CS362/633: Artificial Intelligence Pre-End Lab Report Team - OG

Anurag Jadhav *202051031* | Hardik Garbyal *202051079* | Pratik Mathpati *202051115* | Mayank Mangal Mourya *202051116*

I. Solved Assignments

Lab Assignment 1, Lab assignment 3, Lab Assignment 4, Lab Assignment 7 are solved which are in order -

- Lab Assignment 6
- Lab Assignment 8
- Lab Assignment 9
- Lab Assignment 10
- Lab Assignment 11

II. Lab Assignment 6

A. ***Read reference on MENACE by Michie and check for its implementations. Pick the one that you like the most and go through the code carefully. Highlight the parts that you feel are crucial. If possible, try to code the MENACE in any programming language of your liking***

.

We calculated -pass, -pass, di-gammas for re-estimating the state transition probability matrix (A), observation probability matrix (B), and initial state distribution () for Leo Tolstoy's work War and Peace using the reference "A Revealing Introduction to Hidden Markov Models" [b4]. We re-estimated A,B, and based on the observed sequence O, using initial values for A, B and and computed the , ,di-gammas and log likelihood for the data (War and Peace). We used 50000 letters from the book, deleting all punctuation and changing the letters to lower case, as shown in the section 8 [b4] example issue.We initialized each element of and A randomly to approximately 1/2

The following are the initial values:

$$\pi = \begin{bmatrix} 0.51316 & 0.48684 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.47468 & 0.52532 \\ 0.51656 & 0.48344 \end{bmatrix}$$

Each element of B was initialized to approximately 1/27. The precise values in the initial B are given in the Table After initial iteration,

$$\log(P(O/\lambda)) = 142533.41283009356$$

After 100 iterations,

$$\log(P(O/\lambda)) = 138404.4971796029$$

$$\pi = \begin{bmatrix} 0.00000 & 1.00000 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.28438805 & 0.71561195 \\ 0.81183208 & 0.18816792 \end{bmatrix}$$

This shows that the model = (A, B,) has improved significantly. After 100 iterations the model converged to, with final values of transpose of B in the Table,
By looking at the B matrix we can see that the hidden state contains vowels and consonants and space is counted as a vowel. The first column of initial and final values in Table are the vowels and the second column are the consonants.

B. ***Ten bent (biased) coins are placed in a box with unknown bias values. A coin is randomly picked from the box and tossed 100 times. A file containing results of five hundred such instances is presented in tabular form with 1 indicating head and indicating tail. Find out the unknown bias values. (2020 ten bent coins.csv) To help you, a sample code for two bent coin problem along with data is made available in the work folder: two bent coins.csv and embentcoinsol.m***

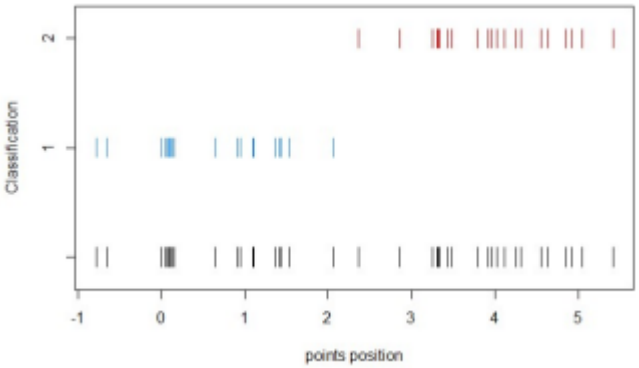
We used the Expectation Maximization algorithm from the previously shared reference material [b5] to solve the above problem. An expectation–maximization (EM) algorithm is an iterative approach for determining (local) maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models with unobserved latent variables[b5]. The coin types are the latent variables in our situation. We don't know what the coins are, but we do know the outcome of 500 trials. Using the present parameters, we used EM to predict the probabilities for each conceivable completion of the missing data.In our case, the latent variables are z, the coin types. We have no knowledge of the coins but the final outcome of 500 trials. Using the current parameter , we used EM to predict the probabilities for each possible completion of the missing data. We tried to implement EM for 10 bent coins to estimate their

