

Assignment No. 1**Aim:**

Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relation by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations.

Objectives:

1. To learn different fuzzy operation on fuzzyset.
2. To learn min-max composition on fuzzyset.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth- truth values between "completely true" and "completely false". As its name suggests, it is the logic underlying modes of reasoning which are approximate rather than exact. The importance of fuzzy logic derives from the fact that most modes of human reasoning and especially common sense reasoning are approximate innature.

The essential characteristics of fuzzy logic as founded by ZaderLotfi are as follows:

- In fuzzy logic, exact reasoning is viewed as a limiting case of approximatereasoning
- In fuzzy logic everything is a matter ofdegree
- Any logical system can befuzzified
- In fuzzy logic, knowledge is interpreted as a collection of elastic or, equivalently , fuzzy constraint on a collection ofvariables
- Inference is viewed as a process of propagation of elasticconstraints

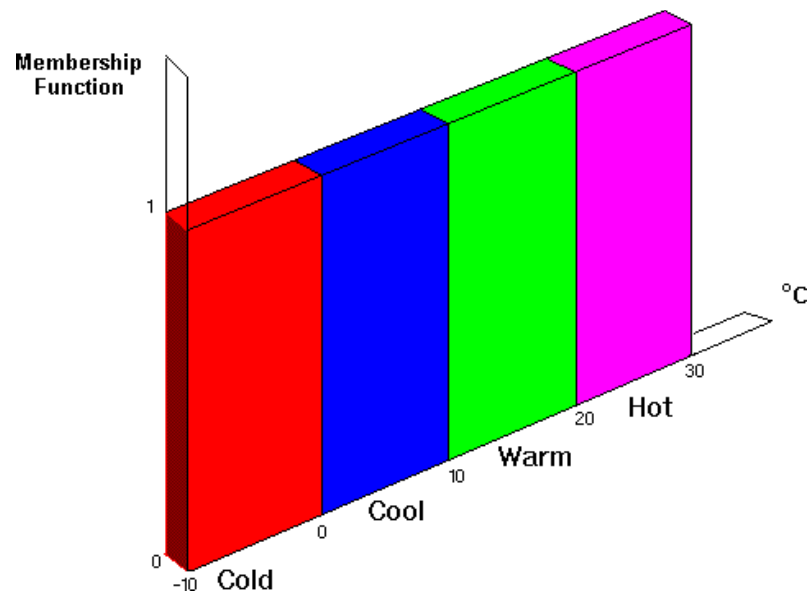
The third statement hence, defines Boolean logic as a subset of Fuzzylogic.

Fuzzy Sets

Fuzzy Set Theory was formalised by Professor LoftiZadeh at the University of California in 1965. What Zadeh proposed is very much a paradigm shift that first gained acceptance in the Far East and its successful application has ensured its adoption around theworld.

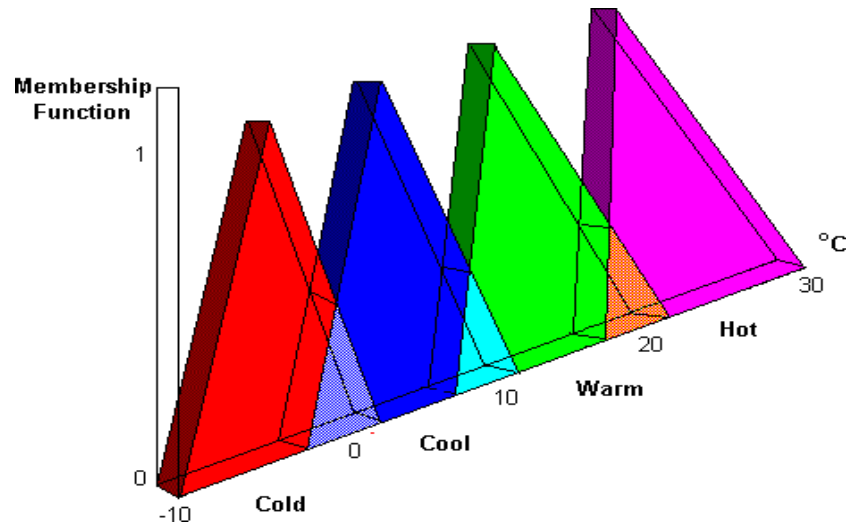
A paradigm is a set of rules and regulations which defines boundaries and tells us what to do to be successful in solving problems within these boundaries. For example the use of transistors instead of vacuum tubes is a paradigm shift - likewise the development of Fuzzy Set Theory from

Bivalent Set Theory can be somewhat limiting if we wish to describe a 'humanistic' problem mathematically. For example, Fig 1 below illustrates bivalent sets to characterise the temperature of a room.



