Project 3: Better, Faster, Smarter

Lab report

1. The Environment

The environment is made by using a graph of nodes that are connected by edges. This graph is circular with 50 nodes and the edges are added at random for increased connectivity.
The graphs are made using the python library networkx. This library is used to create and manipulate the circular graph—functions like add_node(), add_edge() etc are used for this purpose. New edges are added to the circular graph by selecting at random a node with a degree less than 3. Edges are added to the 5 steps forward and backward nodes of this random node. This process is repeated till no more nodes can be added. We have made one graph and stored it in `myGraph.gpickle` file. We use this graph for all our agents..

Q. How many distinct states (configurations of the predator, agent, prey) are possible in this environment?
The number of states possible of the prey, predator and agent is 50 each. So the total distinct configurations that are possible are 50*50*50 = 125,000

Q. What states S are easy to determine U∗ for?
The states that are easy to determine for U* are:

- If the agent, prey and predator are in the same node, the utility and rewards are set to 0.
- If the agent and the prey are in the same node, the utility and rewards are set to 0.
- If the agent and predator are in the same node the utility is 0 and the reward is infinity.

2. Write a program to determine U∗ for every state s, and for each state what action the algorithm should take. Describe your algorithm in detail.

The program we have written to determine the value of U* for every state S is done using value iteration. Its process is as follows:
First, we start by initializing utility and rewards for each state. It is done as follows:

- If the agent, prey and predator are in the same node, the utility and rewards are set to 0.
- If the agent and the prey are in the same node, the utility and rewards are set to 0.
- If the agent and predator are in the same node the utility is 0 and the reward is infinity.

Then we apply value iterations to find the utility by using the following formula:

$$V_{K+1}(X) = \max[R^{a}_{x} + \beta \sum_{X'} P_{x\,x'}{}^{a}\, V_{k}(X')]$$

Shushang Nair, sgn40
Soumya Tyagi, st1108

There are 4*3*3 possible neighbouring states from a particular state. And for one move of the agent, there are 12 possible states of the prey and predator (i.e. 4*3). We then proceed to compute the prey and predator probability for each state utility. The two are then multiplied by each other and since there are 12 probabilities and 12 utilities all of them are multiplied and summed up together. This process is done for all 3 possible moves of the agent and individually added to a list.

While computing the utility, if we find that one of the neighbour nodes of the agent has the prey in it, the agent should always choose to move to that node. Hence, we update the list to set that state's utility to 0. On the other hand, if the predator is present in any of the neighbour nodes of the agent, this means the utility of that state has to be set as infinity in the list as the agent should never choose that node.

Once we have updated our list, we will have three utilities of the three possible agent states, in the end, the minimum of the 3 in the list is considered to be the utility. If the agent neighbour consists of the predator we set the utility of that state as infinity as the agent should never choose the node with the predator in the neighbouring nodes.

We then store this utility file to `utility_dict_1st_graph` and use this utility for all our further tasks.

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

By using value iteration, we essentially add the rewards of that state to the minimum utility of the neighbouring states. By applying value iteration, the nodes that are the neighbours of the prey are computed to be 1 as the rewards are set to 1 in this case. And the nodes with the predator in the neighbour, the utility of such nodes comes out to be infinity. Hence, the further we move from the prey or the closer we move to the predator, our utility increases.

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

There are certain starting states where the agent will not be able to capture the prey. If the predator and the agent are spawned right next to each other, all the neighbouring nodes will have an utility of infinity and hence the agent can't decide what step to take next. We have defined an edge known as the overlap edge. Here, the previous and next nodes of the node in between are directly connected by another edge. If the starting node is an overlap edge where the agent is in one of these nodes and the predator is also in one of these nodes, in this case, the agent cannot escape wherever it goes and hence there is a failure.