	Initial		Final	
a	0.03735	0.03909	7.74626608e-02	6.20036245e-02
b	0.03408	0.03537	9.00050395e-10	2.34361673e-02
c	0.03455	0.03537	2.85482586e-08	5.54954482e-02
d	0.03828	0.03909	1.90968101e-10	6.91132175e-02
e	0.03782	0.03583	1.70719479e-01	2.11816156e-02
f	0.03922	0.03630	2.33305198e-12	2.99248707e-02
g	0.03688	0.04048	3.57358519e-07	3.08209307e-02
h	0.03408	0.03537	6.11276932e-02	4.10475218e-02
i	0.03875	0.03816	1.04406415e-01	1.70940624e-02
j	0.04062	0.03909	6.60956491e-26	1.92099741e-03
k	0.03735	0.03490	2.53743574e-04	1.21345926e-02
l	0.03968	0.03723	1.94001259e-02	4.17688047e-02
m	0.03548	0.03537	4.65877545e-12	3.85907034e-02
n	0.03735	0.03909	4.83856571e-02	6.14790535e-02
o	0.04062	0.03397	1.05740124e-01	4.23129392e-05
p	0.03595	0.03397	2.82866053e-02	1.84540755e-02
q	0.03641	0.03816	9.92576058e-19	1.32335377e-03
r	0.03408	0.03676	8.29107989e-06	1.07993337e-01
s	0.04062	0.04048	2.54927739e-03	9.75975025e-02
t	0.03548	0.03443	3.96236489e-05	1.50347802e-01
u	0.03922	0.03537	2.94555063e-02	8.54757059e-03
v	0.04062	0.03955	2.83949667e-19	3.05652032e-02
w	0.03455	0.03816	4.70315572e-25	3.37241767e-02
x	0.03595	0.03723	2.20419809e-03	1.38065996e-02
y	0.03408	0.03769	6.42635319e-04	3.04765034e-02
z	0.03408	0.03955	4.88081324e-27	1.10990961e-03
Space	0.03688	0.03397	3.49317577e-01	4.28089789e-08

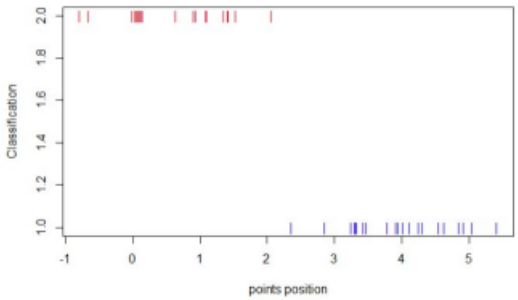
unknown bias,We took reference from Karl Rosaen's solution for 2 bent coins problem[b6].

C. A point set with real values is given in 2020 em clustering.csv. Considering that there are two clusters, use EM to group together points belonging to the same cluster. Try and argue that k-means is an EM algorithm

We used Expectation Maximization algorithm for grouping the points belonging to the same cluster. We also used k-means and compared the results of EM algorithm clustering and kmeans clustering [b5].



In Fig , the black lines represent points from the given dataset, while the blue and red lines represent the two clusters produced by the EM algorithm from the dataset, respectively. The points that belong to the same cluster have been grouped by the model.



Hard-clustering is used by k, which indicates that a point belongs to the cluster or does not belong to the cluster.The EM algorithm assigns points to clusters using a soft-clustering technique, and it gives the probability that a given point belongs to a cluster.. In this assignment, from the given dataset both the methods EM algorithm and k means yield the same result because the dataset was in such a way that the probability of a point belonging to a cluster was sufficient to assign the point to that cluster therefore the soft-clustering and hard-clustering yield the same result.

III. Lab Assignment 8

Basics of data structure needed for state-space search tasks and use of random numbers required for MDP and RL

The Markov decision process (MDP) is a mathematical tool used for decision-making problems where the outcomes are partially random and partially controllable.An MDP is defined by a set of states, a set of actions, a transition function, a reward function, and a discount factor. The transition function describes the probability of moving from one state to another when an action is taken. The reward function assigns a numerical value to each state and action, representing the desirability of being in that state or taking that action. The discount factor is a number between 0 and 1 that represents the relative importance of immediate rewards versus future

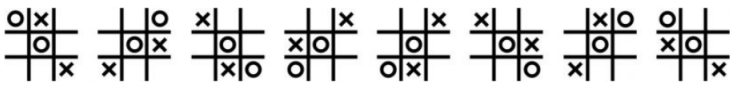
rewards.