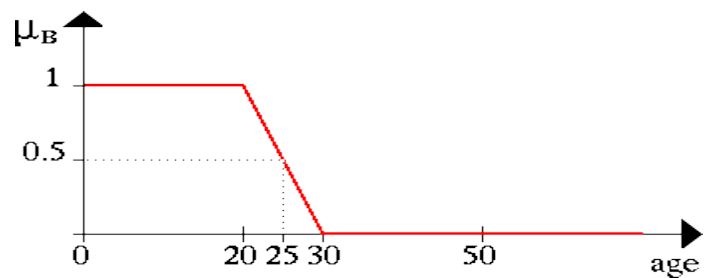
The most obvious limiting feature of bivalent sets that can be seen clearly from the diagram is that they are mutually exclusive - it is not possible to have membership of more than one set (opinion would widely vary as to whether 50 degrees Fahrenheit is 'cold' or 'cool' hence the expert knowledge we need to define our system is mathematically at odds with the humanistic world). Clearly, it is not accurate to define a transition from a quantity such as 'warm' to 'hot' by the application of one degree Fahrenheit of heat. In the real world a smooth (unnoticeable) drift from warm to hot would occur. This natural phenomenon can be described more accurately by Fuzzy Set Theory. Fig.2 below shows how fuzzy sets quantifying the same information can describe this natural drift.

The whole concept can be illustrated with this example. Let's talk about people and "youthness". In this case the set S (the universe of discourse) is the set of people. A fuzzy subset YOUNG is also defined, which answers the question "to what degree is person x young?" To each person in the universe of discourse, we have to assign a degree of membership in the fuzzy subset YOUNG. The easiest way to do this is with a membership function based on the person's age.



$\text{young}(x) = \{ 1, \text{ if } \text{age}(x) \leq 20, \\ (30 - \text{age}(x))/10, \text{ if } 20 < \text{age}(x) \leq 30, \\ 0, \text{ if } \text{age}(x) > 30 \}$

A graph of this looks like:



Given this definition, here are some example values:

Person	Age	Degree of Youth
John	10	1.00
Edwin	21	0.90
Parthiban	25	0.50
Arosha	26	0.40
Chin Wei	28	0.20
Rajkumar	83	0.00

So given this definition, we would say that the degree of truth of the statement "Parthiban is YOUNG" is 0.50.

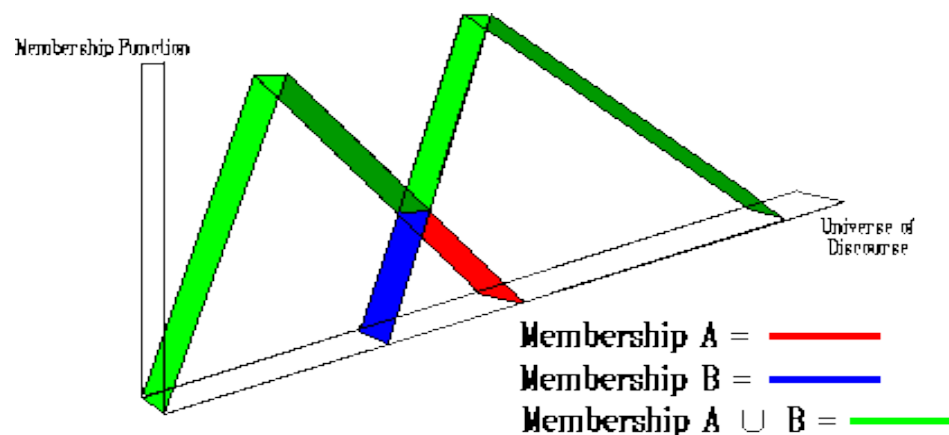
Note: Membership functions almost never have as simple a shape as $\text{age}(x)$. They will at least tend to be triangles pointing up, and they can be much more complex than that. Furthermore, membership functions so far is discussed as if they always are based on a single criterion, but this isn't always the case, although it is the most common case. One could, for example, want to have the membership function for YOUNG depend on both a person's age and their height (Arosha's short for his age). This is perfectly legitimate, and occasionally used in practice. It's referred to as a two-dimensional membership function. It's also possible to have even more criteria, or to have the membership function depend on elements from two completely different universes of discourse.

Fuzzy Set Operations.

Union

The membership function of the Union of two fuzzy sets A and B with membership functions μ_A and μ_B respectively is defined as the maximum of the two individual membership functions. This is called the *maximum* criterion.

$$\mu_{A \cup B} = \max(\mu_A, \mu_B)$$



The Union operation in Fuzzy set theory is the equivalent of the **OR** operation in Boolean algebra.

