MARCH REMEDIAL

Name: KERFEGAR IRANI

Branch: TT AIML

ROLL NO : 18

REINFORCEMENT LEARNING

Reinforcement learning is a type of machine learning that involves training an agent to make decisions based on feedback from its environment. In reinforcement learning, the agent is given a set of possible actions and a reward signal that is used to evaluate the agent's performance.

The agent learns by trial and error, gradually improving its performance as it receives feedback from the environment. The goal is to maximize the cumulative reward over a series of actions. Reinforcement learning has been used in a variety of applications, including robotics, gaming, and recommendation systems.

The basic components of a reinforcement learning system include the agent, the environment, and the reward signal. The agent selects actions based on its current state, and the environment responds by providing a new state and a reward signal. The agent then uses this feedback to update its policy, which is the set of rules that govern its behavior.

Several algorithms have been developed for reinforcement learning, including Q-learning, policy gradients, and actor-critic methods. These algorithms differ in their approach to updating the agent's policy, and the choice of algorithm depends on the specific problem being addressed.

ORIGIN OF RL

The origin of reinforcement learning can be traced back to the mid-20th century, with the work of behaviorists such as B.F. Skinner and Edward Thorndike. Skinner's experiments with operant conditioning, in which animals were trained to perform certain behaviors through a system of rewards and punishments, laid the foundation for the idea of using rewards to shape behavior.

In the 1950s and 1960s, the concept of reinforcement learning was further developed by researchers in the field of artificial intelligence. The earliest algorithms for reinforcement learning were based on dynamic programming, a mathematical technique for solving optimization problems.

In recent years, reinforcement learning has gained renewed attention and interest due to advances in computing power and the availability of large datasets. Reinforcement learning has been applied to a wide range of problems, from robotics and game playing to natural language processing and recommendation systems.

As the field continues to evolve, researchers are exploring new techniques and algorithms to address the challenges of reinforcement learning, including exploration-exploitation tradeoffs, sample inefficiency, and generalization to new environments.

HYBRID SYSTEMS

Hybrid systems are systems that combine elements of two or more different types of systems. These systems are becoming increasingly popular in a wide range of fields, from engineering and computer science to healthcare and finance. In this cue card, we will explore the basics of hybrid systems, their applications, advantages, and challenges. Example: a hybrid vehicle that combines a gasoline engine with an electric motor.

Hybrid systems are a powerful tool for solving complex problems and achieving better performance than individual systems. They have applications in many fields, but they also present challenges related to complexity, integration, and cost. As technology continues to evolve, we can expect hybrid systems to become even more prevalent in the future.

INTEGRATION OF NEURAL NETWORKS

Neural networks are a powerful tool for solving complex problems in a wide range of fields, including image recognition, natural language processing, and predictive analytics. However, in some cases, a single neural network may not be sufficient to achieve the desired results.The process of combining multiple neural networks to improve performance is called integration of neural network.

Example: combining a convolutional neural network (CNN) with a recurrent neural network (RNN) to recognize images and generate captions.
Types of integration of neural networks:
Ensemble learning: combining multiple neural networks with the same architecture and training data.
Stacked generalization: using multiple neural networks to make predictions, and then combining the predictions.
Multi-task learning: training a single neural network to perform multiple tasks.

Integration of neural networks is an important technique for improving performance and handling complex problems. There are several types of integration, each with its own advantages and challenges. As neural networks continue to be developed and applied in new fields, we can expect integration to become an increasingly important tool for achieving better results.

History of RL

Reinforcement learning is a branch of machine learning that focuses on learning through interaction with an environment. It has become a popular area of research in recent years, but the history of reinforcement learning can be traced back to the mid-20th century.

