NAME :- DIVYANG BAGLA
PANEL :- C PC-33
C-3

PAGE NO.:

AI LAB ASSIGNMENT-5

TITLE :- Write a program to develop mini expert system using prolog.

AIM :- Implementation of expert system.

REQUIREMENTS :- SWI Prolog, Turbo Prolog.

THEORY :-

(1) Architecture of expert system :-
An expert system is a computer program that is designed to solve complex problems and to provide decision making ability like human expert.

(2) Main players of expert system :-
There are 5 members of the expert system :-
→ Domain expert
→ Knowledge engineer
→ Programmer
→ Project Manager
→ End user.

INPUT :- Run program on SWI Prolog.

OUTPUT :- Gives decisions based on rules provided in program

PLATFORM :- Linux / Windows.

FAQ's :-

Q1) Write in brief forward chaining & backward chaining of inference engine.

→ FORWARD CHAINING :-

It is also known as forward deduction or forward reasoning method when using an inference engine.

Properties ⇒

1) It is a down up approach.
2) It is also called as data driven.

→ BACKWARD CHAINING :-

It is also known as Backward deduction or backward reasoning method when using an inference engine.

Properties ⇒

1) It is known as top down approach.
2) It is called as goal driven approach.

Q2) First down the applications of expert system.

→ Different types of medical diagnosis like viral medicine etc.
→ Forecasting top damage
→ Diagnosis of software development project.
→ Teaching students specialize task.

AI - Lab 5 Code

```
import numpy as np
class NeuralNetwork():

    def __init__(self):
        # seeding for random number generation
        np.random.seed(1)

        #converting weights to a 3 by 1 matrix with values from -1 to 1 and mean of
0
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1

    def sigmoid(self, x):
        #applying the sigmoid function
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        #computing derivative to the Sigmoid function
        return x * (1 - x)

    def train(self, training_inputs, training_outputs, training_iterations):

        #training the model to make accurate predictions while adjusting weights
continually
        for iteration in range(training_iterations):
            #siphon the training data via  the neuron
            output = self.think(training_inputs)

            #computing error rate for back-propagation
            error = training_outputs - output

            #performing weight adjustments
            adjustments = np.dot(training_inputs.T, error *
self.sigmoid_derivative(output))

            self.synaptic_weights += adjustments

    def think(self, inputs):
        #passing the inputs via the neuron to get output
        #converting values to floats

        inputs = inputs.astype(float)
        output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
        return output


'''
```

OUTPUT:-

```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training:
[[10.08740896]
 [-0.20695366]
 [-4.83757835]]
User Input One: 0
User Input Two: 1
User Input Three: 1
Considering New Situation:  0 1 1
New Output data:
[0.00640321]
Wow, we did it!
'''
```