**ASSIGNMENT 06:**

**Implementing Knowledge Representation System (Semantic Network):**

**Introduction:**

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

This representation consist of mainly two types of relations:

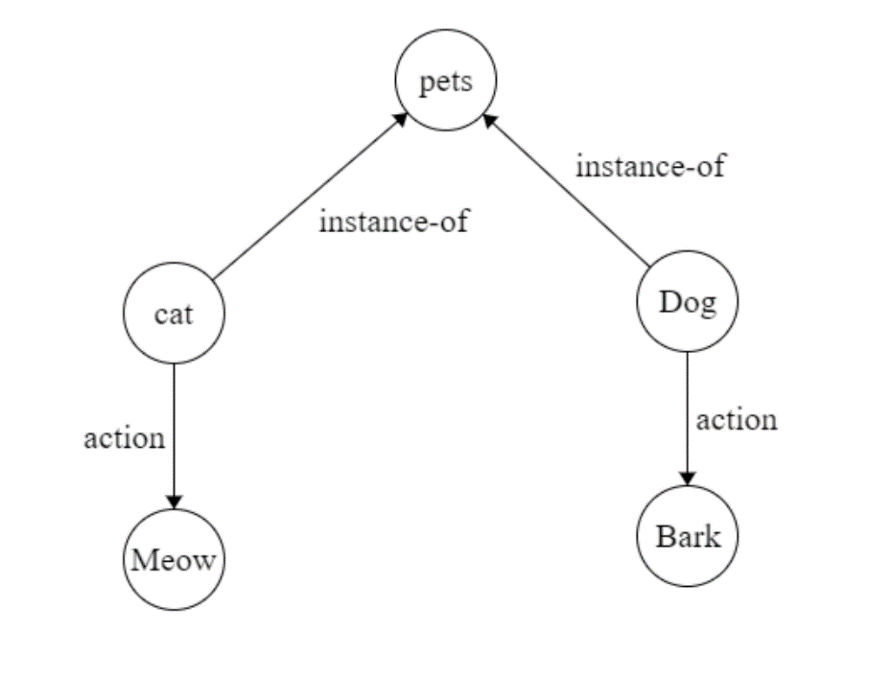
IS-A relation (Inheritance)

Kind-of-relation

**Problem Description:**

The task is to develop a simple Knowledge Representation System (KRS) that utilizes a semantic network paradigm. The system should allow users to define entities, attributes, and relationships between entities. The given facts are:

1. Cats meow.
2. Dogs bark.
3. Cats can be pets.
4. Dogs can be pets.

**Graph Representation:**

**Source Code:**

class SemanticNetwork:

    def \_\_init\_\_(self):

        self.nodes = {}

    def add\_relation(self, node1, node2):

        if node1 not in self.nodes:

            self.nodes[node1] = []

        if node2 not in self.nodes:

            self.nodes[node2] = []

        self.nodes[node1].append(node2)

        self.nodes[node2].append(node1)

    def print\_network(self):

        for node, neighbors in self.nodes.items():

            neighbors\_str = ' '.join(neighbors)

            print(f"{node}: {neighbors\_str}")

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

    network = SemanticNetwork()

    # Adding relations

    network.add\_relation("cat", "meow")

    network.add\_relation("dog", "bark")

    network.add\_relation("cat", "pet")

    network.add\_relation("dog", "pet")

    # Printing the semantic network

    network.print\_network()

**Output:**

cat: meow pet

meow: cat

dog: bark pet

bark: dog

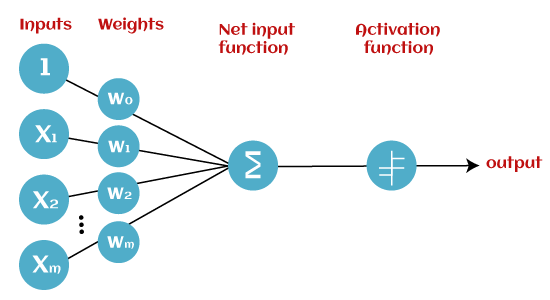
pet: cat dog

**ASSIGNMENT 07:**

**Implementing Knowledge Representation System (Semantic Network):**

**Introduction:**

Artificial neural network is non-linear, parallel, distributed, and highly connected network having capabilities of adaptability, self-organization, fault tolerance, and Very Large Scale Integration (VLSI) implementation, which closely resembles with physical nervous system.

Perceptron Learning algorithm was originally developed by Frank Rosenblatt in late 1950s. Training pattern are presented to the network’s input; the output is computed. Then the connection weights wj are modified by an amount that is proportional to the product of difference between the actual output, y, and the desired output,d ,and the input pattern,x.

**Algorithm:**

**Step-1:**

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

∑wi\*xi = x1\*w1 + x2\*w2 +…wn\*xn

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

**∑wi\*xi + b**

**Step-2:**

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

**Y = f(∑wi\*xi + b)**

**Problem Description:**

Implement a NAND logic function using the Perceptron model. The NAND logic function returns 1 only if both inputs are 0; otherwise, it returns 0. We'll use the Perceptron model, which consists of weights (w) and a bias (b), along with the unit step function to determine the output.

**Source Code:**

import numpy as np

# define Unit Step Function

def unitStep(v):

    if v >= 0:

        return 1

    else:

        return 0

# design Perceptron Model

def perceptronModel(x, w, b):

    v = np.dot(w, x) + b

    y = unitStep(v)

    return y

# NOT Logic Function

# wNOT = -1, bNOT = 0.5

def NOT\_logicFunction(x):

    wNOT = -1

    bNOT = 0.5

    return perceptronModel(x, wNOT, bNOT)

# AND Logic Function

# w1 = 1, w2 = 1, bAND = -1.5

def AND\_logicFunction(x):

    w = np.array([1, 1])

    bAND = -1.5

    return perceptronModel(x, w, bAND)

# NAND Logic Function with AND and NOT function calls in sequence

def NAND\_logicFunction(x):

    output\_AND = AND\_logicFunction(x)

    output\_NOT = NOT\_logicFunction(output\_AND)

    return output\_NOT

if \_\_name\_\_ == '\_\_main\_\_':

    x, y = input("Enter the input x, y: ").split(' ')

    x = int(x)

    y = int(y)

    print(f"NAND({x}, {y}) = {NAND\_logicFunction(np.array([x, y]))}")

**Output:**

Enter the input x, y: 0 0

NAND(0, 0) = 1

Enter the input x, y: 0 1

NAND(0, 1) = 1

Enter the input x, y: 1 0

NAND(1, 0) = 1

Enter the input x, y: 1 1

NAND(1, 1) = 0