Machine Learning: Symbol Based - **Introduction**

Symbol-based machine learning, also known as symbolic AI or symbolic reasoning, is a branch of artificial intelligence (AI) that focuses on the manipulation and interpretation of symbols or abstract representations to perform tasks. Unlike other approaches to machine learning, such as neural networks, which rely on statistical patterns and numerical data, symbol-based machine learning operates on explicit rules, logic, and symbols.

In symbol-based machine learning, knowledge is represented in the form of symbols and relationships between them, often organized in structures such as graphs, trees, or logical rules. These symbols can represent various entities, concepts, or features relevant to the task at hand. The system then uses logical reasoning, inference, and symbolic manipulation techniques to derive new knowledge or make decisions based on the existing knowledge.

Key components of symbol-based machine learning include:

- Symbolic Representation: Data and knowledge are represented using symbols, which can include logical propositions, predicates, objects, or any other abstract representation relevant to the problem domain.
- Logical Reasoning: Symbolic machine learning systems employ logical reasoning techniques such as deduction, induction, and abduction to draw conclusions, make inferences, and solve problems.
- Symbolic Manipulation: Algorithms are used to manipulate symbols and symbolic expressions, enabling tasks such as pattern matching, inference, and transformation of knowledge.
- Knowledge Representation: Symbol-based machine learning often relies on explicit knowledge representation frameworks, such as semantic networks, knowledge graphs, or rule-based systems, to organize and store knowledge in a structured form.
- Inference Engines: These are mechanisms that perform logical inference based on the rules and knowledge encoded in the system, enabling it to make decisions or generate new knowledge.

Symbol-based machine learning has been applied to various domains, including expert systems, natural language understanding, automated reasoning, and robotics. While it has its strengths in tasks that require explicit reasoning and interpretation of symbols, it also faces challenges in dealing

with uncertainty, handling large-scale data, and learning from raw sensory inputs, which are areas where statistical methods like neural networks excel.

In recent years, there has been increasing interest in integrating symbolic and subsymbolic approaches in AI, aiming to combine the strengths of both paradigms to address a wider range of tasks effectively. This interdisciplinary approach, often referred to as hybrid AI, seeks to leverage the advantages of both symbolic and connectionist methods to create more robust and flexible intelligent systems.

framework

In the context of symbol-based machine learning and artificial intelligence, a "framework" typically refers to a structured approach or a set of tools and methodologies for organizing and manipulating knowledge, reasoning, and decision-making. One such framework commonly used in symbolic AI is the frame-based approach.

A frame is a data structure used to represent knowledge about objects, concepts, or situations in a hierarchical manner. It consists of slots (attributes or properties) and fillers (values or instances) that describe the characteristics or features of the represented entity. Frames can be thought of as templates or prototypes for organizing and storing knowledge in a structured form.

Key features of a frame-based framework include:

Hierarchical Organization: Frames are often organized in a hierarchical structure, with more general frames at higher levels and more specific frames at lower levels. This hierarchical organization allows for the representation of concepts at different levels of abstraction and specialization.
Slots and Fillers: Each frame contains slots, which represent attributes or properties of the entity being described. These slots can have fillers, which are the values or instances that provide specific information about each attribute.
Inheritance: Frames can inherit properties from other frames in the hierarchy. This means that a specific frame can inherit slots and their values from more general frames higher up in the hierarchy. Inheritance allows for the sharing and reuse of knowledge across related concepts.

Default Values and Constraints: Frames can specify default values for slots, which are used when specific values are not provided. They can also include constraints that define valid values for slots or relationships between slots.
Frames and Instances: In a frame-based system, frames represent general concepts or categories, while instances represent specific examples or instances of those concepts. For example, "Car" might be a frame representing the concept of a car, while "Toyota Camry" could be an instance of the "Car" frame.
Reasoning and Inference: Frame-based systems often include mechanisms for reasoning and inference, allowing them to derive new knowledge or make decisions based on the knowledge represented in the frames.

Frame-based frameworks have been used in various AI applications, including expert systems, natural language understanding, and knowledge representation. They provide a flexible and intuitive way to organize and manipulate knowledge, making them particularly well-suited for domains where structured representation of knowledge is important. However, they may face challenges in handling complex and ambiguous knowledge or in dealing with large-scale data.

Decision Tree

A decision tree in machine learning is a versatile, interpretable algorithm used for predictive modelling. It structures decisions based on input data, making it suitable for both classification and regression tasks. This article delves into the components, terminologies, construction, and advantages of decision trees, exploring their applications and learning algorithms.

Decision Tree in Machine Learning

A decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data. It is a tree-like structure where each internal node tests on attribute, each branch

corresponds to attribute value and each leaf node represents the final decision or prediction. The decision tree algorithm falls under the category of supervised learning. They can be used to solve both **regression** and **classification problems**.

Decision Tree Terminologies

There are specialized terms associated with decision trees that denote various components and facets of the tree structure and decision-making procedure. :

- **Root Node:** A decision tree's root node, which represents the original choice or feature from which the tree branches, is the highest node.
- **Internal Nodes (Decision Nodes)**: Nodes in the tree whose choices are determined by the values of particular attributes. There are branches on these nodes that go to other nodes.
- **Leaf Nodes (Terminal Nodes)**: The branches' termini, when choices or forecasts are decided upon. There are no more branches on leaf nodes.
- **Branches (Edges)**: Links between nodes that show how decisions are made in response to particular circumstances.
- **Splitting**: The process of dividing a node into two or more sub-nodes based on a decision criterion. It involves selecting a feature and a threshold to create subsets of data.
- **Parent Node**: A node that is split into child nodes. The original node from which a split originates.

- **Child Node**: Nodes created as a result of a split from a parent node.
- **Decision Criterion**: The rule or condition used to determine how the data should be split at a decision node. It involves comparing feature values against a threshold.
- **Pruning**: The process of removing branches or nodes from a decision tree to improve its generalisation and prevent overfitting.

Understanding these terminologies is crucial for interpreting and working with decision trees in machine learning applications.

How Decision Tree is formed?

The process of forming a decision tree involves recursively partitioning the data based on the values of different attributes. The algorithm selects the best attribute to split the data at each internal node, based on certain criteria such as information gain or Gini impurity. This splitting process continues until a stopping criterion is met, such as reaching a maximum depth or having a minimum number of instances in a leaf node.