Intersection

The membership function of the Intersection of two fuzzy sets A and B with membership functions μ_A and μ_B respectively is defined as the minimum of the two individual membership functions. This is called the *minimum* criterion.

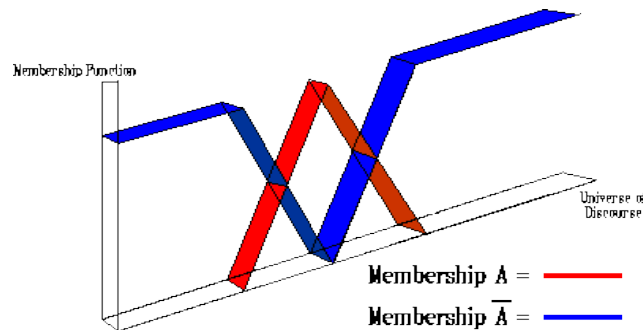
$$\mu_{A \cap B} = \min(\mu_A, \mu_B)$$

The Intersection operation in Fuzzy set theory is the equivalent of the **AND** operation in Boolean algebra.

Complement

The membership function of the Complement of a Fuzzy set A with membership function μ_A is defined as the negation of the specified membership function. This is called the *negation* criterion.

$$\mu_{\bar{A}} = 1 - \mu_A$$



The Complement operation in Fuzzy set theory is the equivalent of the NOT operation in Boolean algebra.

The following rules which are common in classical set theory also apply to Fuzzy set theory.

De Morgans law

$$\overline{(A \cap B)} = \bar{A} \cap \bar{B} \quad \overline{(A \cup B)} = \bar{A} \cap \bar{B}$$

Associativity

$$(A \cap B) \cap C = A \cap (B \cap C)$$

$$(A \cup B) \cup C = A \cup (B \cup C)$$

Commutativity

$$A \cap B = B \cap A, \quad A \cup B = B \cup A$$

Distributivity

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

Let A_1, A_2, \dots, A_n be fuzzy sets in U_1, U_2, \dots, U_n , respectively. The Cartesian product of A_1, A_2, \dots, A_n is a fuzzy set in the space $U_1 \times U_2 \times \dots \times U_n$ with the membership function as: $\mu_{A_1 \times A_2 \times \dots \times A_n}(x_1, x_2, \dots, x_n) = \min [\mu_{A_1}(x_1), \mu_{A_2}(x_2), \dots, \mu_{A_n}(x_n)]$

So, the Cartesian product of A_1, A_2, \dots, A_n are denoted by $A_1 \times A_2 \times \dots \times A_n$.

Cartesian Product: Example Let $A = \{(3, 0.5), (5, 1), (7, 0.6)\}$ Let $B = \{(3, 1), (5, 0.6)\}$

The product is all set of pairs from A and B with the minimum associated memberships $A \times B =$

$\{[(3, 3), \min(0.5, 1)], [(5, 3), \min(1, 1)], [(7, 3), \min(0.6, 1)], [(3, 5), \min(0.5, 0.6)], [(5, 5), \min(1, 0.6)], [(7, 5), \min(0.6, 0.6)]\} = \{[(3, 3), 0.5], [(5, 3), 1], [(7, 3), 0.6], [(3, 5), 0.5], [(5, 5), 0.6], [(7, 5), 0.6]\}$.

Conclusion

Thus we learnt implementation of Union, Intersection, Complement and Difference operations on fuzzy sets. Also created fuzzy relation by Cartesian product of any two fuzzy sets.

Code:

```
import numpy as np
D=[]
A = []
m = int(input('Enter how many elements you want in set A : '))
for i in range(0, m):
    x = float(input('Enter the numbers into the set A: '))
    A.append(x)
print(A)

B = []
n = int(input('Enter how many elements you want in set B: '))
for i in range(0, n):
    x = float(input('Enter the numbers into the set B: '))
    B.append(x)
print(B)

C = []
q = int(input('Enter how many elements you want in C set: '))
for i in range(0, q):
    x = float(input('Enter the numbers into the Set C: '))
    C.append(x)
print(C)

matAB=[[0 for j in range(n)] for i in range(m)]
matBC=[[0 for j in range(q)] for i in range(n)]

def unionAB():
    for i in range(m):
        D.append(max(A[i],B[i]))
    print("Union of the two set is:",D)

def interAB():
    for i in range(n):
        D.append(min(A[i],B[i]))
    print("intersection of the two set is:",D)

def compa():
    for i in range(m):
        D.append(1-A[i])
    print("Complement of A is:",D)

def compb()
```

```

for i in range(n):
    D.append(1-B[i])
print("Complement of B is:",D)

```

```

def compc():
    for i in range(q):
        D.append(1-C[i])
    print("Complement of C is:",D)