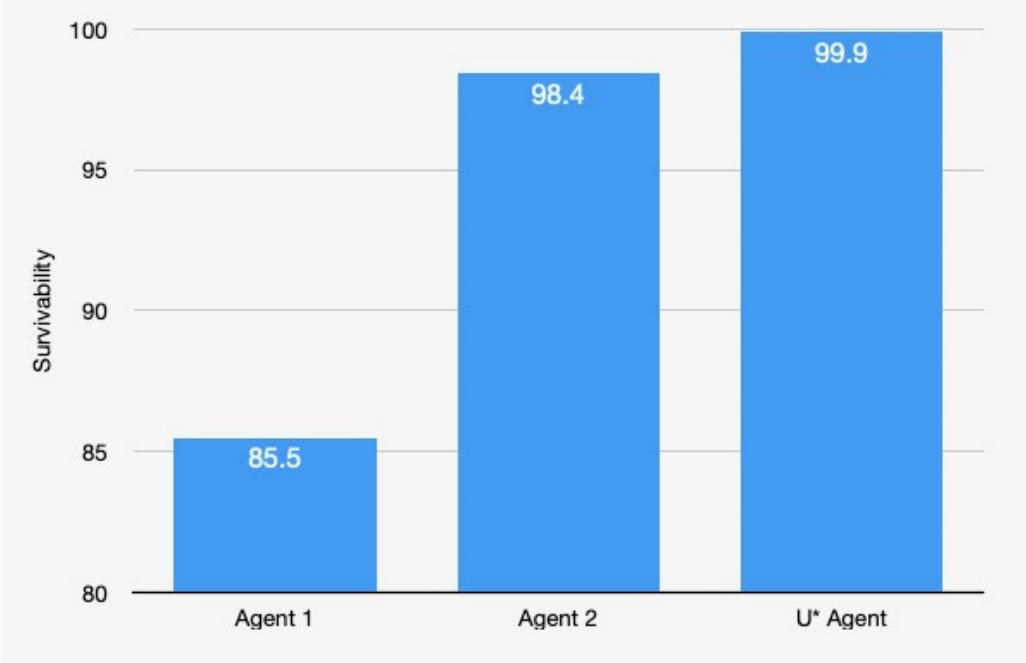
Q. Find the state with the largest possible finite value of U*, and give a visualization of it.

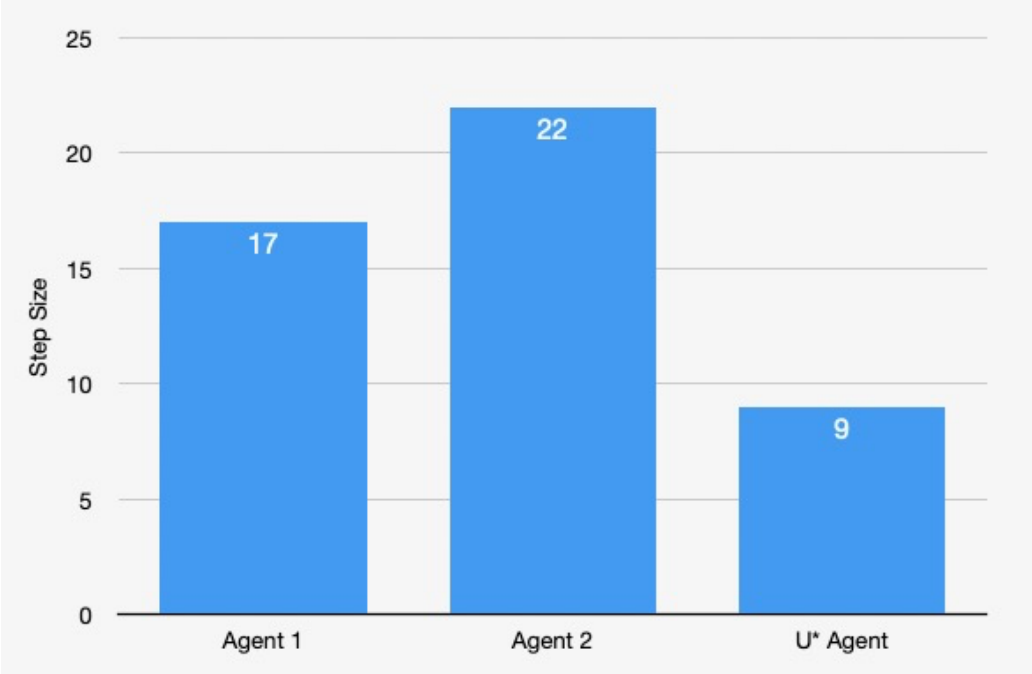
When prey and predator are on the same node and the agent is in close vicinity. The agent can't directly catch the prey, so essentially it has to circle around the whole graph dragging the predator along with it and then catch the prey. So these states have maximum utility. Below attached screenshot shows an example of the same. The odd lines are the position of agent, prey and predator respectively and the even lines contain the utility of the neighbouring nodes which tells the agent what step to take next.

```
Agent Loc Prey Loc Predator Loc 50 47 47
{49: inf, 1: 22.99719557737593, 46: inf}
Agent Loc Prey Loc Predator Loc 1 48 45
{2: 20.630343704813395, 50: 21.861466707449896, 6: 21.75530728896349}
Agent Loc Prey Loc Predator Loc 2 44 47
{1: 22.173032818403126, 3: 22.111819999005405, 49: inf}
Agent Loc Prey Loc Predator Loc 3 45 48
{2: 22.5883880154283, 4: 22.28449558419896, 7: 22.37849247534385}
Agent Loc Prey Loc Predator Loc 4 44 49
{3: 21.664798513850577, 5: 20.35335147983422, 8: 20.267862710110414}
Agent Loc Prey Loc Predator Loc 8 48 2
{7: 21.753981443340155, 9: 20.429070574667936, 4: 21.742936065640592}
Agent Loc Prey Loc Predator Loc 9 44 3
{8: 19.44752839012264, 10: 17.784463905425373, 5: 19.43755573075377}
Agent Loc Prey Loc Predator Loc 10 48 2
{9: 20.429070574667936, 11: 19.280803114349826, 12: 19.280803114349826}
Agent Loc Prey Loc Predator Loc 12 48 3
{11: 18.40700804196396, 13: 17.8371388099411, 10: 19.129847493921595}
Agent Loc Prey Loc Predator Loc 13 47 7
{12: 17.968414394830084, 14: 16.612406443555585, 11: 17.968414394830084}
Agent Loc Prey Loc Predator Loc 14 46 8
{13: 17.09712759170138, 15: 15.666201543509404, 16: 16.310332755812304}
Agent Loc Prey Loc Predator Loc 15 45 4
{14: 16.143551768242762, 16: 16.141177267157797, 18: 15.073745675093303}
Agent Loc Prey Loc Predator Loc 18 46 5
{17: 15.677270252552251, 19: 14.623526862433156, 15: 15.680035401034406}
Agent Loc Prey Loc Predator Loc 19 50 9
{18: 15.35068758315165, 20: 15.350686691187445, 23: 14.207810131473257}
Agent Loc Prey Loc Predator Loc 23 1 10
{22: 15.050186609550481, 24: 13.826136465638898, 19: 15.050186609550481}
Agent Loc Prey Loc Predator Loc 24 1 12
{23: 13.77976993598538, 25: 13.77976993598538, 29: 12.433945653180496}
Agent Loc Prey Loc Predator Loc 29 1 13
{28: 11.708236588890129, 30: 11.004293234562374, 24: 12.534771807660293}
Agent Loc Prey Loc Predator Loc 30 2 14
{29: 11.134802116296031, 31: 11.150433254044348, 35: 9.31519868999088}
Agent Loc Prey Loc Predator Loc 35 49 15
{34: 6.805659666756189, 36: 6.805659666756189, 30: 8.833844366107792}
Agent Loc Prey Loc Predator Loc 36 48 18
{35: 6.646638511555303, 37: 5.565747909699786, 41: 4.5158089391174725}
Agent Loc Prey Loc Predator Loc 41 47 19
{40: 3.5890254499003498, 42: 3.5890254499003498, 36: 5.574439552105958}
Agent Loc Prey Loc Predator Loc 42 47 23
{41: 4.547860420249677, 43: 2.5571621471470127, 38: 4.547860420249677}
Agent Loc Prey Loc Predator Loc 43 48 22
{42: 3.372340872888713, 44: 1, 40: 3.372340872888713}
Agent Loc Prey Loc Predator Loc 44 48 21
{43: 2.3981295629839794, 45: 2.3981295629839794, 48: 0}
[24]
```