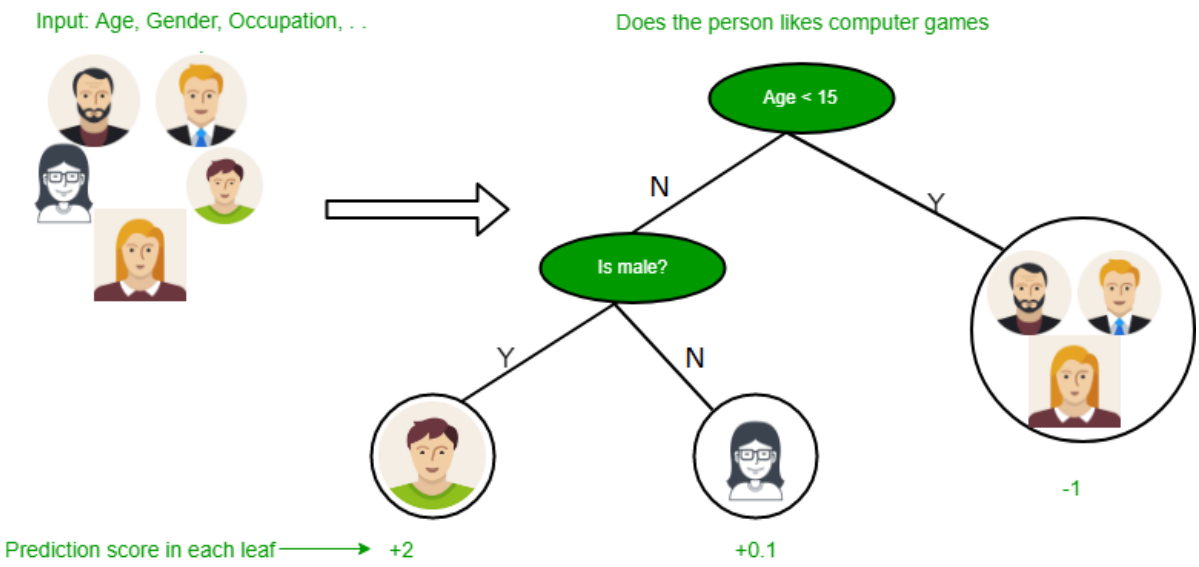
Why Decision Tree?

Decision trees are widely used in machine learning for a number of reasons:

- Decision trees are so versatile in simulating intricate decision-making processes, because of their interpretability and versatility.
- Their portrayal of complex choice scenarios that take into account a variety of causes and outcomes is made possible by their hierarchical structure.
- They provide comprehensible insights into the decision logic, decision trees are especially helpful for tasks involving categorisation and regression.
- They are proficient with both numerical and categorical data, and they can easily adapt to a variety of datasets thanks to their autonomous feature selection capability.
- Decision trees also provide simple visualization, which helps to comprehend and elucidate the underlying decision processes in a model.

Decision Tree Approach

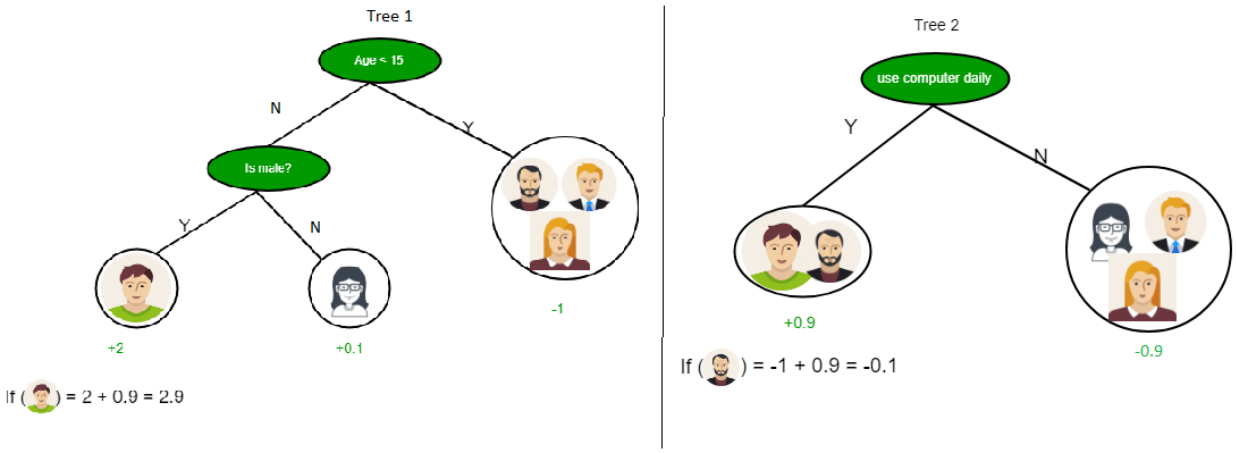
Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.



Below are some assumptions that we made while using the decision tree:

At the beginning, we consider the whole training set as the root.

- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values, records are distributed recursively.
- We use statistical methods for ordering attributes as root or the internal node.



As you can see from the above image the Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image, we are predicting the use of computer in the daily life of people. In the Decision Tree, the major challenge is the identification of the attribute for the root node at each level. This process is known as attribute selection.

Iterative Dichotomiser 3 (ID3) Algorithm From Scratch

In the realm of machine learning and data mining, decision trees stand as versatile tools for classification and prediction tasks. The ID3 (Iterative Dichotomiser 3) algorithm serves as one of the foundational pillars upon which decision tree learning is built. Developed by Ross Quinlan in the 1980s, ID3 remains a fundamental algorithm, forming the basis for subsequent tree-based methods like C4.5 and CART (Classification and Regression Trees).

Introduction to Decision Trees

Machine learning models called decision trees divide the input data recursively according to features to arrive at a decision. Every internal node symbolizes a

feature, and every branch denotes a potential result of that feature. It is simple to interpret and visualize thanks to the tree structure. Every leaf node makes a judgment call or forecast. To optimize information acquisition or limit impurity, the best feature is chosen at each stage of creation. Decision trees are adaptable and can be used for both regression and classification applications. Although they can overfit, this is frequently avoided by employing strategies like pruning.

Decision Trees

Before delving into the intricacies of the ID3 algorithm, let's grasp the essence of decision trees. Picture a tree-like structure where each internal node represents a test on an attribute, each branch signifies an outcome of that test, and each leaf node denotes a class label or a decision. Decision trees mimic human decision-making processes by recursively splitting data based on different attributes to create a flowchart-like structure for classification or regression.

ID3 Algorithm

A well-known decision tree approach for machine learning is the Iterative Dichotomiser 3 (ID3) algorithm. By choosing the best characteristic at each node to partition the data depending on information gain, it recursively constructs a tree. The goal is to make the final subsets as homogeneous as possible. By

choosing features that offer the greatest reduction in entropy or uncertainty, ID3 iteratively grows the tree. The procedure keeps going until a halting requirement is satisfied, like a minimum subset size or a maximum tree depth. Although ID3 is a fundamental method, other iterations such as C4.5 and CART have addressed.

How ID3 Works

The ID3 algorithm is specifically designed for building decision trees from a given dataset. Its primary objective is to construct a tree that best explains the relationship between attributes in the data and their corresponding class labels.

1. Selecting the Best Attribute

- ID3 employs the concept of entropy and information gain to determine the attribute that best separates the data. Entropy measures the impurity or randomness in the dataset.
- The algorithm calculates the entropy of each attribute and selects the one that results in the most significant information gain when used for splitting the data.

2. Creating Tree Nodes

- The chosen attribute is used to split the dataset into subsets based on its distinct values.