A. Read the reference on MENACE by Michie and check for its implementations. Pick the one that you like the most and go through the code carefully. Highlight the parts that you feel are crucial. If possible, try to code the MENACE in any programming language of your liking.

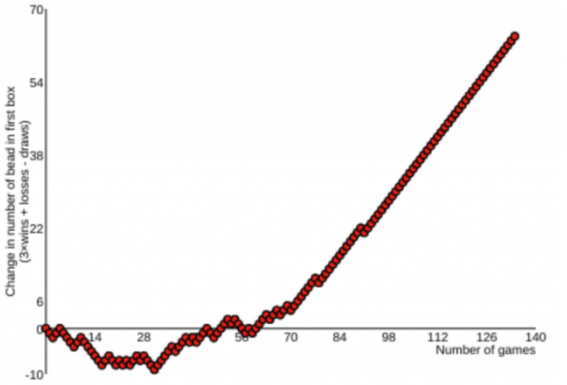
MENACE (Matchbox Educable Noughts And Crosses Engine) is a machine learning algorithm developed by Donald Michie in the early 1960s to play the game of tic-tac-toe (noughts and crosses) using matchboxes filled with beads. The algorithm works by learning from its mistakes and gradually improving its strategy over time. The MENACE algorithm consists of a collection of matchboxes, each containing a set of beads that represent the possible moves in a given state of the game. At the beginning of the game, the algorithm randomly selects a matchbox and removes a bead from it to determine its next move. After each move, the algorithm evaluates the outcome of the game and updates the bead counts in the matchboxes accordingly. The more successful a move is, the more beads are added to the corresponding matchbox, and vice versa.
Over time, the bead counts in the matchboxes converge to a distribution that represents the optimal strategy for playing tic-tac-toe. The algorithm can continue to learn and improve its strategy as it plays more games.

B. Working of MENACE

Each matchbox represents a specific board layout of tic-tac-toe and it does not consist of a box for each unique layout, though To make the model feasible, developer represented all layouts that were rotated versions of the same thing or that were symmetrical to each other with a single box. For example, one box would represent all the layouts below:



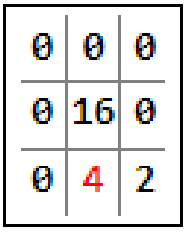
Initially the boxes contain colour-coded beads, where each colour represents a move (or position) on a board.
MENACE makes a move when the human player randomly picks a bead out of the box that represents the game's current state. The colour of the bead determines where MENACE will move.
The human player chooses the beads at random, just like a neural network's weights are random at the start. Also like weights, the beads are adjusted when there is failure or success. At the end of each game, if MENACE loses, each bead MENACE used is removed from each box. If MENACE wins, three beads the same as the colour used during each individual turn are added to their respective box. If if the game resulted in a draw, one bead is added.



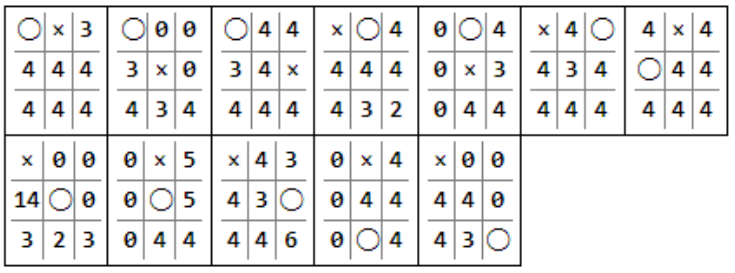
Results when MENACE plays a perfect-playing computer