```

```

def proAB():
    for i in range(n):
        D.append(A[i]*B[i])
    print("Product of A,B is:",D)

```

```

def proBC():
    for i in range(m):
        D.append(B[i]*C[i])
    print("Product of B,C is:",D)

```

```

def cartAB():
    for i in range(m):
        for j in range(n):
            matAB[i][j]=min(A[i],B[j])
    print(matAB)

```

```

def cartBC():
    for i in range(n):
        for j in range(q):
            matBC[i][j]=min(B[i],C[j])
    print(matBC)

```

```

def default():
    print("Invalid option")

```

```

ch='no'
while (ch!='yes'):
    switcher = {
        1: unionAB,
        2: interAB,
        3: compa,
        4: compb,

```

```

5: compc,
6: proAB,
7: proBC,
8: cartAB,
9: cartBC,
10: exit
    }

```

```

defswitch(option):
returnswitcher.get(option, default)()
xin=input("1: unionAB\n2: interAB\n3: compa\n4: compb\n5: compc\n6: proAB\n7: proBC\n8:
cartAB\n9: cartBC\n10:Exit\n\nYour Choice: ")
    x=int(xin)
print(switch(x))

```

Output:

```

E:\CLG\SEM      8\SCOA>"C:/Program      Files/Python37/python.exe"      "e:/CLG/SEM
8/SCOA/fuzzysset.py"
Enter how many elements you want in set A : 5
Enter the numbers into the set A: 2
Enter the numbers into the set A: 3
Enter the numbers into the set A: 1
Enter the numbers into the set A: 4
Enter the numbers into the set A: 5
[2.0, 3.0, 1.0, 4.0, 5.0]
Enter how many elements you want in set B: 5
Enter the numbers into the set B: 4
Enter the numbers into the set B: 2
Enter the numbers into the set B: 6
Enter the numbers into the set B: 8
Enter the numbers into the set B: 7
[4.0, 2.0, 6.0, 8.0, 7.0]
Enter how many elements you want in C set: 5
Enter the numbers into the Set C: 3
Enter the numbers into the Set C: 6
Enter the numbers into the Set C: 4
Enter the numbers into the Set C: 7
Enter the numbers into the Set C: 8
[3.0, 6.0, 4.0, 7.0, 8.0]
1: unionAB

```

2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB

9: cartBC
10:Exit

Your Choice: 1

Union of the two set is: [4.0, 3.0, 6.0, 8.0, 7.0]

None

1: unionAB
2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 2

intersection of the two set is: [4.0, 3.0, 6.0, 8.0, 7.0, 2.0, 2.0, 1.0, 4.0, 5.0]

None

1: unionAB
2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 3

Complement of A is: [4.0, 3.0, 6.0, 8.0, 7.0, 2.0, 2.0, 1.0, 4.0, 5.0, -1.0, -2.0, 0.0, -3.0, -4.0]

None

1: unionAB
2: interAB
3: compa

4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 4

Complement of B is: [4.0, 3.0, 6.0, 8.0, 7.0, 2.0, 2.0, 1.0, 4.0, 5.0, -1.0, -2.0, 0.0, -3.0, -4.0, -3.0, -1.0, -5.0, -7.0, -6.0]

None

1: unionAB
2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 5

Complement of C is: [4.0, 3.0, 6.0, 8.0, 7.0, 2.0, 2.0, 1.0, 4.0, 5.0, -1.0, -2.0, 0.0, -3.0, -4.0, -3.0, -1.0, -5.0, -7.0, -6.0, -2.0, -5.0, -3.0, -6.0, -7.0]

None

1: unionAB
2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 6

Product of A,B is: [4.0, 3.0, 6.0, 8.0, 7.0, 2.0, 2.0, 1.0, 4.0, 5.0, -1.0, -2.0, 0.0, -3.0, -4.0, -3.0, -1.0, -5.0, -7.0, -6.0, -2.0, -5.0, -3.0, -6.0, -7.0, 8.0, 6.0, 6.0, 32.0, 35.0]

None

1: unionAB
2: interAB
3: compa
4: compb

5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 7

Product of B,C is: [4.0, 3.0, 6.0, 8.0, 7.0, 2.0, 2.0, 1.0, 4.0, 5.0, -1.0, -2.0, 0.0, -3.0, -4.0, -3.0, -1.0, -5.0, -7.0, -6.0, -2.0, -5.0, -3.0, -6.0, -7.0, 8.0, 6.0, 6.0, 32.0, 35.0, 12.0, 12.0, 24.0, 56.0, 56.0]

None

1: unionAB
2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 8

[[2.0, 2.0, 2.0, 2.0, 2.0], [3.0, 2.0, 3.0, 3.0, 3.0], [1.0, 1.0, 1.0, 1.0, 1.0], [4.0, 2.0, 4.0, 4.0, 4.0], [4.0, 2.0, 5.0, 5.0, 5.0]]