Shushang Nair, sgn40
Soumya Tyagi, st1108

Q. 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?





Shushang Nair, sgn40
Soumya Tyagi, st1108

Q. 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? How do you represent the states?

We ran the agent 1, agent 2 and U* agent for all the 125,000 states and recorded the next step that the agent would take in all the states and stored it in the file named `agent_move.txt`. One such state where the U* agent makes a different choice than Agent 1is in the screenshot attached below.

Agent 1 and Agent 2 prioritize a node that is away from the predator while the U* agent takes the step according to the utility and hence it makes a different choice.

3. Build a model to predict the value of U* (s) from the states S as input for your model

We built a model to predict the value of U* from states S as input for the model. This model is made by building a neural network from scratch.The output of this neural network is the utility of that state. Our neural network is described by:

- Layers: [2,4,3,1]
- Learning rate: 0.001
- Iterations: 5000

We built this model by following the steps below:

Data collection

Using the library pickle we stored the utility of all 3 states of the agent in the utility_dict_1.
We access this dictionary and go to all states and find the distance between the agent and predator and the distance between the agent and the prey. These are the only features that are relevant and hence feeded to the neural network. At every state we calculate the distances and save them in a pandas dataframe. The total number of dataframes will be 125,000. These are stored using the pickle library into utility.pkl file.

We access this file and for X we drop the utility column. We split the data into training and testing sets. X_train has the first 100,000 data points and X_test has the last 25,000.
Y is the utility dataset where Y_train contains the first 100,000 datapoints and Y_test has the last 25,000.