- For each subset, ID3 recurses to find the next best attribute to further partition the data, forming branches and new nodes accordingly.

3. Stopping Criteria

- The recursion continues until one of the stopping criteria is met, such as when all instances in a branch belong to the same class or when all attributes have been used for splitting.

4. Handling Missing Values

- ID3 can handle missing attribute values by employing various strategies like attribute mean/mode substitution or using majority class values.

5. Tree Pruning

- Pruning is a technique to prevent overfitting. While not directly included in ID3, post-processing techniques or variations like C4.5 incorporate pruning to improve the tree's generalization.

Mathematical Concepts of ID3 Algorithm

Now let's examine the formulas linked to the main theoretical ideas in the ID3 algorithm:

1. Entropy

A measure of disorder or uncertainty in a set of data is called entropy. Entropy is a tool used in ID3 to measure a dataset's disorder or impurity. By dividing the data into as homogenous subsets as feasible, the objective is to minimize entropy.

For a set S with classes {c1, c2, ..., cn}, the entropy is calculated as:

Where, p_i is the proportion of instances of class c_i in the set.

2. Information Gain

A measure of how well a certain quality reduces uncertainty is called Information Gain. ID3 splits the data at each stage, choosing the property that maximizes Information Gain. It is computed using the distinction between entropy prior to and following the split.

Information Gain measures the effectiveness of an attribute A in reducing uncertainty in set S.

Where, $|S_v|$ is the size of the subset of S for which attribute A has value v.

3. Gain Ratio

Gain Ratio is an improvement on Information Gain that considers the inherent worth of characteristics that have a wide range of possible values. It deals with

the bias of Information Gain in favor of characteristics with more pronounced values.

Iterative Dichotomiser 3 (ID3) Implementation using Python

Let's create a simplified version of the ID3 algorithm from scratch using Python.

Importing Libraries

Importing the necessary libraries:

Python3

```
from collections import Counter
```

```
import numpy as np
```

- **collections** for the Counter class to count occurrences.
- **numpy** as np for numerical operations and array handling.

Defining Node Class

Python3

```
class Node:

    def __init__(self, feature=None, value=None, results=None,
true_branch=None, false_branch=None):

        self.feature = feature  # Feature to split on
```

```
        self.value = value       # Value of the feature to split on

        self.results = results  # Stores class labels if node is a
leaf node

        self.true_branch = true_branch  # Branch for values that are
True for the feature

        self.false_branch = false_branch  # Branch for values that
are False for the feature
```

The provided Python code defines a class called Node for constructing nodes in a decision tree. Each node encapsulates information crucial for decision-making within the tree. The feature attribute signifies the feature used for splitting, while value stores the specific value of that feature for the split. In the case of a leaf node, results holds class labels. The node also has branches, with true_branch representing the path for values evaluating to True for the feature, and false_branch for values evaluating to False. This class forms a fundamental building block for creating decision trees, enabling the representation of decision points and outcomes in a hierarchical structure.

Entropy Calculation Function

Python3

```
def entropy(data):

    counts = np.bincount(data)

    probabilities = counts / len(data)

    entropy = -np.sum([p * np.log2(p) for p in probabilities if p > 0])

    return entropy
```

The entropy function calculates the entropy of a given dataset using the formula for information entropy. It first computes the counts of occurrences for each unique element in the dataset using np.bincount. Then, it calculates the probabilities of each element and uses these probabilities to compute the

entropy using the standard formula – . The function ensures that the logarithm is not taken for zero probabilities, avoiding mathematical errors. The result is the entropy value for the input dataset, reflecting its degree of disorder or uncertainty.

Splitting Data Function

Python3

```
def split_data(X, y, feature, value):

    true_indices = np.where(X[:, feature] <= value)[0]

    false_indices = np.where(X[:, feature] > value)[0]

    true_X, true_y = X[true_indices], y[true_indices]
```

```
    false_X, false_y = X[false_indices],
y[false_indices]



    return true_X, true_y, false_X, false_y
```

The split_data function divides a dataset into two subsets based on a specified feature and threshold value. It uses NumPy to identify indices where the feature values satisfy the condition (<= value for the true branch and > value for the false branch). Then, it extracts the corresponding subsets for features (true_X and false_X) and labels (true_y and false_y). The function returns these subsets, enabling the partitioning of data for further use in constructing a decision tree.

Building the Tree Function

Python3

```
def build_tree(X, y):

    if len(set(y)) == 1:

        return Node(results=y[0])




    best_gain = 0

    best_criteria = None

    best_sets = None

    n_features = X.shape[1]
```

```
    current_entropy = entropy(y)




    for feature in range(n_features):


        feature_values = set(X[:, feature])


        for value in feature_values:


            true_X, true_y, false_X, false_y = split_data(X, y,
feature, value)


            true_entropy = entropy(true_y)
```

```
            false_entropy = entropy(false_y)

            p = len(true_y) / len(y)

            gain = current_entropy - p * true_entropy - (1 - p) *
false_entropy

            if gain > best_gain:

                best_gain = gain

                best_criteria = (feature, value)

                best_sets = (true_X, true_y, false_X, false_y)
```

```
    if best_gain > 0:

        true_branch = build_tree(best_sets[0], best_sets[1])

        false_branch = build_tree(best_sets[2], best_sets[3])

        return Node(feature=best_criteria[0],
value=best_criteria[1], true_branch=true_branch,
false_branch=false_branch)

    return Node(results=y[0])
```

The build_tree function recursively constructs a decision tree using the ID3 algorithm. It first checks if the labels in the current subset are homogenous; if so, it creates a leaf node with the corresponding class label. Otherwise, it iterates

through all features and values, calculating information gain for each split and identifying the one with the highest gain. The function then recursively calls itself to build the true and false branches using the best split criteria. The resulting decision tree is constructed and returned. The process continues until further splits do not yield positive information gain, resulting in the creation of leaf nodes.

Prediction Function

Python3

```
def predict(tree, sample):

    if tree.results is not None:

        return tree.results
```

```
    else:

        branch = tree.false_branch

        if sample[tree.feature] <= tree.value:

            branch = tree.true_branch

        return predict(branch, sample)
```

The predict function uses a trained decision tree to predict the class label for a given sample. It recursively navigates the tree by checking if the current node is a leaf node (indicated by non-None results). If it is a leaf, it returns the class labels. Otherwise, it determines the next branch to traverse based on the feature value of the sample compared to the node's splitting criteria. The function then calls itself with the appropriate branch until a leaf node is reached, providing the final predicted class labels for the input sample.

Dataset and Tree Building

Python3

```
X = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])

y = np.array([1, 1, 0, 0])



# Building the tree

decision_tree = build_tree(X, y)
```

The code creates a dataset X with binary features and their corresponding labels y. Then, it constructs a decision tree using the build_tree function, which recursively builds the tree using the ID3 algorithm based on the provided

dataset. The resulting decision_tree is the root node of the constructed decision tree.