None

1: unionAB
2: interAB
3: compa
4: compb
5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 9

[[3.0, 4.0, 4.0, 4.0, 4.0], [2.0, 2.0, 2.0, 2.0, 2.0], [3.0, 6.0, 4.0, 6.0, 6.0], [3.0, 6.0, 4.0, 7.0, 8.0], [3.0, 6.0, 4.0, 7.0, 7.0]]

None

1: unionAB
2: interAB
3: compa
4: compb

5: compc
6: proAB
7: proBC
8: cartAB
9: cartBC
10:Exit

Your Choice: 10

Assignment No. 2

Aim:

Implement genetic algorithm for benchmark function (eg. Square, Rosenbrock function etc). Initialize the population from the Standard Normal Distribution. Evaluate the fitness of all its individuals. Then you will do multiple generation of a genetic algorithm. A generation consists of applying selection, crossover, mutation, and replacement.

Use:

- Tournament selection without replacement with tournament sizes
- One point crossover with probability P_c
- bit-flip mutation with probability P_m
- Use full replacement strategy

Objectives:

1. To learn Genetic Algorithm.
2. To learn about optimization algorithm.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Genetic Algorithms are a class of stochastic, population based optimization algorithms inspired by the biological evolution process using the concepts of “Natural Selection” and “Genetic Inheritance” (Darwin 1859) and originally developed by Holland.

GAs are now used in engineering and business optimization applications, where the search space is large and/or too complex (non-smooth) for analytic treatment. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found. This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

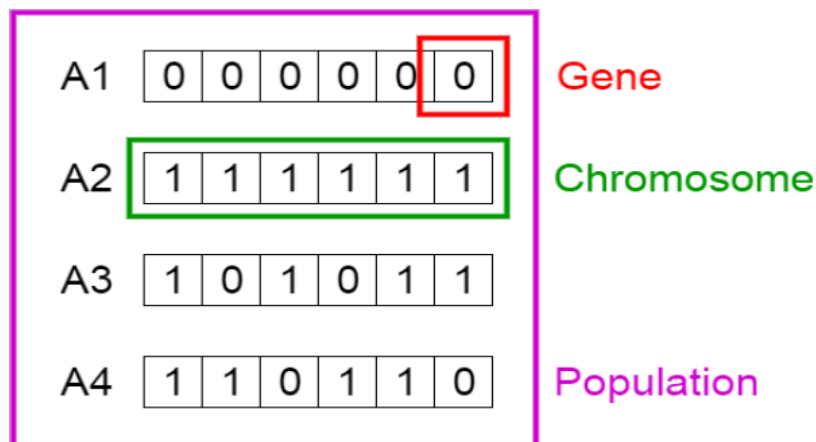
Five phases are considered in a genetic algorithm.

1. Initialpopulation
2. Fitnessfunction
- 3.Selection
- 4.Crossover
- 5.Mutation

Initial Population

The process begins with a set of individuals which is called a **Population**. Each individual is a solution to the problem you want to solve. An individual is characterized by a set of parameters (variables) known as **Genes**. Genes are joined into a string to form a **Chromosome** (solution).

In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.



Fitness Function

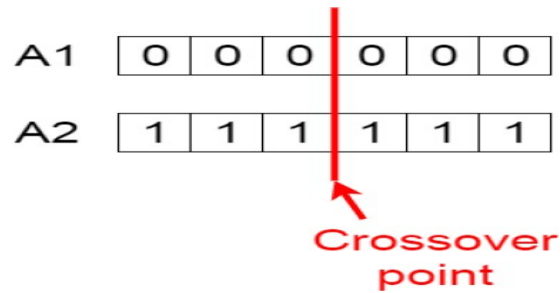
The **fitness function** determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a **fitness score** to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Selection

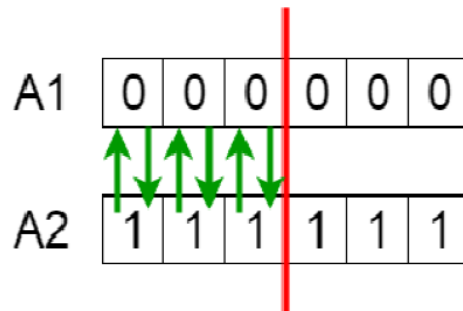
The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation. Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

Crossover

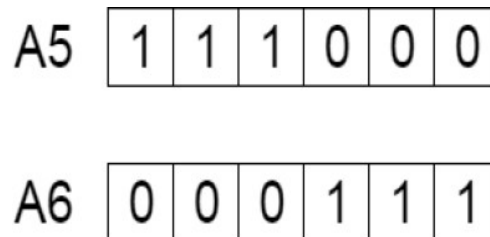
Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a **crossover point** is chosen at random from within the genes. For example, consider the crossover point to be 3 as shown below.



