Advantage Actor-Critic (A2C):- (designed for discreet action spaces)

-) on policy RL algorithm that combines benfits of policy gradient & value function

=> maintains two components : i) actor network that learns policy by selecting actions

ii) critic network that estimates value function.

Pros:- allows parallelization, synchronous updates, learn fast than other policy based methods using value function to guide the policy updates.

Cons:- limited sample efficiency, slow training process because multiple interactions.

Deep deterministic Policy gradient (DDPG):- (designed for continuous action spaces)

-> off policy RL algorithm that combines the benefits of Deep-Q-Network (DQN) and deterministic policy gradient.

-> DDPG also uses actor-critic framework : i) actor network learns deterministic Policy.

ii) critic network estimates value function.

Pros:- Can handel continuous action spaces, uses experience replay buffer to store Past experiences and use them for updation of networks to ↑ accuracy & efficiency, Can learn high-dimensional state using NN.

Cons:- sensitive to hyperparameter tuning, tough to find right balance b/w explor, exploi... DDPG soffers with tasks that have sparse reward (or) log-term credit assignments.

-> Policy evaluation is process of determining the quality (or) values of a given policy.

-> involves estimating expected return (or) values for each state in policy.

-> After evaluation, it can be improved by making changes to policy by using estimated values

This is done iteratively.

Policy evaluation methods:- i) Cost-benefit analysis (Compares the cost and benefits of a Policy to determine wheather it is worthy implementing)

ii) Impact analysis : assesses the short term, long term effects of a policy.

iii) Process evaluation :- checks how a policy is implemented and improve it by updation

vi) once the policy is evaluated, the results can be used to improve the policy by making changes to polices design, implementation & funding.

Pros :- Improve decision making ; Increase accountability ; Increase public trust

Cons :- Cost (expensive) ; Time consuming ; Complexity.

Algorithm for Policy iteration:-

i) initializing the value function to zero.

ii) algorithm is iterated until the convergence (or) max no. of iterations reached.

iii) In each iteration, the algorithm updates the value function for each state.

iv) algorithm checks for convergance after each iteration, if value fun converged then the algorithm terminate, if not, moves to next iteration.

v) alg returns the value function after last iteration.

eq :- $V'(s) = \Sigma\left[P(s'|s,a) * \left(R(s,a,s') + \gamma * V(s')\right)\right]$

Dynamic programming (DP):-

- Technique for solving complex optimization problems efficently
- Breaks down problems into smaller subproblems • uses recursion and memorization by storing the solutions of subproblem in table for) catch allows for effivent retrival and reuse of ampuled results
- Efficiently store large instances of the problem. • used in CS, operations research, AI, scheduling, route planning.

Principle of Optimality:

- fundamental concept in dp in which an optimal solution has optimal sub-solutions.
- Initial decision leads to optimal subsequent decisions • Simplifies complex problems by breaking them down, • avoid redundant computaion by memoriting. similar to dp by reusing Solutions. (bellman policy eq) MDP equation : $V\pi(s) = \Sigma\left[\pi(a|s) * \Sigma\left[P(s'|s,a) * (R(s,a,s') + \gamma * V\pi(s'))\right]\right]$.

Bellman expected eq for state-value function : $v(s) = \Sigma\left[\pi(a|s) * \Sigma\left[P(s'|s,a) * (R(s,a,s') + \gamma * v(s'))\right]\right]$.

Bellman expected eq for action-value function: $Q\pi(s,a) = \Sigma\left[P(s'|s,a) * (R(s,a,s') + \gamma * \Sigma\left[\pi(a'|s') * Q(s',a')\right]\right]$

Bellman optimality equation : $V^*(s) = \max\left[\Sigma\left[P(s'|s,a) * (R(s,a,s') + \gamma * V^*(s')\right]\right]$.

$V(s) \rightarrow$ value of state s ; $V^*(s) \rightarrow$ value function for state s under an optimal policy.

$\pi(a|s) \rightarrow$ Probability of taking action a in state s under current policy.

$P(s'|s,a) \rightarrow$ Probability of transitioning to s' from state s when taking action a.

$R(s,a,s') \rightarrow$ immediate reward obtained by transitionsing from state s to state s' with action

$\gamma \rightarrow$ is discount factor.

$V(s') \rightarrow$ value function of next state s'.

*) Bellmen optimality equation express the relationship blw the value of a state under an optimal policy and the expected return for the best action from that state.