C. Observations

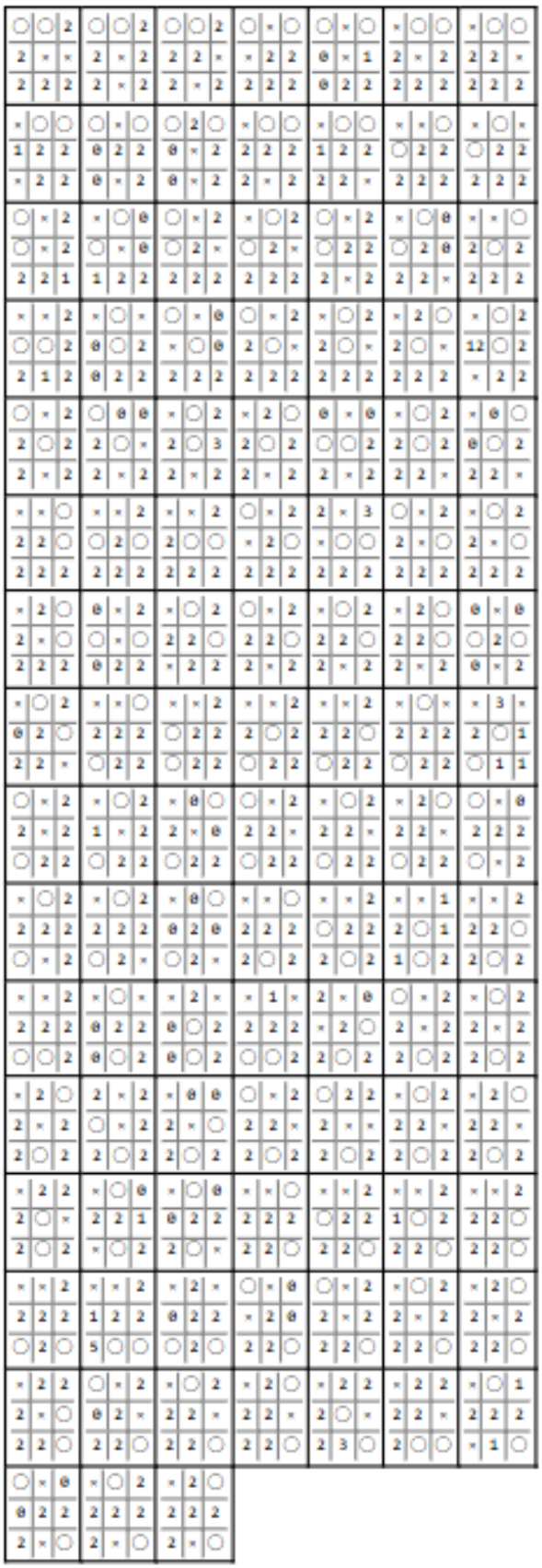
Over the course of many games, MENACE get rid of beads that represent bad moves and get more beads that represent good moves.



Box for first move



12 Box for third move



108 boxes for fifth move

D. Conclusion

MENACE has demonstrated the power of trial-and-error learning and how it can be applied to a variety of tasks beyond playing games. Its development has led to the creation of more advanced machine learning algorithms, such as deep reinforcement learning, which have been used to achieve groundbreaking results in fields such as computer vision and natural language processing.

IV. Lab Assignment 9

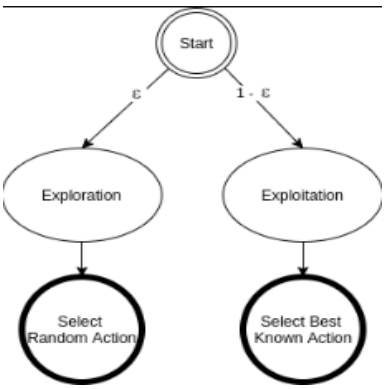
Understanding Exploitation - Exploration in simple n-arm bandit reinforcement learning task, epsilon-greedy algorithm

A. Exploration-Exploitation in Epsilon Greedy Algorithm

Exploitation is when the agent knows all his options and chooses the best option based on the previous success rates. Whereas exploration is the concept where the agent is unaware of his opportunities and tries to explore other options to better predict and earn rewards. As in the example above, going to the same pizza parlor and ordering the best pizza is an example of exploitation; here, the user knows all his options and selects the best option. On the other hand, going to the new Chinese restaurant and trying new things would come under exploration. The user is exploring new things; it could be better or worse, the user is unaware of the result.

B. Epsilon Greedy Action Selection

The epsilon greedy algorithm chooses between exploration and exploitation by estimating the highest rewards. It determines the optimal action. It takes advantage of previous knowledge to choose exploitation, looks for new options, and select exploration.