Early work in reinforcement learning:
B.F. Skinner's work on operant conditioning in the 1930s and 1940s laid the foundation for reinforcement learning.
Learning automata: In the 1950s, Richard Bellman and others developed the concept of learning automata, which were the first computational models for reinforcement learning.
Breakthroughs in reinforcement learning:
Q-learning: In 1989, Chris Watkins proposed Q-learning, which is a simple yet powerful algorithm for reinforcement learning.
TD learning: In the early 1990s, Richard Sutton proposed temporal difference (TD) learning.
AlphaGo: In 2016, AlphaGo, an artificial intelligence program developed by Google DeepMind, defeated the world champion in the board game Go using a combination of deep neural networks and reinforcement learning.

The history of reinforcement learning spans several decades and includes numerous breakthroughs and applications. As the field continues to evolve, researchers are working to address challenges related to sample efficiency, interpretability, and multi-agent learning, among others. Reinforcement learning has the potential to revolutionize many areas, from robotics and beautonomous driving to game playing and beyond.

RL VS Supervised learning

Reinforcement learning and supervised learning are two important branches of machine learning. While both involve learning from data, they have different goals and methods.

Goal: Reinforcement learning is focused on learning a policy that maximizes long-term reward, while supervised learning is focused on learning a mapping between input and output data.
Feedback: Reinforcement learning algorithms receive feedback in the form of rewards or punishments, while supervised learning algorithms receive feedback in the form of labeled examples.
Data: Reinforcement learning algorithms learn from interactions with an environment, while supervised learning algorithms learn from a fixed dataset.

AGENTS ON RL

In reinforcement learning, an agent is an entity that interacts with its environment and learns to take actions that maximize a reward signal. The agent is typically implemented as a computer program or algorithm, and it operates in a dynamic environment that provides feedback in the form of rewards.

The agent's goal is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time. The agent observes the current state of the environment, selects an action, and receives a reward based on the quality of its decision. This process is repeated over multiple iterations, allowing the agent to gradually improve its policy.

Agents in reinforcement learning can be classified into several categories based on their approach to learning and decision-making like Model-based agents, Model-free agents, Exploration vs. exploitation agents: Multi-agent systems

Overall, the design and implementation of reinforcement learning agents is a critical aspect of the field, and it has implications for a wide range of applications, from robotics and gaming to business and finance.

ENVIRONMENT ON RL

In reinforcement learning, the environment is the external system in which an agent operates and learns to take actions that maximize a reward signal. The environment can be physical, such as a robot navigating a maze, or virtual, such as a game-playing agent interacting with a video game.

The environment can be formalized as a Markov Decision Process (MDP), which consists of a set of states, a set of actions, a reward function, and a transition function. The reward function maps each state-action pair to a numerical reward signal that reflects the quality of the agent's decision. The transition function specifies the probability of moving from one state to another when the agent takes a particular action.

The environment can be classified into several categories based on its characteristics, Fully observable, Partially observable , Deterministic, Stochastic ,Episodic, Continuous

Overall, the environment is a critical component of reinforcement learning, as it provides the context in which the agent operates and learns to optimize its behavior. The choice of environment depends on the specific problem being addressed, and researchers continue to explore new environments and extensions of the basic MDP framework.

Name: Kartik Pal
Branch: TT AIML
Roll No. 31

MARCH REMEDIAL

Exploration V/s Exploitation

Exploration and exploitation are two fundamental concepts in reinforcement learning. The exploration-exploitation trade-off refers to the dilemma of choosing between exploring new actions and exploiting the current knowledge to maximize the reward.

Exploration refers to the process of trying out new actions to learn about the environment and improve the policy.

Exploitation, on the other hand, refers to the process of using the current knowledge to maximize the reward.

The exploration-exploitation trade-off is a crucial challenge in reinforcement learning, as it determines the balance between learning and exploiting the current knowledge1. Too much exploration can lead to slow learning and low reward, while too much exploitation can lead to suboptimal policies and missed opportunities for higher reward1.

Dynamic programming

Dynamic programming is a technique for solving complex problems by breaking them down into smaller subproblems. It is a recursive process that involves solving each subproblem independently and using the solutions to solve the larger problem.