We then proceed to initialise the weights for the neural network, params saved as a dictionary.

The dimensions of the weights are as follows:

- W1=[2,4]
- W2=[4,3]
- W3=[3,1]
- b1=[4,1]
- b2=[3,1]
- b3=[1,1]

Forward Propagation

We used several activation functions like ReLu, leaky ReLu, sigmoid and tanh. We found that tanh function gave us the best result. The output after applying the activation function is given by A1. Then we proceeded to calculate Z2 which is given by: Z2 = (A1 . W2) + b2.
We then applied the activation function tanh on Z2 which gives us A2. With the value of A2 we then calculates z3, which is given by Z3 = (A2 . W3) + b3. Then no activation function is applied on Z3. Our yhat is initialized with the value of Z3. This data is then stored in the dictionary.

Loss function

We calculate the loss using the mean squared error loss formula which is as follows:
Loss = (1/n) * (y - yhat)^2

Back propagation

We have done back propagation using gradient descent. First, we find the derivative of the loss, given as dl = (z/n) * (y - yhat)

- As we had applied no activation function on Z3, hence dZ3 = dl;

  $dW3 = A2^T . dZ3$; $dB3 = \sum(dZ3)$

- $dZ2 = dZ3 . W3^T * dtanh(Z2)$; $dW2 = A1^T . dZ2$; $dB2 = \sum(dZ1)$ keeping the axis = 0

- $dZ1 = dZ2 . W2^T * dtanh(Z1)$; $dW1 = X\_train^T . dZ1$; $dB1 = \sum(dZ1)$ keeping the axis = 0

The next step is to update the weights by using the following formula:
W1 = W1 + learning rate * dW for all given weights accordingly. And over several iteration the loss will decrease and finally converge. Once loss converges, the weights are saved in the neural_weights file. This file consists of the trained neural network.

Shushang Nair, sgn40
Soumya Tyagi, st1108

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

We have taken 2 input features that we thought would be relevant that are distance between agent and prey and distance between agent and predator. So we represent our state using these 2 features.

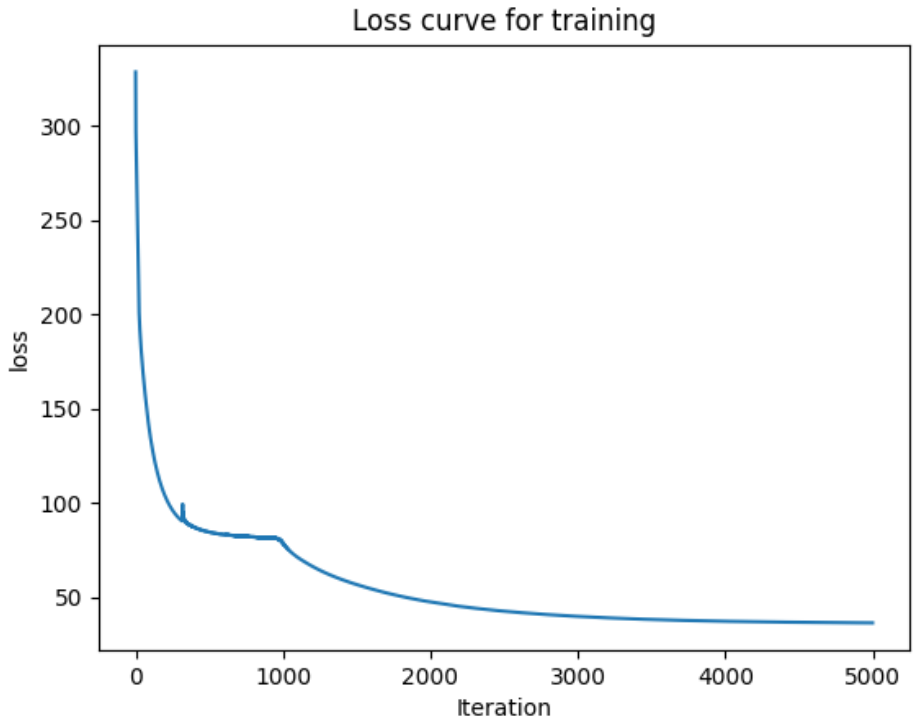
Q. What kind of model are you taking V to be? How do you train it?

We have chosen a neural network model on input 2 nodes that will contain distance between agent, prey and distance between agent, predator. We train it by repeatedly doing forward propagation and backward propagation until our losses finally converge. After convergence we stored the final weight's to `neural_weights_1`. We use these weights to run forward propagation so that we get the utility of that state while running our agent V.

Q. Is overfitting an issue here?

No overfitting is not an issue here because we essentially have covered all the states that are possible and hence there cannot be any new state that the model won't be able to predict.