C. Epsilon greedy policy improvement

The policy improvement is a theorem that states For any epsilon greedy policy π, the epsilon greedy policy π concerning qπ is an improvement. Therefore, the reward for π will be more. The inequality is because the max function

returns a more excellent value than the arbitrary weighted sum.

Action at time(t) { max $Q_t(a)$ with probability 1-ϵ ; any action (a) with probability ϵ

D. Pseudocode of Epsilon Greedy

Algorithm 2: Epsilon-Greedy Action Selection

Data: Q: Q-table generated so far, : a small number, S: current state
Result: Selected action
Function *SELECT-ACTION(Q, S, ϵ)* **is**
  n ← uniform random number between 0 and 1;
  if $n < \epsilon$ **then**
    A ← random action from the action space;
  else
    A ← maxQ(S,.);
  end
  return selected action A;
end

E. Multi-Armed Bandit Problem

The multi-armed bandit problem is used in reinforcement learning to formalize the notion of decision-making under uncertainty. In a multi-armed bandit problem, an agent(learner) chooses between k different actions and receives a reward based on the chosen action. The multi-armed bandits are also used to describe fundamental concepts in reinforcement learning, such as rewards, timesteps, and values. For selecting an action by an agent, we assume that each action has a separate distribution of rewards and there is at least one action that generates maximum numerical reward. Thus, the probability distribution of the rewards corresponding to each action is different and is unknown to the agent(decision-maker). Hence, the goal of the agent is to identify which action to choose to get the maximum reward after a given set of trials.

F. Solution of the Multi-Armed Bandit Problem

- To solve this problem, instead of dealing with distributions caused by certain actions, we need to deal with one number that will somehow quantify the distribution. Every distribution has an expected or a mean value. Consequently, expected or mean values are a good choice for tackling this problem. Let us first introduce a few definitions.
- The number of arms is denoted by N.
- The action is denoted by a_i, where i=1,2,…, N. The action a_i is actually the selection of the arms i.
- The action selected at the time step k is denoted by A_k. Note that A_k is a random variable.
- The action A_k can take different numerical values a_1, a_2,..,…,a_N. A numerical value of the action i is denoted by a_i.
- Every action A_k at the time step k, produces a reward that we denote by R_k. Note that R_k is a random variable. The expected or the mean reward of the action we take is referred to as the value of that action. Mathematically speaking, the value is defined by

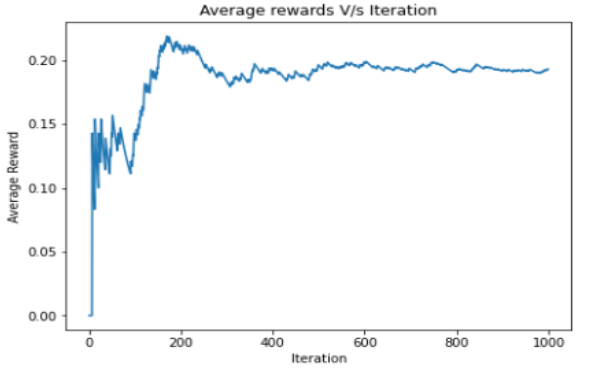
$$v_i = v(a_i) = E[R_k|A_k = a_i]$$

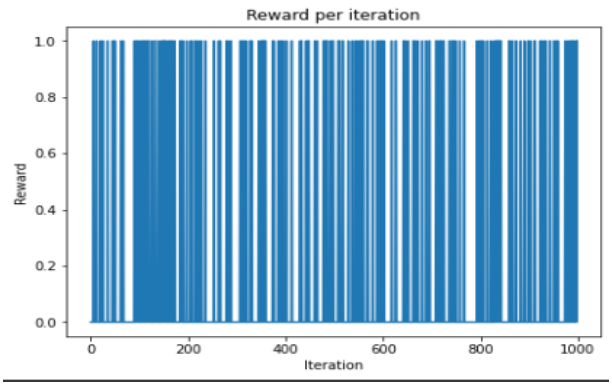
Now, if we would know in advance the values of actions v_i, i=1,2,3,…, N, our task would be completed: we would simply select the action that produces the largest value! For that purpose, we will use an action value method. The action values method aims at estimating the values of actions and using these estimates to take proper actions that will maximize the sum or rewards.
The story goes like this. At every time step k we can select any of the actions a_i. The action that is selected at the time step is denoted by A_k. The reward produced by this action is R_k. So, as the discrete-time k progresses, we will obtain a sequence of actions A_1, A_2,…, and a sequence of produced rewards R_1, R_2,. From this data, we want to estimate a value of every action, that is denoted by $v_i = v(a_i)$. We can estimate these values at the time step k, as follows