Dynamic programming is widely used in various fields, such as computer science, engineering, and finance. It is particularly useful for solving optimization problems, where the goal is to find the best solution among a set of possible solutions.
Dynamic programming algorithms can be classified into two categories: top-down and bottom-up.Top-down algorithms, also known as memorization, start with the largest subproblem and recursively solve smaller subproblems until the base case is reached.
Bottom-up algorithms, also known as tabulation, start with the base case and iteratively compute the solutions for larger subproblems3.

Markov State

A Markov state is a state in a Markov chain where the probability of transitioning to another state depends only on the current state and not on any previous states. In other words, the future state of the system depends only on the present state and not on how the system arrived at the present state.
Markov chains are used to model a wide range of phenomena, including games, finance, and biology. For example, a game of Monopoly can be modeled as a Markov chain, where each state represents the current position of the player on the board, and the probabilities of moving to other positions depend only on the current position and not on any previous positions.
In finance, Markov chains can be used to model credit card risk. Multi-state Markov models can be used to evaluate credit card risk by investigating the characteristics of the cardholders and their payment behavior.

Policy Iteration Algorithm

The policy iteration algorithm is a method used to solve Markov decision processes (MDPs). It involves two steps: policy evaluation and policy improvement. In the policy evaluation step, the value function of a given policy is computed. In the policy improvement step, a new policy is derived by selecting the action that maximizes the expected value of the next state, given the current state and the current policy.
The policy iteration algorithm is guaranteed to converge to the optimal policy in a finite number of iterations. However, it can be computationally expensive, especially for large MDPs.
The policy iteration algorithm is widely used in various fields, including robotics, finance, and healthcare. For example, it can be used to optimize the treatment of chronic diseases by determining the optimal sequence of interventions based on the patient's current state

Generalized Policy Iteration

Generalized policy iteration (GPI) is a framework for solving Markov decision processes (MDPs) that combines policy evaluation and policy improvement. In GPI, the policy is iteratively improved by evaluating the value function of the current policy and then updating the policy to be greedy with respect to the value function.

GPI can be used with various algorithms, such as value iteration and policy iteration. Value iteration is a GPI algorithm that iteratively computes the optimal value function and the optimal policy. Policy iteration is another GPI algorithm that iteratively computes the value function and the policy until convergence to the optimal solution.

GPI has been extended to handle more complex MDPs, such as partially observable MDPs (POMDPs) and multi-objective MDPs. For example, geometric policy iteration is a GPI algorithm that uses geometric properties to solve discounted MDPs.

GPI is a powerful framework for solving MDPs, and it has been applied in various fields, such as robotics, finance, and healthcare. For example, GPI can be used to optimize the treatment of chronic diseases by determining the optimal sequence of interventions based on the patient's current state.

Neuro Fuzzy Hybrid System

A neuro-fuzzy hybrid system is a combination of artificial neural networks (ANNs) and fuzzy logic systems (FLSs) that can be used for modeling and control applications. The combination of ANNs and FLSs allows for the integration of the strengths of both systems, such as the ability of ANNs to learn from data and the ability of FLSs to handle uncertainty and imprecision.

One example of a neuro-fuzzy hybrid system is the coactive neuro-fuzzy inference system (CANFIS), which combines ANNs and FLSs to create a hybrid model that can be used for classification, prediction, and control. CANFIS uses ANNs to learn the input-output mapping and FLSs to handle the uncertainty and imprecision in the input and output variables.

Neuro-fuzzy hybrid systems have been used in various fields, such as robotics, finance, and healthcare. For example, neuro-fuzzy hybrid systems can be used to control the movement of robots by learning from sensory data and handling uncertainty in the environment

Optimal Policy

In a Markov decision process (MDP), the optimal policy is defined as the policy that maximizes the expected sum of rewards over time. The optimal policy is a function that selects an action for every possible state, and it is independent of the actions taken in different states.

