predator and Agent are very close to each other, but the Prey is at the farthest distance from it.
- As Predator is close to the Agent, Agent has to prioritize running away from the predator and than at later point catch the Prey.
- Thus, it will take more steps to catch the Prey.
- This fully justifies the definition of the Utility which is the expected number of rounds to catch the Prey.

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.



(a)

Figure 2.3: (a) Agent 4 Success Rate

- We have identified the scenario depicted above where U* and Agent 1 choose a different course of action.
- Agent 1's rule is to move to its neighboring node which has a greater or equal BFS distance to Predator from the current position.
- BFS length from Agent's current position to Predator is 5.
- BFS length from Agent's neighbors to Predator is 4, which is not greater than or equal to the current BFS length to predator, instead is less compared to the current BFS length.
- This is why Agent 1 is stuck in the same position and is in a Deadlock.
- U* Agent's policy is to choose an action that maximizes utility, so depending upon the values, it can make the correct decision and refrain from ever being part of the above-said Deadlock.

2.4.2 V-model Agent :

Neural Network Model V:

- Input: State, Dist(Agent,Prey) , Dist(Agent,Predator), Utility
- Output: Utility
- Model Type: Regression Model
- Training Algo: Backpropogation in Neural Network
- Metric: Mean-Squared Error (MSE)
- The optimal Utility obtained for each state is given input to the model.
- Hand-made features such as:
  1. Distance between Agent and Prey AND
  2. Distance between Agent and Predator are calculated.
- The above 2 features are used as the input to the model.

- State columns (Col 1, 2, 3) are removed from the features from the model, because this column acts as an index column, and we do not want over model to train based on this. State is not a feature on which we want our model to be biased on. Therefore, it is not used as an input for the modelspace.
- Data is split randomly in 0.2 ratio in test and train set.
- Mean and standard deviation is calculated for each column in the train dataset.
- TrainData is normalized by subtracting the mean and dividing by standard deviation of each column.
- The mean and standard deviation obtained for each column in the train dataset is used to also normalize the test data.
- Neural network model is made with various combinations of layers:
- Dense layers are used inorder to connect with each each layer of Neural Network.
- Activation functions which were experimented were:
    1. ReLU (Rectified Linear Unit)
    2. Sigmoid
    3. Tanh
    4. Linear

Table: Determining the Number of Hidden Layers

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

- The decision on how many hidden layers should be present is based on the above reference obtained from:
https://www.heatonresearch.com/2017/06/01/hidden-layers.html

- The last layer has been given only one neuron as this is a regression problem and we need only number as the output of the model.
- Also, the activation function used in last layer was linear, because we do not want any restriction on the output obtained in that node. Whatsoever is the output should be obtained.
- Normalization on the dataset increased the accuracy of the model.Reducing learning rate to 0.01 worked for our case.Fixing the random_state to a constant number.
- Metric used is MSE (Mean-Squared Error). If the U* value of a given state changes by a very small amount it would not affect much in the policy (decsion making) of the agent.
- But, if the U* value changes by a drastic amount, it definitely will impact the policy of the agent.
- Thus, we want to punish the high values of the Utility (U*), so we have used MSE as an error metric.
- Combination giving least MSE for Model V:

Layer	Activation Function	No. of Neurons
1	Sigmoid	2
2	Sigmoid	4
3	Sigmoid	2
4	Linear	1

- Combinations tried with Model V:

No. of neuron in each layer	Activation Function	Error	Input Data
(2, 32, 32, 1)	Relu	10.2	Unnormalized
(16, 32, 32, 1)	Relu	8.98	Unnormalized
(4, 16, 8, 1)	Relu	5.2	Normalized
(2, 4, 2, 1)	Sigmoid	3.23	Normalized

- The Model V developed above is to be used for the a fixed graph setting.

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

- State can be given in 3 different columns as the input, each representing Agent, Prey and Predator position respectively.
- Hand-made features used such as:
  1. Distance between Agent and Prey
  2. Distance between Agent and Predator are calculated.

- Also, taking features in this way makes the data continuous, so the problem becomes mapping from continuous input space to continuous output space. (Func f: R2 -> R)

Question. What kind of model are you taking V to be? How do you train it? Is overfitting an issue here? What can you do about it? How accurate is V? How can you judge this?

- V has to be a regression model which can have capability of modeling non-linear data.
- We know that Utility is not a direct linear combination of the above 2 features, so the function space which needs to be modeled is non-linear and thus Neural Network will be able to model this Utility values well.
- Here, we do not observe Overfitting, as the train error and the test error is almost same. And test error never surpasses train error.
- Also, even if we have overfitting, it would not be a problem here as we are building the model for a fixed graph and not in a graph agnostic way.
- Furtermore, The model does generalize well and the result obtained when used by the Agent as a policy gives same survivability and also almost the same number of expected steps.
- MSE metric obtained on data is 3.23.
- The model does generalize well and the result obtained when used by the Agent as a policy gives same survivability and also almost the same number of expected steps.
- Avg Number of steps: 9.76
- Survivability: 99.84

2.4.3 $U_{partial}$ Agent:

Algorithm:

1. Initialize States, Create Environment
2. Calculate the Optimal Utility of the states.
   (a) For each state, calculate utility at time t using the given formula
   (b) Check if the error between a state's utility value at time t and at time t-1 converges with 0.000001 error. If no, repeat 2.a.
3. Initialize PreyStateBelief with the belief of having Prey in all other nodes apart from current Agent Position as Prey position is not known and the prey can't be in the agent's current location.
4. Agent run - agent moves with the policy of the $U_{partial}$:

(a) Survey a node and update the PreyStateBelief according to Survey results.