Q. How accurate is V?

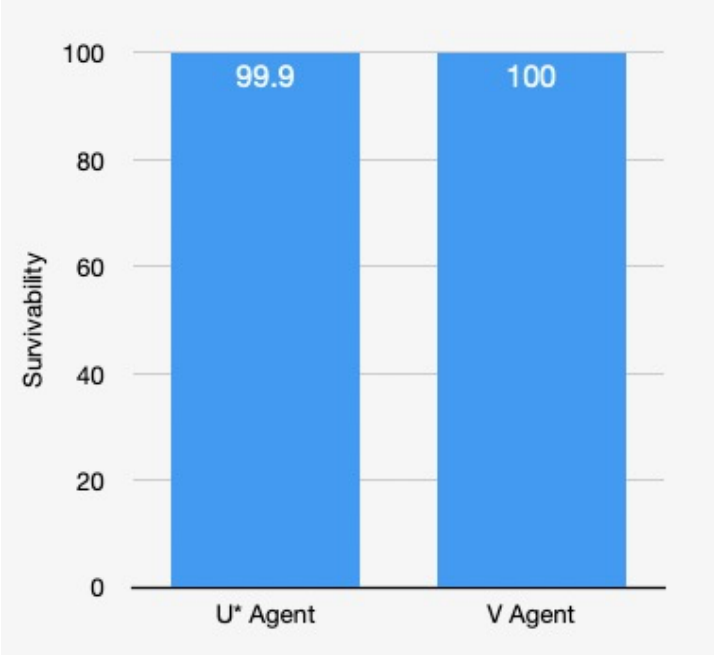


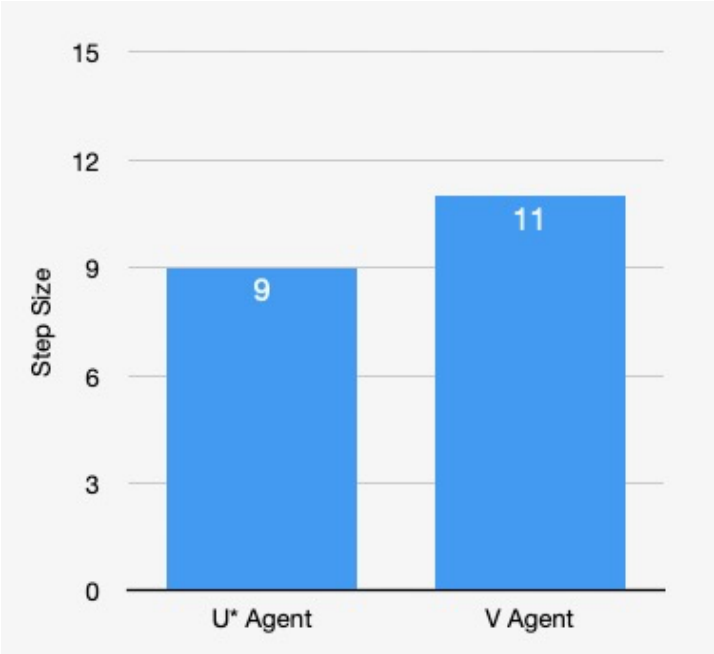
Our MSE losses converged around 40. That means we essentially have a 0.02 variation between each data point which is quite accurate.