The optimal policy can be found using various algorithms, such as value iteration and policy iteration. These algorithms iteratively compute the value function or the policy until convergence to the optimal solution.

In a partially observable Markov decision process (POMDP), the optimal policy is a sequence of actions that maximizes the expected sum of rewards, given the current observation and the history of observations and actions.

The optimal policy is a fundamental concept in reinforcement learning, where an agent learns to interact with an environment by maximizing the cumulative reward. The agent learns a policy that maps states to actions, and the goal is to find the optimal policy that maximizes the expected reward over time.

Agent-Environment Interaction

In reinforcement learning, the agent-environment interaction is the process by which an agent learns to take actions in an environment to maximize the cumulative reward. The agent interacts with the environment by observing the current state, selecting an action, receiving a reward, and transitioning to a new state.

The environment is everything outside the agent, including the state, the reward, and the transition dynamics1. The agent and the environment interact continually, with the agent selecting actions and the environment providing feedback in the form of rewards and new states.

The agent-environment interaction is a fundamental concept in reinforcement learning, as it determines the learning process and the behavior of the agent. The goal of the agent is to learn a policy that maps states to actions, such that the expected cumulative reward is maximized over time.

The agent-environment interaction can be modeled as a Markov decision process (MDP), where the state, action, reward, and transition dynamics satisfy the Markov property. The MDP framework provides a formal way to define the agent-environment interaction and to compute the optimal policy using dynamic programming or Monte Carlo methods.

Name: Vipul Jaiswal
Branch: TT AIML
Roll No. 19

CUE CARD

ANFIS Architecture

ANFIS (Adaptive Neuro Fuzzy Inference System) is an artificial neural network that combines fuzzy logic and neural networks to approximate nonlinear functions.
The ANFIS architecture consists of five layers: fuzzy layer, product layer, normalized layer, de-fuzzy layer, and total output layer.
The first layer takes input values, and the last layer produces the output. The ANFIS model has learning capability and can be used as a universal estimator.
It has potential applications in various fields, including energy management systems.

Neuro Fuzzy Hybrid Systems

Neuro Fuzzy Hybrid Systems are intelligent systems that combine at least two intelligent technologies like Fuzzy Logic, Neural Networks, Genetic Algorithms, reinforcement learning, etc.
The Neuro-fuzzy system is based on a fuzzy system that is trained on the basis of the working of neural network theory. A neuro-fuzzy system can be seen as a 3-layer feedforward neural network, where the first layer represents input variables, the middle (hidden) layer represents fuzzy sets, and the output layer represents the output of the system.
These systems are used for pattern classification and feature reduction.

Agent-Environment Interaction

Agent-environment interaction refers to the continuous interaction between an agent and its environment. The agent is the decision-maker, while the environment is everything outside the agent.
This interaction is modeled as two coupled dynamical systems in which the agent selects actions and the environment responds to those actions and presents new situations to the agent. In reinforcement learning, the environment is typically formulated as a finite-state Markov Decision Process.
Indirect agent-agent interaction through informational signals written in the environment is called stigmergy.

Policy Iteration

Policy iteration is an algorithm used in reinforcement learning to find the optimal policy that maximizes the long-term reward. Unlike other algorithms, policy iteration manipulates the policy directly instead of finding it indirectly via the optimal value function.
The algorithm starts by choosing an arbitrary policy and then computes the value function of the policy by solving the linear equations.
The process is repeated until the optimal policy is found.
Policy iteration is often compared to value iteration, which combines two phases of the policy iteration into a single update operation.

Policy Environment

In reinforcement learning, a policy is a strategy or mapping that an agent uses to determine the actions to be taken in pursuit of its goals.
It is a mapping from the current environment observation to a probability distribution of the actions to be taken.
A value function is another important concept in reinforcement learning, which is a mapping from an environment observation to the expected cumulative long-term reward of a policy.
The policy is crucial in determining an agent's actions in an environment to maximize its long-term reward.