Expected Discounted Return :- $E[\Sigma(\gamma^t * R\_t)]$

-) In RL, the expected discounted return is the sum of all future rewards, discounted by a factor γ. The discount factor is used to account for a fact that the rewards received in the future are less valuable than rewards received immediately.

-) If agent-env interaction breaks naturally into subsequences, which we call episodes, then the expected discount return for an episode is sum of discounted rewards for all steps in the episode.

Pole Balancing as an application of MDP :-

- classic RL problem ⟶ agent learns to balance a pole on cart ⟶ agent can move left, right ⟶ reward is given for keeping pole upright ⟶ should balance for long period of time
- Pole balancing can be modeled as an MDP ⟶ state space is set of all positions and orientation of pole & cart ⟶ action space is set of all possible movements of the cart ⟶ Reward for keeping the pole upright ⟶ Goal: maximize expected discount return.

RL algorithms for Pole balancing :

→ Q learning (value based RL algorithm learns Q-function (state-action values)). simple to implement
-> Policy gradients (learns policy (state to action mapping), more complex than Q-lrn but accuret.
-) Advantage actor critic (Combines strengths of both value based, Policy based a/g) impro performance.

Policies and value functions in MDP :-

-) In markov decision process (MDP), Policies and value functions play a crucial role in determining how an agent behaves and evaluates states and actions.

Policies : determine agents behavior in an MDP.

- Policy is a mapping that specifies actions for states.
- It can be deterministic (or) stochastic.

Value functions : measures the value of being in a particular state or taking an action, value funcs quatify the desirability of state.

- state value function (V) measures the goodness of being in a state.
- Action-value function (Q) measures the value, of taking an action in a state.

Bellman equation for policy :-

$$V_\pi(s) = \Sigma[\pi(a|s) * \Sigma[P(s'|s,a) * (R(s,a,s') + \gamma * V_\pi(s'))]] \text{ for all actions a}$$

a) Reinforcement learning methods and Elements in Tic-Tac-Toe

Ans) RL is a type of feedback based semi-supervised machine learning Approach in which an agent learns to make decision by interacting with an environment. RL methods and elements play a significant role in training an agent to play Tic-Tac-Toe effectively. The methods & rules as following:

i) Agent and Environment: In Tic-Tac-Toe, the RL agent is typically one of the players, while the environment encompasses the game board, rules, and opponent player. The agent interacts with the env by selecting moves and receiving rewards based on the outcome.

ii) State Repesentation: To apply RL, the games status must be represented in a way that the agent can understand and learn from. one common representation to capture the current state of the game is 3x3 grid, where each cell can be empty (or) occupied by agent or opponent's marker.

iii) Actions and policy: Define the action space for the RL agent, like the agent can place its markes in any of the empty cell on game board and design a policy or a reward function to provide feedback to its agent based on its actions. Assign +ve for wins, -ve for looses & neutral for no win or loose. The rewards should guide the agent towards learning optimal strategies.

iv) RL algorithm:- selecting a suitable RL algorithm, such as Q-learning or Deep-Q-Network (DQN), to train agent. These algorithms updates Q-values (expected comulative rewards) based on the state-action-reward seq experienced during training

v) Exploration-Exploitation strategy:- Initally the agent follows exploration strategy (epsilon-greedy) to explor and learn different actions & game dynamics. As training progresses, the agent can gradually shift towards the exploitation to make optimal decisions based on its learned knowledge.

vi) Training process:- Training the RL agent by having it play against opponents and updating its Q-values based on the observed rewards.

- The agent learns to associate states with the best actions to maximize its reward over time

vii) Convergence :- Convergence occurs when further training does not significantly improve the performance of the agents policy. This convergence criteria based on the agent's win rate or the stability of its Q-values.

viii) Evaluation :- Assess the performance of the trained agent by having it play against different opponents or a predefined benchmark. This evaluation helps gauge the effectiveness of the RL approach in playing Tic-Tac-Toe.

By applying RL to Tic-Tac-Toe, the agent learns to make sequential decisions based on the game state and optimize its actions to maximize reward. The training process allows the agent to discover optimal strategies and find all possible solutions that lead to winning outcomes in the game.

Q) Difference b/w trail & error and delayed reward methods in RL.