Shushang Nair, sgn40
Soumya Tyagi, st1108

4. Once you have a model V, simulate an agent based on the values V instead of the values U.

Q. How does its performance stack against the U∗ agent?



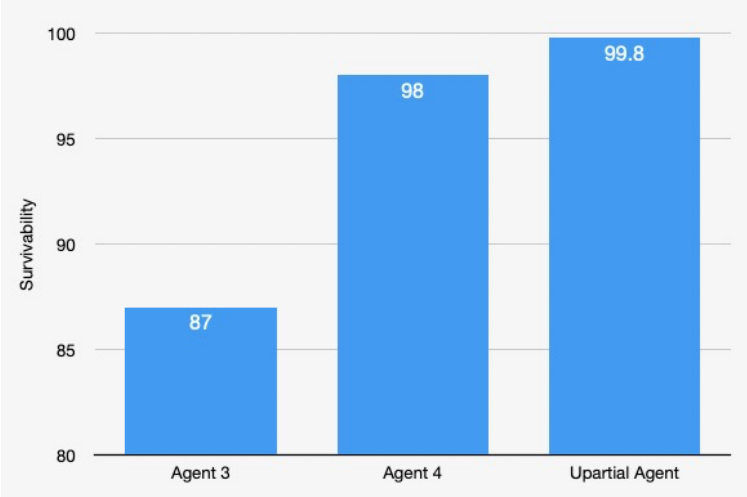


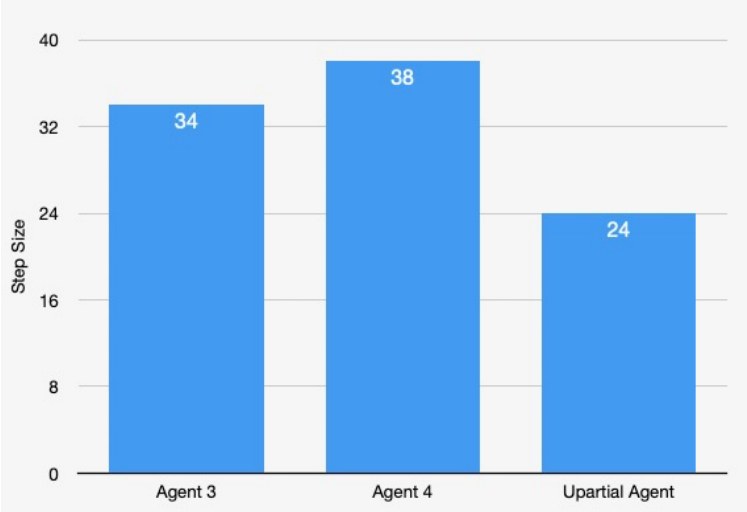
Shushang Nair, sgn40
Soumya Tyagi, st1108

5. Simulate an agent based on U partial, in the partial prey info environment case from Project 2, using the values of U* from above.

In this partial information setting, we have the information of where the agent and the predator are but we do not know where the prey is. For one move of the agent, there are 50 possible states of the prey. We shall calculate the utility for each of the 3 moves of the agent. This is done by multiplying the prey probability by the utility of the state for all 50 states. Since there are 50 states there will be 50 utilities. All these are summed up together and added to a list. This will be the utility for one of the three moves of the agent. In such a manner, the utility for the other two moves of the agent is also calculated and added to the list. Once we have updated our list, we will have three utilities of the three possible agent states, in the end, the minimum of the 3 in the list is considered to be the utility.

Q. How does this agent compare to Agents 3 and 4 from Project 2?





Shushang Nair, sgn40
Soumya Tyagi, st1108

Q. Do you think this partial information agent is optimal?

No, the partial information agent is not optimal because the agent doesn't catch the prey fast enough.

6. Build a model Vpartial to predict the value Upartial for these partial information states. Use as the training data states (including belief states) that actually occur during simulations of the Upartial agent.

While running Upartial we have stored the expected distance between agent and prey and distance between agent and predator along with its expected utility. So we have a large database of states and their expected utility (223140 states) that we can use to train our model. This has been stored in `Partial_Utility.pkl` file.

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

We have taken 2 input features that we thought would be relevant that are expected distance between agent and prey and distance between agent and predator. So we represent our state using these 2 features. Expected distance between agent and prey is given by:

$$\sum_{i=1}^{50} \text{P(Prey in i) * (Distance between agent and i)}$$

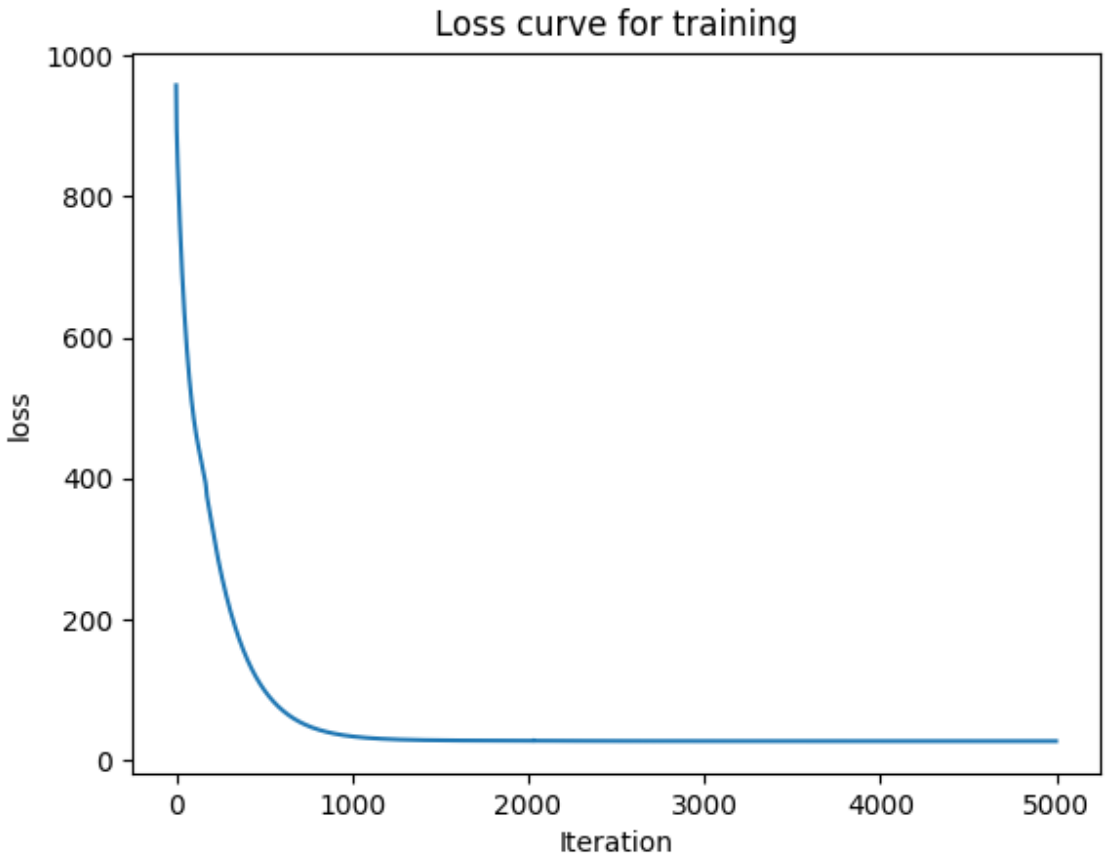
Q. What kind of model are you taking Vpartial to be? How do you train it?