Off-Policy

Off-policy is a term used in reinforcement learning, which is a type of machine learning.
Off-policy learning algorithms evaluate and improve a policy that is different from the policy used for action selection.
This means that the agent learns the value of the optimal policy independently of its actions.
Off-policy methods use a behavioral policy to explore the environment and collect samples, generating the agent's behavior and a second policy.
However, off-policy deep reinforcement learning algorithms can introduce errors due to extrapolation, which can drastically offset the performance of the off-policy agent.

Methods for Model Free Prediction

There are several methods for model-free prediction. Some of these methods are: Random Forest, AdaBoost, XGBoost, Support Vector Machines, Neural Network, Super Learner.
These are all model-free machine learning algorithms that can be used for prediction without explicitly modeling the underlying system.
Another method for model-free prediction is the normalizing and variance stabilizing (NoVaS) method, which is a more robust and accurate prediction technique.
The model-free prediction methods typically employ cross-validation and sample re-use to improve their accuracy.
The key to successful model-free prediction is to transform the non-i.i.d. set-up to an i.i.d. set-up.

Name: Banti Pathak
Branch: TT AIML
Roll No. 36

MARCH REMEDIAL

Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a subfield of machine learning that combines deep learning techniques with reinforcement learning to enable an agent to learn through trial and error in a complex environment.

The use of deep neural networks allows the agent to learn from high-dimensional input spaces, such as images or video, and make decisions based on these inputs.

DRL has been successfully applied to a wide range of applications, including game playing, robotics, natural language processing, and self-driving cars. Some well-known DRL algorithms include Deep Q-Networks (DQN), Actor-Critic, and Proximal Policy Optimization (PPO).

Replay Buffer

A replay buffer is a data structure used in reinforcement learning (RL) that stores a history of the agent's experience. It is used to improve the efficiency and stability of the learning process, especially in deep reinforcement learning algorithms.

During training, the agent's experiences, consisting of observations, actions taken, rewards received, and resulting next state, are stored in the replay buffer. The agent then samples mini-batches of experiences from the buffer to train its neural network, instead of updating the network on every experience as it occurs. Replay buffer helps to address the problem of temporal correlation in the agent's experiences by breaking the sequential dependency of experiences.

Deep Q Learning

Deep Q-Learning is a deep reinforcement learning algorithm that combines the Q-learning algorithm, which is a model-free algorithm for learning optimal actions in a Markov decision process (MDP), with deep neural networks to handle high-dimensional state spaces.

In deep Q-learning, a deep neural network is used to approximate the Q-value function, which estimates the expected reward for taking an action in a given state. The Q-value function is updated iteratively using a variant of the Bellman equation to minimize the mean squared error between the predicted Q-values and the actual Q-values.

Deep Q Network

Deep Q-Network (DQN) is a deep reinforcement learning algorithm that combines deep neural networks with Q-learning, an off-policy value-based reinforcement learning algorithm, to approximate the optimal Q-value function in a Markov decision process (MDP).

In DQN, the Q-value function is represented by a deep neural network that takes the state as input and outputs the Q-value for each possible action. The network is trained using a variant of Q-learning that uses experience replay and a target network to improve learning stability. The target network is a copy of the Q-network that is periodically updated with the Q-network's parameters to reduce the correlation between the current and target Q-values.

Name: Shreyash Salunke
Branch: TT AIML
Roll No: 39

MARCH REMEDIAL (Q-CARD)

Why Deep Reinforcement Learning?

Deep Reinforcement Learning (DRL) is a powerful and promising approach to machine learning that allows agents to learn complex behaviors through trial and error. Here are some reasons why DRL is an attractive approach:

- Can handle complex, high-dimensional state spaces.
- End-to-end learning.
- Adaptability.
- Reward-driven learning.
- Exploration and exploitation.