$$v_{i,k} = v_i a_i = \sum_{j=1}^{k-1} R_{i,j}$$

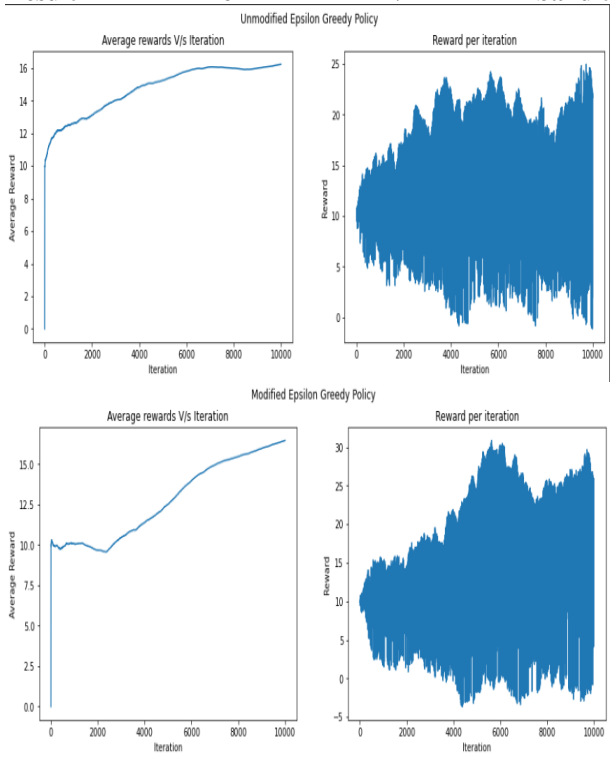
G. Results

1) **Result for o/1 bandit**





2) **Result for N bandit**



H. Conclusion

The n-arm bandit problem is a simple but important problem in reinforcement learning. The epsilon greedy algorithm is an effective method for solving this problem and balancing exploration and exploitation. This implementation provides a basic framework for solving the n-arm bandit problem using the epsilon greedy algorithm.

V. Lab Assignment 10

A. Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework used to model decision-making processes in which an agent interacts with an environment over a series of discrete time steps. It is named after the mathematician Andrei Markov, who developed the theory of Markov chains. An MDP consists of a set of states, a set of actions, a transition function, a reward function. At each time step, the agent observes the current state of the environment, selects an action from the set of available actions, and receives a reward from the environment based on the action taken and the resulting state. The transition function defines the probability distribution over the next state given the current state and action. The reward function specifies the reward the agent receives for each state-action pair. The goal of the agent is to learn a policy that maximizes the expected cumulative reward over time. A policy is a mapping from states to actions, and the value of a state is the expected cumulative reward starting from that state and following the policy. The optimal policy is the one that maximizes the expected cumulative reward. MDPs are used in a variety of fields, including robotics, economics, game theory, and artificial intelligence, to model decision-making problems under uncertainty.

B. Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or -1. We assume that the environment is fully observable so that the agent always knows where it is. You may decide to take the following four actions in every state: Up, Down, Left and Right.

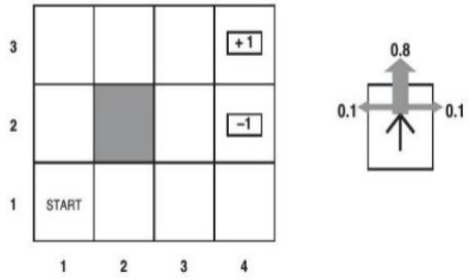


Figure 1: 4 × 3 grid world with uncertain state change.