Prediction

Python3

```
sample = np.array([1, 0])

prediction = predict(decision_tree, sample)

print(f"Prediction for sample {sample}:
{prediction}")
```

Output:

```
Prediction for sample [1 0]: 1
```

- Predicts the class label for the sample using the built decision tree and prints the prediction.
- If we want to predict the class label for the sample [1, 0], the algorithm will traverse the decision tree starting from the root node. As Feature 0 is 1 (greater than 0.5), it will follow the False branch, and thus the prediction will be 1 (Class 1).

Advantages and Limitations of ID3

Advantages

- **Interpretability:** Decision trees generated by ID3 are easily interpretable, making them suitable for explaining decisions to non-technical stakeholders.
- **Handles Categorical Data:** ID3 can effectively handle categorical attributes without requiring explicit data preprocessing steps.
- **Computationally Inexpensive:** The algorithm is relatively straightforward and computationally less expensive compared to some complex models.

Limitations

- **Overfitting:** ID3 tends to create complex trees that may overfit the training data, impacting generalization to unseen instances.

- **Sensitive to Noise:** Noise or outliers in the data can lead to the creation of non-optimal or incorrect splits.
- **Binary Trees Only:** ID3 constructs binary trees, limiting its ability to represent more complex relationships present in the data directly.

Conclusion

The ID3 algorithm laid the groundwork for decision tree learning, providing a robust framework for understanding attribute selection and recursive partitioning. Despite its limitations, ID3's simplicity and interpretability have paved the way for more sophisticated algorithms that address its drawbacks while retaining its essence.

As machine learning continues to evolve, the ID3 algorithm remains a crucial piece in the mosaic of tree-based methods, serving as a stepping stone for developing more advanced and accurate models in the quest for efficient data analysis and pattern recognition.

In the context of artificial intelligence (AI) and machine learning, the concepts of inductive bias and learnability are fundamental to understanding how AI systems generalize from training data to unseen instances and how feasible it is for these systems to learn from data.

Inductive bias and Learnability,

Inductive Bias:

Definition: In AI, inductive bias refers to the set of assumptions, biases, or constraints that are built into the learning algorithm or the AI system's architecture, guiding it to prefer certain hypotheses or solutions over others.

Importance: Inductive bias is crucial because it helps AI systems generalize from limited training data to make accurate predictions or decisions on unseen data. It acts as a regularizer, preventing the AI system from memorizing the training data and instead promoting generalizable patterns or concepts.

Examples of Inductive Bias in AI:

> Model Architecture: Different types of machine learning models, such as neural networks, decision trees, or support vector machines, come with inherent biases based on their architectures. For example, neural networks are biased towards learning hierarchical representations, while decision trees are biased towards learning simple, interpretable rules.
> Feature Engineering: The choice of features or input representations can introduce bias into the learning process. Feature selection or extraction methods may prioritize certain types of information over others, influencing the model's learning behavior.

Regularization Techniques: Regularization methods, such as L1 or L2 regularization in neural networks, introduce bias by penalizing complex models, encouraging simpler solutions that are less prone to overfitting.

Learning Algorithms: The optimization algorithms used during training may have inherent biases, such as stochastic gradient descent's tendency to converge to certain types of solutions based on the learning rate and other hyperparameters.

Learnability:

Definition: Learnability in AI refers to the theoretical property of a learning algorithm or AI system that determines its ability to learn a target concept or task effectively from training data within a given hypothesis space, computational constraints, and noise level.

Importance: Understanding the learnability of AI systems is essential for assessing their capabilities, limitations, and generalization performance. It provides insights into the sample complexity, computational complexity, and robustness of learning algorithms.

Factors Affecting Learnability in AI:

Sample Complexity: The number of training examples required for an AI system to learn a target concept or task with a certain level of accuracy. Lower sample complexity indicates higher learnability.

Computational Complexity: The computational resources, such as time and memory, required for training and inference. Lower computational complexity contributes to higher learnability.

Noise and Ambiguity: The presence of noise, uncertainty, or ambiguity in the data can affect the learnability of AI systems by making it harder to distinguish between relevant and irrelevant information or patterns.

Model Complexity: The complexity of the AI system's hypothesis space or model architecture influences learnability. More complex models may have higher expressive power but require more data and computational resources to learn effectively.

By understanding and managing inductive bias and learnability in AI systems, researchers and practitioners can develop more robust, interpretable, and generalizable AI solutions across various domains and applications.

Knowledge and Learning

Knowledge and learning are fundamental aspects of artificial intelligence (AI), playing crucial roles in how AI systems operate, reason, and make decisions. Let's delve into each of these aspects in detail:

1. **Knowledge Representation**: Knowledge in AI refers to the information that an AI system possesses about the world. This information can be represented in various forms, including:

   - **Symbolic Representation**: In this approach, knowledge is represented using symbols, such as logical propositions or semantic networks. For example, in a medical diagnosis system, symptoms, diseases, and their relationships could be represented using symbols and logical rules.

   - **Statistical Representation**: With the rise of machine learning and statistical methods, knowledge can also be represented using statistical models, such as probabilistic graphical models or deep neural networks. These models learn patterns and relationships from data, allowing AI systems to make predictions and decisions based on statistical inference.

- **Hybrid Representation**: Many AI systems use a combination of symbolic and statistical representations to capture different aspects of knowledge effectively. For instance, a natural language understanding system might use symbolic rules for grammar and semantics alongside neural networks for learning word embeddings and contextual information.

2. **Learning**: Learning in AI refers to the process through which AI systems acquire knowledge or improve their performance over time. There are several types of learning commonly used in AI:

- **Supervised Learning**: In supervised learning, the AI system learns a mapping from input data to output labels based on labeled examples provided during training. It aims to generalize from the given examples to make predictions or decisions on unseen data.

- **Unsupervised Learning**: Unsupervised learning involves learning patterns and structures from unlabeled data. The system tries to find hidden patterns or clusters in the data without explicit guidance.

- **Reinforcement Learning**: Reinforcement learning involves learning optimal decision-making strategies through interaction with an environment. The system learns to take actions that maximize cumulative rewards, often through trial and error.

- **Semi-supervised Learning**: This learning paradigm combines elements of supervised and unsupervised learning, where the system learns from a combination of labeled and unlabeled data.

- **Transfer Learning**: Transfer learning involves transferring knowledge from one domain or task to another. Pre-trained models, such as language models like GPT (like the one you're interacting with), are often fine-tuned on specific tasks, leveraging knowledge learned from large-scale datasets.

- **Meta-Learning**: Meta-learning focuses on learning to learn. The system learns how to adapt its learning process or acquire new skills more efficiently based on prior experience with different tasks.

3. **Knowledge Acquisition**: Knowledge acquisition is the process of gathering, extracting, and formalizing knowledge for use in AI systems. This can involve various methods:

- **Manual Knowledge Engineering**: Experts encode knowledge into AI systems manually, often using symbolic representations and rules. This approach is time-consuming and requires expertise but can be precise and interpretable.