Aspect	Trail and Error	Delayed Reward
Objective	The trail & error method focuses on finding immediate rewards without considering long-term consequences	The delayed reward method aims to max long term rewards by considering cumulative or future consequences of actions
Approach	This method involves exploring the env by trying diff actions and observing the immediate reward obtained.	This method takes into account the delayed rewards associated with actions in different states.
Decision making	Action are chosen based on immediate feedback and are not based on long-term goals or expected cumulative reward	Actions are chosen based on their potential to optimize the overall expected return.
learning Efficiency	This can be inefficient as it may requries a large no/of attempts to discover an optimal or near-optimal policy	This is more efficient in learning optimal Policies by considering the long-term consequences of actions.
Exploration	High exploration, low exploitation	Balancing exploration and exploitation.
Planning	Short-sighted decisions	long-term planning and foresight
optimality	Suboptimal in the absence of exploration	Can converge to optimal policies
Complexity	Simple and straightforward	More complex decision-making

a)

To calculate the expected value of a discrete random variable and a continuous random variable, we use different methods. Here's how you can calculate the expected value for each and then prove the given equation:

i) Expected value of a Discrete Random Variable (x):

For a discrete random variable X that takes on values $x_1, x_2, \ldots, x_n$ with corresponding probabilities $P(x=x_1), P(x=x_2), \ldots, P(x=x_n)$, the expected value ($E(x)$) is calculated as:

$$\therefore E(x) = x_1 * P(x=x_1) + x_2 * P(x=x_2) + \ldots + x_n * P(x=x_n)$$

This formula calculates the weighted average of the possible values of x, where the weights are the probabilities associated with each value.

ii) Expected value of a Continuous Random Variable(x):

For a continuous random variable X with probability density function (PDF) $f(x)$ over a certain range, the expected value ($E(x)$) is calculated as the integral of x times the PDF over the range:

$$E(x) = \int [x * f(x)]\, dx$$

This integral sums up the product of each value of x and its corresponding Probability density.

Now, let's proove the given equation:

We are given : $E(axA^2 + bx)$

expanding the equation, we have: $E(axA^2 + bx) = E(axA^2) + E(bx)$

Using linearity of the expected value, we can split the eq into two separate terms:

$E(axA^2) + E(bx) = aE(A^2) + bE(x)$

This eq holds because $E(A^2)$ represents the expected value of the constant A^2, and $E(x)$ represents the expected value of the random variable X.

$\therefore$ we have shown that $E(axA^2 + bx) = aE(xA^2) + bE(x)$ using the linearity Property of the expected value.

Advantage Actor-Critic (A2C):

This is an on-policy RL algorithm that combines elements of both actor-critic methods. It has the following key features:

i) **Actor-Critic Architecture:** A2C employs an actor-critic architecture, where the actor network selects actions based on the current policy, and the critic network estimates the value function to evaluate the quality of the chosen actions.

ii) **Advantage Estimation:** A2C uses advantage estimation to update the actor and critic networks. The advantage represents the diff b/w the estimated value of a state-action pair and the expected value of the state.

iii) **Synchronous Updates:** A2C updates the actor and critic network simultaneously using the same set of collected experiences. This approach simplifies the learning process and allows for more stable updates.

iv) **Policy Gradient Optimization:** A2C optimizes the actor network using policy gradients, typically with the advantage function as a baseline. This enable the actor to improve its policy based on the feedback received from the critic.

Deep Deterministic Policy Gradient (DDPG):-

is an off-policy RL algorithm designed specifically for continuous action spaces. It has following features:-

i) **Actor-Critic Architecture:** DDPG also uses an actor-critic architecture, where the actor network directly outputs actions and the critic network estimates the value function.

ii) **Experience Replay:** DDPG incorporates an experience replay buffer to store and sample past experiences. This buffer breaks the sequential nature of data, reducing correlation b/w the consecutive samples and enabling more efficient learning.

iii) **Deterministic policy:** By learning this, DDPG maps states to specific actions. This is particularly useful in continuous action spaces where stochastic policies may not be suitable.

iv) Target Network: These were utilized for both actor and critic. These target network are slowly updated using a soft update mechanism, stabilizing the learning process and reducing the effects of target value estimation errors.

v) Exploration - Exploitation: DDPG uses noise injection, such as Ornstein-Uhlenbeck noise, to explore the action space during training. This noise helps balance and encourage the agent to explore different actions.

a) Policy evaluation is the process of determining the quality or value of a given policy in RL. It involves estimating the expected return or value of each state or state-action pair under the policy. Once a policy is evaluated, it can be improved by making changes to the policy, based on the estimated values. The process of policy evaluation and improvement is typically performed iteratively.