Offsprings are created by exchanging the genes of parents among themselves until the crossover point is reached.



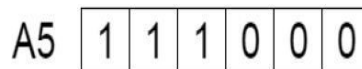
The new offspring are added to the population.



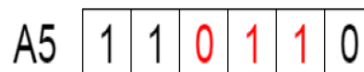
Mutation

In certain new offspring formed, some of their random probability. This implies that some of the bits in the bit string can be flipped.

Before Mutation



After Mutation

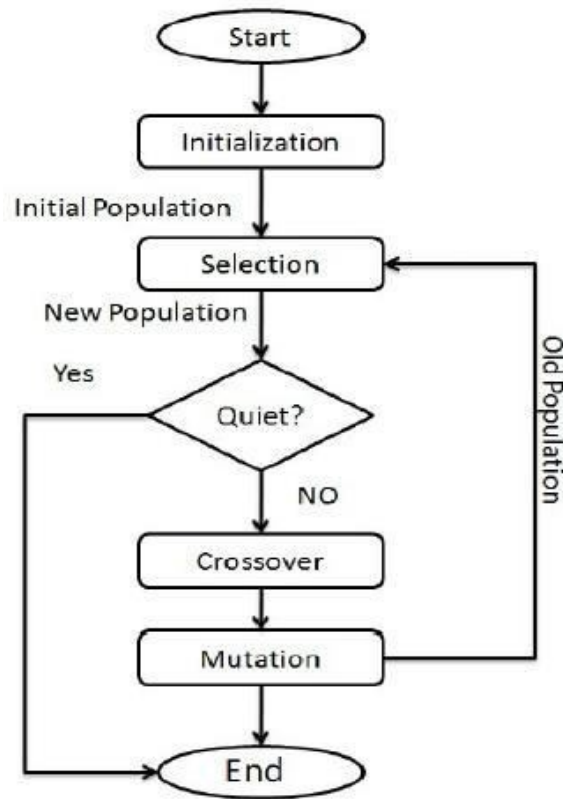


Mutation occurs to maintain diversity within the population and prevent premature convergence.

Termination

The algorithm terminates if the population has converged (does not produce offspring which are

significantly different from the previous generation). Then it is said that the genetic algorithm has provided set of solutions to our problems.



Comments

The population has a fixed size. As new generations are formed, individuals with least fitness die, providing space for new offspring. The sequence of phases is repeated to produce individuals in each new generation which are better than the previous generation.

GA Software: Python, Weka ,Mendal Soft ,Matlab

Conclusion

Thus we learnt implementation of genetic algorithm for benchmark function.

Code:**FitnessCalc.py**

```
class FitnessCalc():

    Solution = bytearray(64)

    @staticmethod
    def set_solution(solution_passed):
        for i in range(len(solution_passed)):
            FitnessCalc.Solution[i] = int(solution_passed[i])

    @staticmethod
    def get_max_fitness():
        return len(FitnessCalc.Solution)
```

Algorithm.py

```
from Population import Population
from Individual import Individual
from random import random, randint
```

```
class Algorithm():
```

#Constants

```
Uniform_rate = 0.5
Mutation_rate = 0.015
Tournament_size = 5
Elitism = True
```

@staticmethod

```
def evolve_population(population_passed):
    print("Evolving population...")
    new_population = Population(population_passed.size(), False)
    if Algorithm.Elitism:
        new_population.individuals.append(population_passed.get_fittest())
        elitism_off_set = 1
    else:
        elitism_off_set = 0

    #Do crossover over the entire population
    for i in range(elitism_off_set, population_passed.size()):
```

```

individual1 =
Algorithm.tournament_selection(population_passed)
    individual2 = Algorithm.tournament_selection(population_passed)
    new_individual = Algorithm.crossover(individual1, individual2)
    new_population.individuals.append(new_individual)

    #Do mutation randomly
    for i in range(elitism_off_set, population_passed.size()):
        Algorithm.mutate(new_population.get_individual(i))

    return new_population

@staticmethod
def crossover(individual1_passed, individual2_passed):
    new_sol = Individual()
    for i in range(individual1_passed.size()):
        if random() <= Algorithm.Uniform_rate:
            new_sol.set_gene(i, individual1_passed.get_gene(i))
        else:
            new_sol.set_gene(i, individual2_passed.get_gene(i))

    return new_sol

@staticmethod
def mutate(individual_passed):
    for i in range(individual_passed.size()):
        if random() <= Algorithm.Mutation_rate:
            gene = randint(0,1)
            individual_passed.set_gene(i, gene)

@staticmethod
def tournament_selection(population_passed):
    #Tournament pool
    tournament = Population(Algorithm.Tournament_size, False)

    """ Tournament selection technique.
    How it works: The algorithm choose randomly five
    individuals from the population and returns the fittest one """
    for i in range(Algorithm.Tournament_size):
        random_id = int(random() * population_passed.size())
        tournament.individuals.append(population_passed.get_individual(random_id))