Overall, DRL has the potential to enable autonomous systems to learn complex behaviors in a wide range of applications, from video games to robotics and beyond.

Applications of Deep Reinforcement Learning:

Deep Reinforcement Learning (DRL) has a wide range of potential applications, including:

- Game playing
- Robotics
- Autonomous driving
- Personalized medicine
- Recommender systems
- Finance
- Natural language processing
- Industrial control systems

Overall, DRL has the potential to revolutionize many industries and enable the development of autonomous systems that can learn and adapt to complex environments.

What is Parameterized Representation?

In machine learning, parameterized representation refers to the use of a set of parameters to represent a function or model that maps inputs to outputs. The parameters are typically learned from data using an optimization algorithm, such as stochastic gradient descent.

Parameterized representations are used in many machine learning models, including neural networks, decision trees, and linear models. In a neural network, for example, the parameters correspond to the weights and biases of the network, which are used to compute the output of each neuron in the network. By adjusting the parameters, the network can learn to approximate a wide range of functions.

Role of reasoning in Cognitive Computing:

Reasoning plays a critical role in cognitive computing. It allows cognitive computing systems to make inferences, draw conclusions, and generate new knowledge from existing data. Following are some ways in which reasoning is used in cognitive computing:

- Pattern recognition
- Decision-making
- Natural language understanding
- Knowledge representation
- Problem-solving

Overall, reasoning is an essential component of cognitive computing. By enabling systems to understand and reason about complex data, reasoning makes it possible to create intelligent systems that can learn, adapt, and generate new knowledge.

Name: Aryan Singh
Branch: TT AI&ML
Roll no: 49

MARCH REMEDIAL

RL in finance	**Use cases for RL in finance**
In finance, RL can be used for a wide range of tasks, including algorithmic trading, portfolio optimization, risk management, fraud detection, and more. One example of how RL can be used in finance is for algorithmic trading. An RL agent can be trained to learn from historical price data and make decisions on buying or selling stocks based on the current market conditions. Another use case of RL in finance is for portfolio optimization. An RL agent can be trained to select the optimal mix of assets for a portfolio, based on historical data and other relevant factors such as risk tolerance and investment goals	Algorithmic Trading: RL can be used to train an agent to make buy or sell decisions based on market data. The agent learns from historical data, patterns and can take different actions and evaluate which action results in the highest reward. Risk Management: RL can be used for managing risk in financial institutions. An RL agent can be trained to monitor different risks such as market risk, credit risk, operational risk, and fraud. Pricing: RL can be used for dynamic pricing in financial markets. The agent can learn from historical data, market trends, and other relevant factors to determine the optimal price for financial products such as stocks, bonds, and options.
Advantages of CC over AI **Scalability:** Cloud computing provides the ability to scale up or down resources as needed, which can be beneficial for AI applications that require significant computing power. **Cost-Effective:** Cloud computing can be more cost-effective than building and maintaining an in-house infrastructure for AI applications, as it allows organizations to pay only for the resources they use. **Accessibility:** Cloud computing allows users to access AI applications from anywhere and at any time, as long as they have an internet connection. **Security:** Cloud providers invest heavily in security measures and compliance standards, which can help protect sensitive data and reduce the risk of security breaches.	**What is Data Modelling?** Data modeling is the process of creating a conceptual or logical representation of data and its relationships to other data elements. The goal of data modeling is to create a structured and organized view of data, which can be used to support various business processes and applications. **Types:** conceptual and logical. Conceptual data modeling is a high-level representation of data and its relationships, without taking into account the technical details of how the data will be implemented in a database or system. Logical data modeling, on the other hand, is a more detailed representation of data and its relationships, taking into account the technical aspects of how the data will be stored and processed in a database or system. It often involves the use of a data modeling notation or language, such as the Entity-Relationship (ER) model or Unified Modeling Language (UML).