- **Machine Learning from Data**: AI systems can learn knowledge directly from data through various machine learning algorithms. This approach is data-driven and can capture complex patterns but may lack interpretability and require large amounts of labeled data.

- **Knowledge Base Population**: This involves automatically extracting knowledge from unstructured sources such as text documents, databases, or the web, and populating structured knowledge bases or ontologies.

4. **Reasoning and Inference**: Once knowledge is acquired and represented, AI systems use reasoning and inference mechanisms to derive new knowledge, make decisions, or answer queries. This can involve logical deduction, probabilistic reasoning, or heuristic-based approaches depending on the nature of the knowledge representation and the problem domain.

In summary, knowledge and learning are core components of AI systems, enabling them to understand the world, make decisions, and adapt to new situations. The interaction between knowledge representation, learning, reasoning, and inference forms the foundation of AI's ability to perform intelligent tasks.

Unsupervised learning

What is Unsupervised learning?

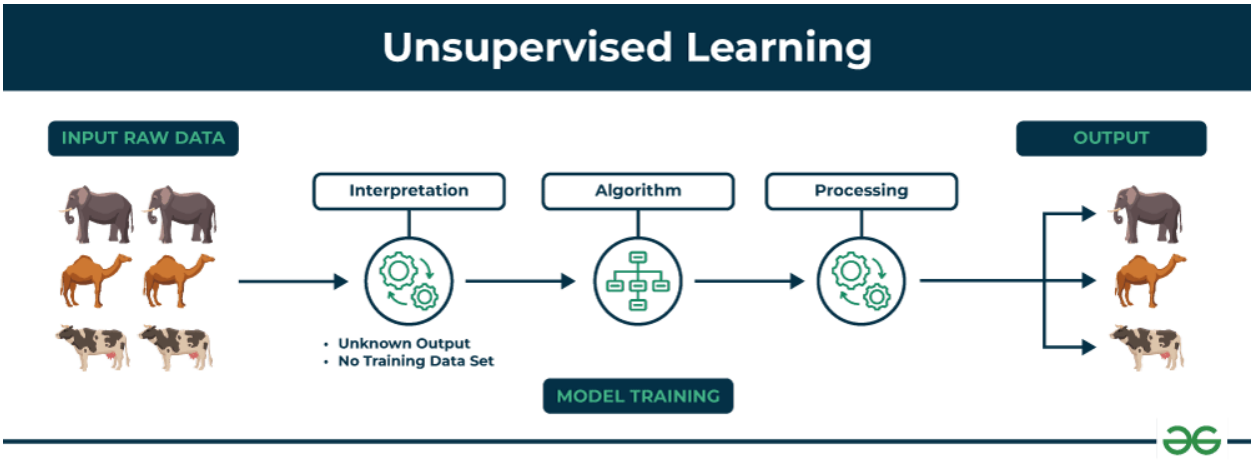
Unsupervised learning is a type of machine learning that learns from unlabeled data. This means that the data does not have any pre-existing labels or categories. The goal of unsupervised learning is to discover patterns and relationships in the data without any explicit guidance.

Unsupervised learning is the training of a machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of the machine is to group unsorted information according to similarities, patterns, and differences without any prior

training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore the machine is restricted to find the hidden structure in unlabeled data by itself.

You can use unsupervised learning to examine the animal data that has been gathered and distinguish between several groups according to the traits and actions of the animals. These groupings might correspond to various animal species, providing you to categorize the creatures without depending on labels that already exist.



Key Points

- Unsupervised learning allows the model to discover patterns and relationships in unlabeled data.
- Clustering algorithms group similar data points together based on their inherent characteristics.
- Feature extraction captures essential information from the data, enabling the model to make meaningful distinctions.
- Label association assigns categories to the clusters based on the extracted patterns and characteristics.

Example

Imagine you have a machine learning model trained on a large dataset of unlabeled images, containing both dogs and cats. The model has never seen an image of a dog or cat before, and it has no pre-existing labels or categories for these animals. Your task is to use unsupervised learning to identify the dogs and cats in a new, unseen image.

For instance, suppose it is given an image having both dogs and cats which it has never seen.

Thus the machine has no idea about the features of dogs and cats so we can't categorize it as 'dogs and cats '. But it can categorize them according to their similarities, patterns, and differences, i.e., we can easily categorize the above picture into two parts. The first may contain all pics having dogs in them and the second part may contain all pics having cats in them. Here you didn't learn anything before, which means no training data or examples.

It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabelled data.

Types of Unsupervised Learning

Unsupervised learning is classified into two categories of algorithms:

- Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Clustering

Clustering is a type of unsupervised learning that is used to group similar data points together. Clustering algorithms work by iteratively moving data points closer to their cluster centers and further away from data points in other clusters.

1. Exclusive (partitioning)
2. Agglomerative
3. Overlapping
4. Probabilistic

Clustering Types:-

1. Hierarchical clustering
2. K-means clustering
3. Principal Component Analysis
4. Singular Value Decomposition
5. Independent Component Analysis
6. Gaussian Mixture Models (GMMs)
7. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Association rule learning

Association rule learning is a type of unsupervised learning that is used to identify patterns in a data. Association rule learning algorithms work by finding relationships between different items in a dataset.

Some common association rule learning algorithms include:

- Apriori Algorithm

- Eclat Algorithm
- FP-Growth Algorithm

Evaluating Non-Supervised Learning Models

Evaluating non-supervised learning models is an important step in ensuring that the model is effective and useful. However, it can be more challenging than evaluating supervised learning models, as there is no ground truth data to compare the model's predictions to.

There are a number of different metrics that can be used to evaluate non-supervised learning models, but some of the most common ones include:

- Silhouette score: The silhouette score measures how well each data point is clustered with its own cluster members and separated from other clusters. It ranges from -1 to 1, with higher scores indicating better clustering.
- Calinski-Harabasz score: The Calinski-Harabasz score measures the ratio between the variance between clusters and the variance within clusters. It ranges from 0 to infinity, with higher scores indicating better clustering.
- Adjusted Rand index: The adjusted Rand index measures the similarity between two clusterings. It ranges from -1 to 1, with higher scores indicating more similar clusterings.
- Davies-Bouldin index: The Davies-Bouldin index measures the average similarity between clusters. It ranges from 0 to infinity, with lower scores indicating better clustering.

- F1 score: The F1 score is a weighted average of precision and recall, which are two metrics that are commonly used in supervised learning to evaluate classification models. However, the F1 score can also be used to evaluate non-supervised learning models, such as clustering models.

Application of Unsupervised learning

Non-supervised learning can be used to solve a wide variety of problems, including:

- Anomaly detection: Unsupervised learning can identify unusual patterns or deviations from normal behavior in data, enabling the detection of fraud, intrusion, or system failures.
- Scientific discovery: Unsupervised learning can uncover hidden relationships and patterns in scientific data, leading to new hypotheses and insights in various scientific fields.
- Recommendation systems: Unsupervised learning can identify patterns and similarities in user behavior and preferences to recommend products, movies, or music that align with their interests.
- Customer segmentation: Unsupervised learning can identify groups of customers with similar characteristics, allowing businesses to target marketing campaigns and improve customer service more effectively.
- Image analysis: Unsupervised learning can group images based on their content, facilitating tasks such as image classification, object

detection, and image retrieval.