Algorithm for Iterative Policy Evaluation:-

* Input:
- Policy π
- Environment dynamics model (transition probabilities and rewards)
- Discount factor γ
- Maximum number of iterations N.

* Output:
- Estimated value function V

i) Initialize V with arbitrary values for all states

ii) Repeat the following steps until convergence or reaching the maximum number of iterations:

a) Initialize a new value function V' with all values set to 0.

b) For each state s:

Compute the state value using the Bellman expectation equation:

$$V'[s] = \sum [R + \gamma * V[s']] * P(s'|s, \pi(s))$$, where s' is the next state, R is the reward; P is the transition probability, and $\pi(s)$ is the policy at state S.

c) Update the value function V with the new value: $V = V'$;

d) Check for convergence by calculating the maximum change in the value function.

$\Delta = \max(|V'[s] - V[s]|)$ for all states S.

e) If $\Delta < \varepsilon$ (a small threshold), where "ε" is the desired convergence threshold, terminate the loop.

iii) Return the estimated value function V.

Note: The iterative policy Evaluation algorithm updates the value function iteratively based on the expected rewards and next state values. It continues this process until the values converge or the max no. of iterations is reached.

By evaluating the policy and estimating the value function, it becomes possible to improve the policy using various methods such as policy iteration or value iteration.

a) Dynamic Programming (DP):-

is a powerful method used in RL to solve complex problems by breaking them down into smaller subproblems. DP widely used in RL for optimization and decision-making tasks. It leverages the principle of optimality to efficiently find optimal solutions.

Principle of optimality:-

The principle of optimality states that an optimal policy has the property that, regardless of the inital state and decision, the remaining decisions must be optimal with respect to the subproblem starting from the resulting state. It allows DP algorithm to solve problems by decomposing them into smaller subproblems and guaranteeing overall optimality.

Bellman Expectation Equations:-

- State-Value Function (v): $V(s) = \Sigma\left[P(s'|s,a) * (R + \gamma * V(s'))\right]$

This equation estimates the value of a state by summing the expected immediate reward (R) and the discount value of next state ($V(s')$) across all possible actions (a) and next state(s').

- Action-Value Function (Q): $Q(s,a) = \Sigma\left[P(s'|s,a) * (R + \gamma * \Sigma\left[\pi(a'|s') * Q(s',a')\right])\right]$

This equation estimates the value of a state-action pair by summing the expected immediate reward (R) and the discounted value of the next state-action pair ($Q(s',a')$) across all possible next states (s') and actions (a').

→ These equations serve as the basis for DP algorithms and allow us to estimate the values of states and state-action pairs, enabling policy evaluation and decision-making in RL.

a) Bellman Optimality Equation:-

The Bellman optimality equation is a fundamental concept in RL. that expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state. It provides a way to find the optimal value function and policy.

Optimal Value function:

The optimal value function, denoted as $V^*(s)$, represents the maximum expected return achievable from a give state s under an optimal policy:-

$$V^*(s) = \max\left[\Sigma\left[P(s'|s,a) * (R + \gamma * V^*(s'))\right]\right], \text{ for all actions } a$$

This equation states that the value of a state under an optimal policy is the maximum expected sum of immediate reward (R) and the discounted value of the next state ($V^*(s')$), considering all possible actions (a) and next states (s').

Expected Return for Best Action:

The expected return for the best action from a state s is the max expected sum of immediate reward (R) and the discounted value of the next state ($V^*(s')$) achievable by selecting the best action a^* from state s:-

$$Q^*(s,a^*) = \Sigma\left[P(s'|s,a^*) * (R + \gamma * V.^*(s'))\right]$$

Here, a^* represents the optimal action that maximizes the expected retu

Equivalence:

The Bellman optimality eq states that the value of a state under an optimo Policy ($V^*(s)$) is equal to the expected return for the best action from that state ; ($Q^*(s,a^*)$. -> $V^*(s) = \max\left[Q^*(s,a^*)\right]$

This equation shows that the optimal value function is equal to the max expected return achievable from a state by selecting the best action.

By solving the Bellman Optimality Equation, we can find the optimal value function and policy, leading to the selection of actions that maximize the expected return from each state, resulting in an optimal policy.