However, the environment is stochastic, which means the action that you take may not lead you to the desired state. Each action achieves the intended effect with a probability of 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction with equal probabilities (0.1). Furthermore, if the agent bumps into a wall, it stays in the same square. The immediate reward for moving to any state (s) except for the terminal states S+ is r(s)= -0.04. And the reward for moving to terminal states is +1 and -1 respectively. Find the value function corresponding to the optimal policy using value iteration. Find the value functions corresponding optimal policy for r(s)= -2, 0.1, 0.02, 1

C. BELLMAN EQUATION

The utility/value of a state is the expected reward for the next transition plus the discounted utility/value of the next state, assuming that the agent chooses the optimal action.

- U(s): Utility/Value of state s
- P(s '— s, a): Probability that action a in state s reaches state s '
- R(s, a, s '): Reward received when agent transitions from state s to s ' via action a
- Gamma: Discount Value

D. POLICY ITERATION

It is a technique used in the field of reinforcement learning (RL) for finding an optimal policy for an agent in an environment. The policy iteration algorithm consists of two steps: policy evaluation and policy improvement. In the policy evaluation step, the algorithm evaluates the current policy by computing the expected value of the total reward over a given number of steps, starting from the current state. This is done using the Bellman equation, which expresses the expected reward as a sum of the immediate reward plus the

discounted value of the expected future reward. In the policy improvement step, the algorithm updates the policy based on the value function computed in the previous step. The new policy is obtained by selecting the action that maximizes the expected value of the total reward from the current state.

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $a \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

E. VALUE ITERATION

It is a technique used in the field of reinforcement learning (RL) for finding an optimal policy for an agent in an environment. The value iteration algorithm is a dynamic programming algorithm that computes the optimal value function of a Markov decision process (MDP) by iteratively updating the value estimates of each state based on the Bellman equation. The Bellman equation expresses the value of a state as the maximum expected sum of rewards that can be obtained from that state and following the optimal policy thereafter.

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
      $v \leftarrow V(s)$
      $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
   $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

F. Formulate the continuing finite MDP

A Markov decision process (MDP) is a mathematical framework used in reinforcement learning to model decision-making problems in which the outcomes are partially random and partially under the control of an agent. A continuing finite MDP is a type of MDP in which the agent interacts with the environment for an infinite number of time steps, but the state and action spaces are finite. Formally, a continuing finite MDP is defined by a tuple (S, A, P, R,), where:

S is a finite set of states

A is a finite set of actions

P is a transition function that defines the probability of transitioning from state s to state s' when taking action a, i.e., P(s — s, a)

R is a reward function that defines the immediate reward received when transitioning from state s to state s' by taking action a, i.e., R(s, a, s')

is a discount factor that determines the importance of future rewards relative to immediate rewards.

G. Continuing Finite MDP is Formulated As :-

- State space (S): The MDP has a finite set of states S = s1, s2, ..., sn.
- Action space (A): The MDP has a finite set of actions A = a1, a2, ..., am.
- Transition probabilities (P): The transition probability P(s', r — s, a) is the probability of transitioning to state s' and receiving reward r when taking action a in state s. This can be represented as a matrix of size n x m x n x R, where n is the number of states, m is the number of actions, and R is the range of possible rewards.
- Reward function (R): The reward function R(s, a, s') is the immediate reward received when transitioning from state s to state s' by taking action a. This can be represented as a matrix of size n x m x n, where n is the number of states and m is the number of actions.
- Discount factor (): The discount factor is a value between 0 and 1 that determines the importance of future rewards relative to immediate rewards.

The agent interacts with the environment over an infinite number of time steps, starting from an initial state s0. At each time step t, the agent observes the current state st, selects an action at from the set of available actions A(st), and receives a reward R(st, at, st+1) for transitioning to the next state st+1 according to the transition probability P(st+1, r — st, at). The goal of the agent is to learn a policy (s) that maps each state to an action that maximizes the expected sum of discounted rewards over time. The expected sum of discounted rewards starting from state s and following policy is given by: V(s) = [t=0 t R(st, at, st+1) — s0 = s,] The optimal policy * is the policy that maximizes the expected sum of discounted rewards over all states: * = argmax V(s) Value iteration and policy iteration are two common algorithms used to find the optimal policy for a continuing finite MDP.