Advantages of Unsupervised learning

- It does not require training data to be labeled.
- Dimensionality reduction can be easily accomplished using unsupervised learning.
- Capable of finding previously unknown patterns in data.
- Unsupervised learning can help you gain insights from unlabeled data that you might not have been able to get otherwise.
- Unsupervised learning is good at finding patterns and relationships in data without being told what to look for. This can help you learn new things about your data.

Disadvantages of Unsupervised learning

- Difficult to measure accuracy or effectiveness due to lack of predefined answers during training.
- The results often have lesser accuracy.
- The user needs to spend time interpreting and label the classes which follow that classification.
- Unsupervised learning can be sensitive to data quality, including missing values, outliers, and noisy data.
- Without labeled data, it can be difficult to evaluate the performance of unsupervised learning models, making it challenging to assess their

effectiveness.

Supervised vs. Unsupervised Machine Learning

Parameters	Supervised machine learning	Unsupervised machine learning
Input Data	Algorithms are trained using labeled data.	Algorithms are used against data that is not labeled
Computational Complexity	Simpler method	Computationally complex
Accuracy	Highly accurate	Less accurate
No. of classes	No. of classes is known	No. of classes is not known

Data Analysis	Uses offline analysis	Uses real-time analysis of data
Algorithms used	Linear and Logistics regression, Random forest, multi-class classification, decision tree, Support Vector Machine, Neural Network, etc.	K-Means clustering, Hierarchical clustering, KNN, Apriori algorithm, etc.
Output	Desired output is given.	Desired output is not given.
Training data	Use training data to infer model.	No training data is used.
Complex model	It is not possible to learn larger and more complex models than with	It is possible to learn larger and more complex models with unsupervised

	supervised learning.	learning.
Model	We can test our model.	We can not test our model.
Called as	Supervised learning is also called classification.	Unsupervised learning is also called clustering.
Example	Example: Optical character recognition.	Example: Find a face in an image.
Supervision	supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.

Conclusion

Supervised and unsupervised learning are two powerful tools that can be used to solve a wide variety of problems. Supervised learning is well-suited for tasks where the desired output is known, while unsupervised learning is well-suited for tasks where the desired output is unknown.

Reinforcement learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

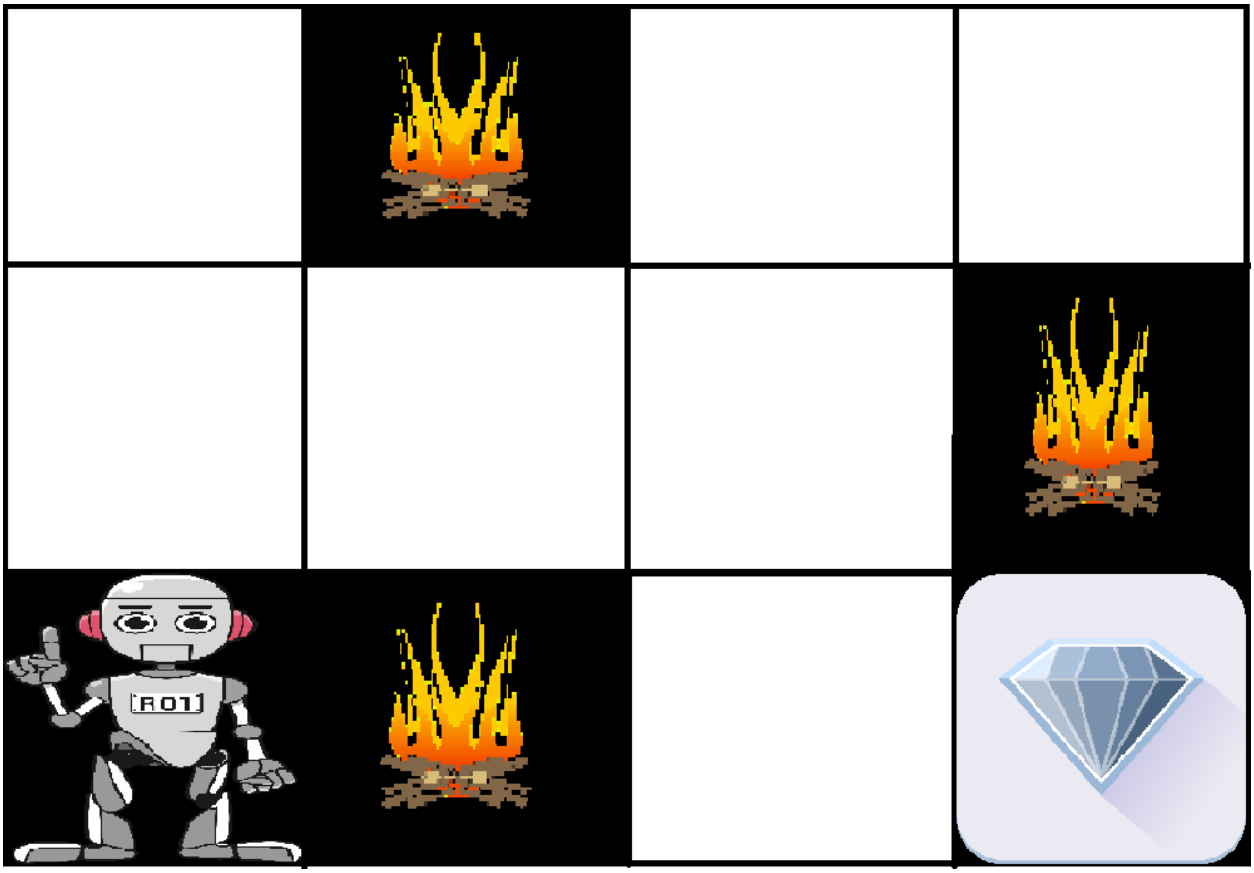
Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small

decisions without human guidance.

Reinforcement learning is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

Example:

The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.



The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired. The robot learns by trying all the possible paths and then choosing the path which gives

him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Difference between Reinforcement learning and Supervised learning:

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the	In Supervised learning, the decision is made on the initial input or the input given at the start

previous input	
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game,text summarization	Example: Object recognition,spam detetction

Types of Reinforcement:

There are two types of Reinforcement:

1. **Positive:** Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.
   Advantages of reinforcement learning are:
   - Maximizes Performance
   - Sustain Change for a long period of time
   - Too much Reinforcement can lead to an overload of states which can diminish the results
2. **Negative:** Negative Reinforcement is defined as strengthening of

behavior because a negative condition is stopped or avoided.