We have chosen a neural network model on input 2 nodes that will contain expected distance between agent, prey and distance between agent, predator. We train it by repeatedly doing forward propagation and backward propagation until our losses finally converge. After convergence we stored the final weight's to `neural_weights_vpartial_1`. We use these weights to run forward propagation so that we get the utility of that state while running our agent Vpartial.

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

Yes, overfitting is an issue here because even though we have a larger dataset in comparison to V, there can be infinitely more possible states depending upon the prey belief vector. We can keep aside some part of the data and test our final weights on these data if it gives us the same losses as compared to train data we have prevented overfitting.

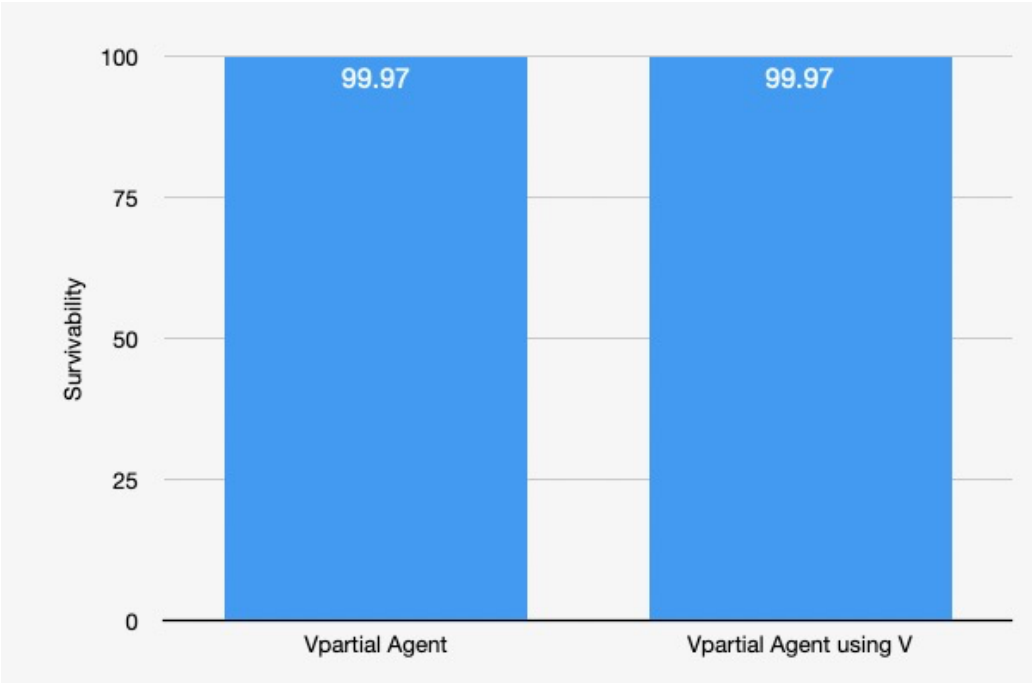
Q. How accurate is Vpartial? How can you judge this?