H. ***Write a program for policy iteration and resolve the Gbike bicycle rental problem with the following changes. One of your employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one bike to the second location for free. Each additional bike still costs INR 2, as do all bikes moved in the other direction. In addition, you have limited parking space at each location. If more than 10 bikes are kept overnight at a location (after any moving of cars), then an additional cost of INR 4 must be incurred to use a second parking lot (independent of how many cars are kept there).***

I. ***Results***

- **Result for Bellman Equation**

```
{(0, 1): 9.998955043236686,
 (1, 2): 9.998955043236686,
 (2, 1): 9.998955043236686,
 (0, 0): 9.998955043236686,
 (3, 1): -1.0,
 (2, 0): 9.998955043236686,
 (3, 0): 9.998955043236686,
 (0, 2): 9.998955043236686,
 (2, 2): 9.998955043236686,
 (1, 0): 9.998955043236686,
 (3, 2): 1.0}
```

- **Result for 2nd Problem**

- **Result for 3rd Problem**

VI. LAB ASSIGNMENT 11

To understand the design of type-1 (Mamdani) Fuzzy expert system

In this laboratory, we will try to identify the appropriate time needed to wash the load of clothes, given the dirtiness and the volume of the load. This is a simplified design, in practice a lot more variables are involved including: water level, amount of detergent to be dispensed, and temperature of water (recent development) and variables that you and I may imagine in future.

A simple rule-base for the washing machine time problem is given below for reference.

A. The rule table consists of 12 rules.

If load volume is medium load
And load dirtiness is lightly dirty
Then washing time is medium time

Answer: ;We examine a simple two-input one-output problem that includes total 12 rules using Mamdami-style fuzzy inference process

input : x and y (Load Dirtiness and Load Volume) with linguistic values A1,A2,A3,A4 (very dirty,medium dirty,low dirty,not dirty) and B1,B2,B3 (full load,medium load,low load) respectively

Output: z (time needed) with linguistic values C1,C2,C3, and C4 (very long time,long time,medium time and little time). Let's define membership functions for all input linguistic value (i.e., for A1) membership function for Load Dirtiness

$$A1(x) = (x60)/40, 60 \leq x \leq 100$$

$$A2(x) = (65x)/15, 50 \leq x \leq 65$$

$$A3(x) = (35x)/15, 20 \leq x \leq 35$$

$$A4(x) = (5x)/5, 0 \leq x \leq 5$$

membership function for Load Volume

$$B1(x) = (x50)/50, 50 \leq x \leq 100$$

$$B2(x) = (100x)/50, 50 \leq x \leq 100$$

$$B3(x) = (50x)/50, 0 \leq x \leq 50$$

Step 1: Fuzzification

1) Take the crisp inputs, x1 and y1 (Load Dirtiness and Load Volume)

2) Determine the degree to which these inputs belong to each of the appropriate fuzzy sets.

Step 2: Rule Evaluation

Take the fuzzified inputs, and apply them to the antecedents of the fuzzy rules. If a given fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation. This number (the truth value) is then applied to the consequent membership function. For example, the entry in the second row and the third column of the table specifies the rule:

If load volume (y) is medium load (B1)

AND load dirtiness (x) is low dirty (A3) Then washing time is medium time and for x1 = 91, y1 = 79

$$A_3 \cap B_1(x) \quad = \quad min[A_3, B_1] \quad = \quad min(0, 0.58) = 0(7)$$

OR

$$A_3 \cap B_1(x) \quad = \quad prod[A_3, B_1] \quad = \quad A_3 x B_1 \quad = \quad 0x0.58 = 0(8)$$

The result of the antecedent evaluation can be applied to the membership function of the consequent. In other words, the consequent membership function is clipped or scaled to the level of the truth value of the rule antecedent

Step 3 Aggregation of the rule outputs

the membership functions of all rule consequent previously clipped or scaled and combine them into a single fuzzy set.

Step 4 Defuzzification

Centroid Technique: Here we find the centroid of the found model using the below formula:

$$COG \quad = \quad \frac{\int_a^b u.A(x)x\,dx}{\int_a^b u.A(x)\,dx}$$