Advantages of reinforcement learning:

- Increases Behavior
- Provide defiance to a minimum standard of performance
- It Only provides enough to meet up the minimum behavior

Elements of Reinforcement Learning

Reinforcement learning elements are as follows:

1. Policy
2. Reward function
3. Value function
4. Model of the environment

Policy: Policy defines the learning agent behavior for given time period. It is a mapping from perceived states of the environment to actions to be taken when in those states.

Reward function: Reward function is used to define a goal in a reinforcement learning problem.A reward function is a function that provides a numerical score based on the state of the environment

Value function: Value functions specify what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

Model of the environment: Models are used for planning.

Credit assignment problem: Reinforcement learning algorithms learn to generate an internal value for the intermediate states as to how good they are in leading to the goal. The learning decision maker is called the agent. The agent interacts with the environment that includes everything outside the agent.

The agent has sensors to decide on its state in the environment and takes action that modifies its state.

The reinforcement learning problem model is an agent continuously interacting with an environment. The agent and the environment interact in a sequence of time steps. At each time step t, the agent receives the state of the environment and a scalar numerical reward for the previous action, and then the agent then selects an action.

Reinforcement learning is a technique for solving Markov decision problems.

Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem.

Various Practical Applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.

- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

Application of Reinforcement Learnings

1. Robotics: Robots with pre-programmed behavior are useful in structured environments, such as the assembly line of an automobile manufacturing plant, where the task is repetitive in nature.

2. A master chess player makes a move. The choice is informed both by planning, anticipating possible replies and counter replies.

3. An adaptive controller adjusts parameters of a petroleum refinery's operation in real time.

RL can be used in large environments in the following situations:

1. A model of the environment is known, but an analytic solution is not available;
2. Only a simulation model of the environment is given (the subject of simulation-based optimization)
3. The only way to collect information about the environment is to interact with it.

Advantages and Disadvantages of Reinforcement Learning

Advantages of Reinforcement learning

1. Reinforcement learning can be used to solve very complex problems that

cannot be solved by conventional techniques.

2. The model can correct the errors that occurred during the training process.

3. In RL, training data is obtained via the direct interaction of the agent with the environment

4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.

5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.

6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Disadvantages of Reinforcement learning

1. Reinforcement learning is not preferable to use for solving simple problems.

2. Reinforcement learning needs a lot of data and a lot of computation

3. Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.

4. Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

Implementation:

Python3

```
import gym

import numpy as np




# Define the Q-table and learning rate

q_table = np.zeros((state_size, action_size))

alpha = 0.8

gamma = 0.95
```

```
# Train the Q-Learning algorithm

for episode in range(num_episodes):

    state = env.reset()

    done = False

    while not done:

        # Choose an action

        action = np.argmax(

            q_table[state, :] + np.random.randn(1, action_size) *
```

```
(1. / (episode + 1)))




        # Take the action and observe the new state and reward

        next_state, reward, done, _ = env.step(action)




        # Update the Q-table

        q_table[state, action] = (1 - alpha) * q_table[state,
action] + \

            alpha * (reward + gamma * np.max(q_table[next_state,
:]))
```

```
        state = next_state




# Test the trained Q-Learning algorithm

state = env.reset()

done = False

while not done:

    # Choose an action
```

```
    action = np.argmax(q_table[state, :])









    # Take the action


    state, reward, done, _ = env.step(action)


    env.render()
```

Machine Learning: Connectionist - Introduction, foundations,

Perceptron (check in detail)

Connectionism, a foundational concept in machine learning and artificial intelligence (AI), is a paradigm inspired by the structure and function of the human brain. Also known as neural network modeling or artificial neural networks (ANNs), connectionist systems are composed of interconnected processing units, or neurons, that work collaboratively to solve complex problems. This approach aims to mimic the brain's ability to learn from experience and generalize patterns.

Here's an overview of connectionism, including its introduction, foundations, and the Perceptron in AI:

Introduction to Connectionism:

Inspiration from Neuroscience: Connectionism draws inspiration from the biological neural networks present in the human brain. It attempts to emulate the brain's structure and functioning using computational models.

Learning from Data: Connectionist systems learn by adjusting the strengths of connections (weights) between neurons based on input data. This learning process allows the network to recognize patterns, make predictions, and perform various tasks.

Parallel Distributed Processing: Connectionist models often involve parallel processing of information across multiple interconnected nodes. This parallelism

enables efficient computation and robustness against failures in individual components.

Foundations of Connectionism:

Neural Network Architecture: Connectionist systems typically consist of layers of interconnected neurons. These layers can be divided into input, hidden, and output layers. Information flows from the input layer through the hidden layers to the output layer, with each neuron performing simple computations on its inputs.

Activation Functions: Neurons in a connectionist system apply activation functions to their inputs to produce output signals. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), and tanh (Hyperbolic Tangent). These functions introduce non-linearity into the network, enabling it to model complex relationships in data.

Learning Algorithms: Connectionist systems employ learning algorithms to adjust the weights of connections between neurons. Backpropagation is a widely used algorithm for training neural networks. It involves iteratively updating the network's weights to minimize the difference between the predicted outputs and the true targets.

Perceptron in AI:

The Perceptron is one of the simplest forms of neural networks, consisting of a single layer of neurons. It was introduced by Frank Rosenblatt in 1957 and marked an

important milestone in the development of artificial neural networks. Here's a detailed overview of the Perceptron:

Structure: The Perceptron consists of an input layer, which receives input signals, and an output layer, which produces output signals. Each input is associated with a weight, and the Perceptron computes a weighted sum of its inputs. This sum is then passed through an activation function to produce the output.

Activation Function: The Perceptron traditionally uses a step function as its activation function. It produces a binary output (0 or 1) based on whether the weighted sum of inputs exceeds a certain threshold. This simple activation function allows the Perceptron to make binary decisions.

Learning Rule: The Perceptron learning rule is based on the concept of supervised learning. It adjusts the weights of connections between neurons in response to training data. The goal is to minimize the error between the Perceptron's output and the desired output for each input pattern. The learning rule iteratively updates the weights until the network achieves satisfactory performance on the training data.

Limitations: While the Perceptron is effective for linearly separable problems, it has limitations in handling non-linearly separable data. This led to the development of more complex neural network architectures, such as multi-layer perceptrons (MLPs), capable of learning non-linear decision boundaries.

In summary, connectionism provides a powerful framework for building intelligent systems that can learn from data and generalize patterns. The Perceptron, as a fundamental building block of neural networks, exemplifies the principles of connectionism and has paved the way for more advanced AI techniques.

Machine learning: Social and emergent: Models, The Genetic Algorithm, Artificial Life and Social based Learning..

Social and emergent models in machine learning represent a class of algorithms inspired by social behavior, evolutionary principles, and emergent phenomena observed in natural systems. These models aim to mimic the dynamics of complex adaptive systems, where simple interactions between individual entities give rise to collective behaviors and patterns. Here, we'll delve into three key aspects of social and emergent models in machine learning: Genetic Algorithms, Artificial Life, and Social-based Learning.