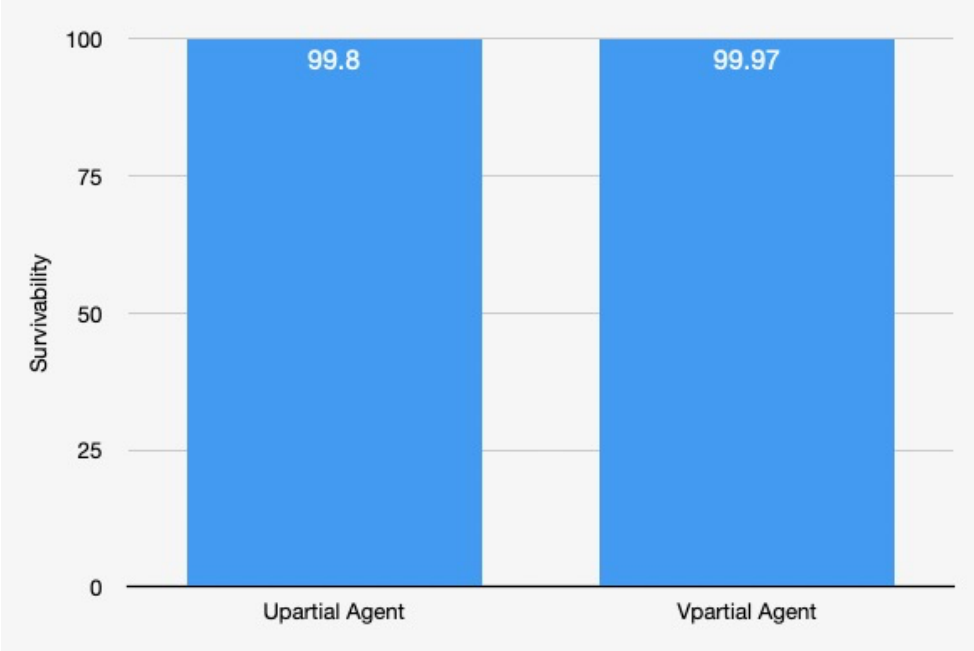
Our MSE losses converged around 35. That means we essentially have a 0.02 variation between each data point which is quite accurate.

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

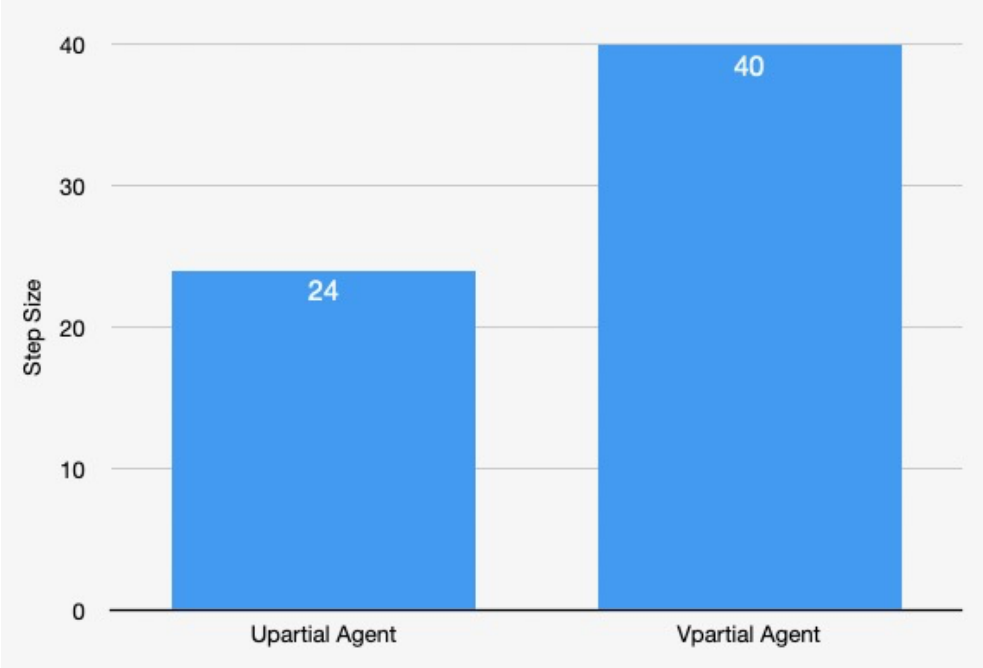
We got the same survivability rate in both the cases as shown below.



Q. Simulate an agent based on Vpartial. How does it stack against the Upartial agent?



Shushang Nair, sgn40
Soumya Tyagi, st1108



7. Reference

To learn and understand neural networks, we used the following resource as reference.

https://heartbeat.comet.ml/building-a-neural-network-from-scratch-using-python-part-1-6d399df8d432