```

```
fittest = tournament.get_fittest()
```

```
return fittest
```

Individual.py

```
from random import randint
from FitnessCalc import FitnessCalc
```

```
class Individual():
```

```
    DefaultGeneLength = len(FitnessCalc.Solution)
```

```
    def __init__(self):
        self.genes = bytearray(Individual.DefaultGeneLength)
        for i in range(Individual.DefaultGeneLength):
            gene = randint(0,1)
            self.genes[i] = gene
```

```
    def get_gene(self, index):
        return self.genes[index]
```

```
    def set_gene(self, index, what_to_set):
        self.genes[index] = what_to_set
```

```
    def size(self):
        return len(self.genes)
```

Population.py

```
from Individual import Individual
from FitnessCalc import FitnessCalc
```

```
class Population():
```

```
    def __init__(self, population_size, initialise):
        self.individuals = []
```

```
    #Creates the individuals
```

```
    if (initialise):
        for i in range(population_size):
            new_individual = Individual()
            self.individuals.append(new_individual)
```



```

generation_count += 1
print("Generation : %s Fittest : %s " % (generation_count, my_pop.fitness_of_the_fittest()))
my_pop = Algorithm.evolve_population(my_pop)
print("*****")

genes_the_fittest = []
for i in range(len(FitnessCalc.Solution)):
    genes_the_fittest.append(my_pop.get_fittest().genes[i])

print("Solution found !\nGeneration : %s Fittest : %s " % (generation_count + 1,
    my_pop.fitness_of_the_fittest()))
print("Genes of the Fittest : %s " % (genes_the_fittest))

finish = time()
print ("Time elapsed : %s " % (finish - start))

```

Output:

```

E:\CLG\SEM 8\SCOA\Outputs\Assignment 2>"C:/Program Files/Python37/python.exe"
"e:/CLG/SEM 8/SCOA/Outputs/Assignment 2/GA.py"

```

Generation : 1 Fittest : 42

Evolving population...

Generation : 2 Fittest : 44

Evolving population...

Generation : 3 Fittest : 47

Evolving population...

Generation : 4 Fittest : 48

Evolving population...

Generation : 5 Fittest : 51

Evolving population...

Generation : 6 Fittest : 55

Evolving population...

Generation : 7 Fittest : 55

Evolving population...

Evolving population...

Evolving population

Evolving population

Evolving population

Evolving population...

Evolving population

Evolving population

Evolving population

Evolving population

Evolving population

Evolving population

Evolving population

Generation : 21 Fittest : 64

Genes of the Eittest : [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[illegible]

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]$$

Timeelapsed:1.246286153793335

Assignment No. 3

Aim:

Implement Particle swarm optimization for benchmark function (eg. Square, Rosenbrock function). Initialize the population from the Standard Normal Distribution. Evaluate fitness of all particles.

Use:

- $c1=c2 = 2$
- Inertia weight is linearly varied between 0.9 to 0.4.
- Global best variation

Objectives:

1. To learn swarm algorithm
2. To learn about optimization algorithm.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

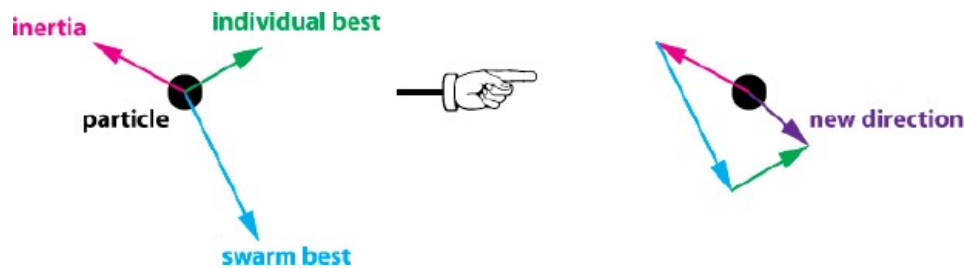
Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Compared to GA, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas where GA can be applied.

PSO uses a bunch of particles called *the swarm*. These particles are allowed to move around & explore the search-space.