1. Genetic Algorithms (GA):

Genetic Algorithms are optimization techniques inspired by the principles of natural selection and genetics. They are used to find approximate solutions to optimization and search problems. Here's a detailed overview:

- Representation: Solutions to a problem are represented as strings of symbols, often binary strings, called chromosomes or genotypes.
- Fitness Function: A fitness function evaluates the quality of each solution based on how well it performs on the given task. This function guides the evolutionary process by determining which solutions are more likely to survive and reproduce.
- Selection: Individuals with higher fitness values are selected with higher probability for reproduction. Various selection methods such as roulette wheel selection, tournament selection, or rank-based selection can be used.

- Crossover: Crossover involves combining genetic material from two parent solutions to produce offspring solutions. Different crossover techniques like single-point, two-point, or uniform crossover are employed.
- Mutation: Mutation introduces random changes in offspring solutions to maintain diversity in the population. It prevents the algorithm from getting stuck in local optima.
- Termination Criteria: The algorithm terminates when a stopping criterion is met, such as reaching a maximum number of generations or achieving a satisfactory solution.

2. Artificial Life (ALife):

Artificial Life is a multidisciplinary field that studies life-like processes in artificial systems. It focuses on synthesizing living-like behaviors and emergent properties in computational models. Key components of Artificial Life include:

- Agents: Individual entities in the artificial environment capable of sensing, acting, and interacting with the environment and other agents.
- Environment: The simulated world or space where agents reside and interact. The environment provides the context for emergent behaviors to arise.
- Emergence: Emergent phenomena refer to complex patterns or behaviors that emerge from interactions among simple components. These emergent properties are not explicitly programmed but arise spontaneously from the dynamics of the system.
- Evolutionary Dynamics: Evolutionary algorithms, such as Genetic Algorithms, are often employed in Artificial Life simulations to model the process of adaptation and evolution over generations.

3. Social-based Learning:

Social-based learning approaches leverage concepts from social psychology and sociology to design machine learning algorithms that incorporate social interactions and dynamics. Key aspects of social-based learning include:

- Learning from Peers: Agents learn from interactions with other agents in the system, exchanging knowledge, experiences, or strategies to improve their own performance.
- Social Influence: The behavior and decisions of agents are influenced by social factors such as norms, reputation, or the opinions of others.
- Collaboration and Competition: Agents may collaborate towards common goals or compete with each other for resources or rewards. This competition and cooperation drive the emergence of collective behaviors.
- Adaptive Behavior: Agents adapt their strategies or behaviors based on feedback from the environment and social interactions, leading to the emergence of adaptive and intelligent behaviors at the system level.

In summary, social and emergent models in machine learning provide powerful tools for solving complex optimization problems, synthesizing life-like behaviors in artificial systems, and designing algorithms that can learn and adapt in social environments. These approaches draw inspiration from natural systems and collective behavior to create intelligent and adaptive computational systems.

OR

Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. **They are commonly used to generate high-quality solutions for optimization problems and search problems.**

Genetic algorithms simulate the process of natural selection which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate “survival of the fittest” among individuals of consecutive generations to solve a problem. **Each generation consists of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

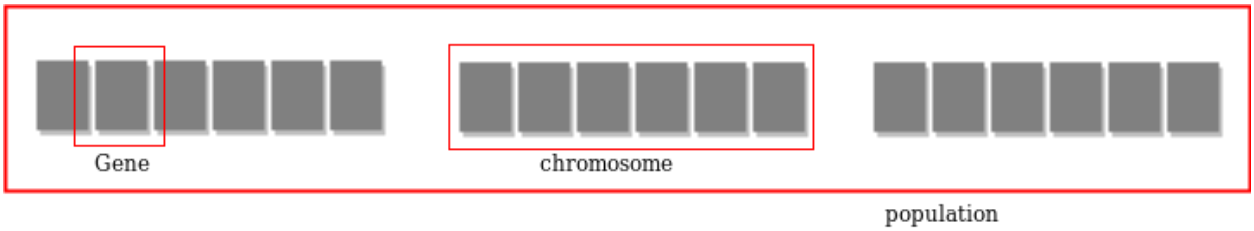
Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with the genetic structure and behavior of chromosomes of the population. Following is the foundation of GAs based on this analogy –

1. Individuals in the population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from the “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Search space

The population of individuals are maintained within search space. Each individual represents a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).



Fitness Score

A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.

The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores.The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce **better offspring** by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.

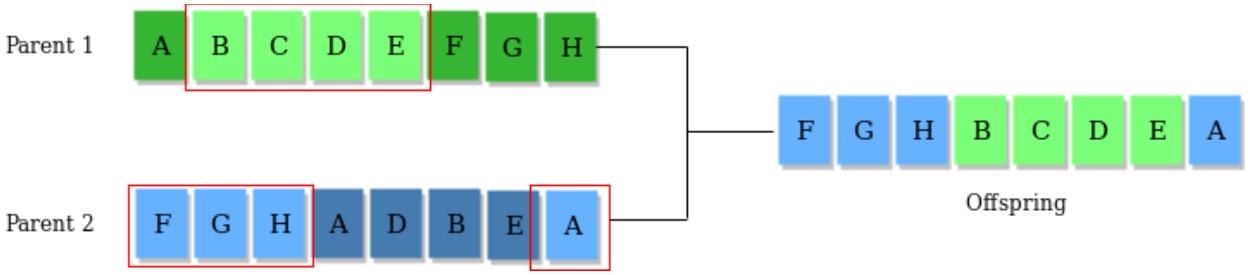
Each new generation has on average more “better genes” than the individual (solution) of previous generations. Thus each new generations have better **“partial solutions”** than previous generations. Once the offspring produced having no significant difference from offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

Operators of Genetic Algorithms

Once the initial generation is created, the algorithm evolves the generation using following operators –

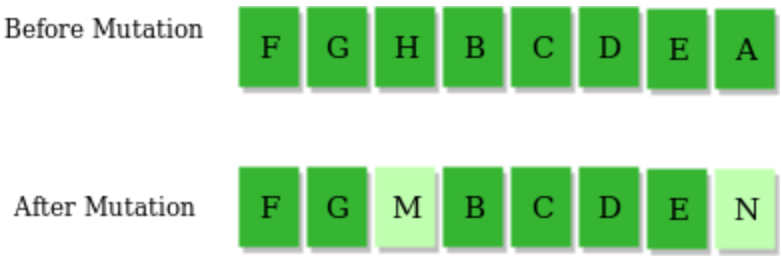
1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). For example –



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. For

example –



The whole algorithm can be summarized as –

```
1) Randomly initialize populations p

2) Determine fitness of population

3) Until convergence repeat:

     a) Select parents from population

     b) Crossover and generate new population

     c) Perform mutation on new population

     d) Calculate fitness for new population
```

Example problem and solution using Genetic Algorithms

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9, and other special symbols are considered as genes

- A string generated by these characters is considered as chromosome/solution/Individual

Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.