(b) Predator position is known, check the next actions possible for the Predator with each action's transitional probability.

(c) For each state of Agent actions and Predator positions (as calculated from 5.a), calculate (Transitional Probability of Predator to that cell * $U_{partial}$ (as per formula given)).

(d) According to the policy, Agent takes the next action with max value as calculated in 5.b.

(e) Prey moves according to their movement behavior, PreyStateBelief updated.

(f) Predator moves according to their movement behavior.

(g) Check if Agent won or lost; continue the agent run if not.

$U_{partial}$ is not an optimal agent and its analysis presented in the last section

2.4.4 $V_{partial}$ Agent :

Question. How do you represent the states s_{agent}, $s_{predator}$,p as input for your model? What kind of features might be relevant?

- State can be given in 2 different columns as the input, each representing Agent and Predator position respectively.
- Features used:
  1. Distance between Agent and Predator AND
  2. Expected Distance between Agent and Prey
- It is calculated based on:
  Sum of (Prey belief * Distance of that node)

Question. What kind of model are you taking $V_{partial}$ to be? How do you train it? Is overfitting an issue here? What can you do about it? How accurate is $V_{partial}$? How can you judge this?

- Vpartial also has to be a regression model that models non-linear relationships.
- Utility Partial (UPartial) is not a direct linear combination of the above 2 features, so the function space which needs to be modeled is non-linear and thus Neural Network will be able to model this UPartial values well.
- The data collected for this model is based on the steps the Upartial agent takes while doing the simulation.
- The same strategy of doing normalization and using different activation functions is applied in modeling Vpartial also.

- Overfitting could be an issue here.
- We need model that generalizes well and works on the input space that it has not seen during the training of the model.
- On doing the simulation of the agent on the runtime, generalization of Vpartial model has been validated.
- We are able to see the the model with the lowest training loss is not working the best.
- The model with a little higher training loss worked well in the Vpartial agent simulation.

Layer	Activation Function	Number of Neurons
1 - Input Layer	Sigmoid	2
2	Sigmoid	4
3	Sigmoid	8
4	Sigmoid	8
5- Output Layer	Linear	1

- Accuracy of Vpartial model cannot be just judged by the MSE. Because the test set does not contain the whole possible combinations of the state.
- We need to do the simulation of the agent and compare the accuracies of the Vpartial agent and thus find the test error. MSE obtained: 6.24

Question. Is $V_{partial}$ more or less accurate than simply substituting V into equation (1)?

- Substituting V in the equation (1) to calculate Uparital, almost gives the same performance as Vpartial agent.
- It takes on avg 1 step less than steps taken by Vpartial agent.

Question. Simulate an agent based on $V_{partial}$. How does it stack against the $U_{partial}$ agent?

- Vpartial almost performs similar to Upartial agent.

2.4.5 Bonus:

BONUS 1: Build a model to predict the actual utility of the $U_{partial}$ agent. Where are you getting your data, and what kind of model are you using? How accurate is your model, and how does its predictions compare to the estimated $U_{partial}$ values?

- Predicting actual utility of the Upartial agent can be a difficult task as there are infinite no.of states possible.

- So, we have built the Neural network model with some added features in order to build an optimal agent.
- Vpartial model cannot take U* value as an input feature for it's modeling and predicted Upartial.
- So, we can go for building the model which can take following features:
- Features used:
    1. Distance between Agent and Predator AND
    2. Expected Distance between Agent and Prey It is calculated based on: Sum of (Prey belief * Distance of that node)
    3. Expected Utility of Prey. Sum of (Prey Belief * U* of that node)
- Performance of Bonus1 agent is little better than Vpartial2 agent.

3

Analysis

Question.
How does its (V Model Agent) performance stack against the 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?

Question.
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?

Agent	Survivability (%)	Avg. Number of Steps
Agent 1	89.1	15.6615
Agent 2	99.96	18.0429
Agent 3	89.13	30.5746
Agent 4	99.94	27.3865
Agent U*	99.90	9.31
Agent V	99.84	9.76
$U_{partial}$	99.83	40.63
$V_{partial}$	99.76	41.21
$V_{partial}\,2$	99.79	40.95
Bonus 1	99.71	40.32

- U* is the optimal Complete Information Setting agent. It takes a minimum number of steps to catch the Prey. And survives maximally
- V model agent tries to mimic U* agent by finding the value using a NeuralNetwork model. And is able to do it very well. It just has a little extra avg. no. of steps.

- $U_{partial}$ is not an optimal Partial Prey Info Setting agent. $V_{partial}$ is able to mimic it well using Neural Network model.
- $V_{partial}$ 2 agent also performs similar to Upartial, by plugging in value of V in Upartial equation.
- Project2's Agent 1, 2, 3, 4 accuracies drop with very few % with the easily distracted predator.