These particles move in a direction which is guided by —

1. The particle's own previous velocity (*Inertia*)
2. Distance from the individual particles' best known position (*CognitiveForce*)
3. Distance from the swarms best known position (*SocialForce*)



Essentially the particles collectively communicate with each other to converge faster. The swarm doesn't fully explore the search space but potentially finds a better solution. Interestingly the overall direction of the swarm movement can be changed at any point of time when a particle's individual best is better than the swarm best. This allows a lot of *disorder* and more chances of getting close to the global minima of the cost function.

Algorithm

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest. After finding the two best values, the particle updates its velocity and positions with following equation (a) and (b).

$$v[] = v[] + c1 * \text{rand}() * (\text{pbest}[] - \text{present}[]) + c2 * \text{rand}() * (\text{gbest}[] - \text{present}[]) \dots \dots \dots (a)$$

$$\text{present}[] = \text{present}[] + v[] \dots \dots \dots (b)$$

$v[]$ is the particle velocity, $\text{present}[]$ is the current particle (solution). $\text{pbest}[]$ and $\text{gbest}[]$ are defined as stated before. $\text{rand}()$ is a random number between (0,1). $c1, c2$ are learning factors.

Usually

$$c1 = c2 = 2.$$

The pseudo code of the procedure is as follows:

For each particle

Initialize particle

END

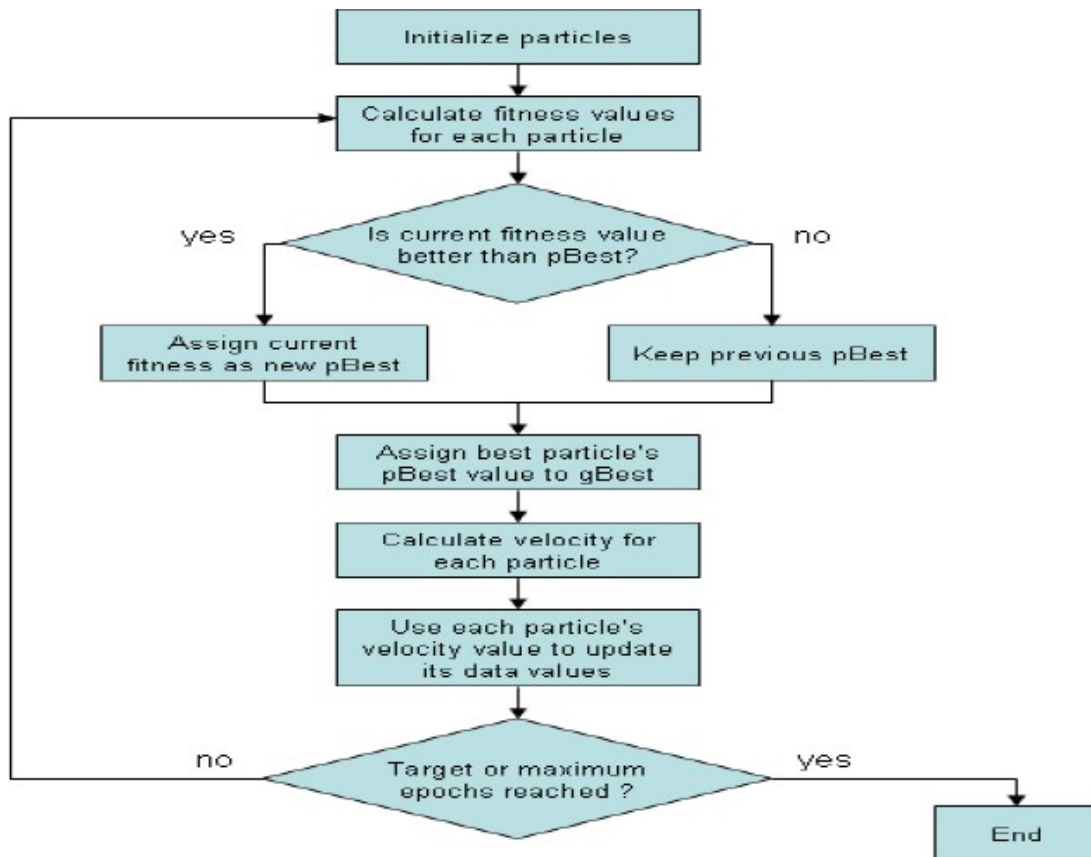
Do

For each particle

Calculate fitness value

If the fitness value is better than the best fitness value (pBest) in history set current value as the new pBest

End



Choose the particle with the best fitness value of all the particles as the gBest
For each particle

Calculate particle velocity according equation (a)

Update particle position according equation (b)

End

While maximum iterations or minimum error criteria is not attained

Particles' velocities on each dimension are clamped to a maximum velocity V_{max} . If the sum of accelerations would cause the velocity on that dimension to exceed V_{max} , which is a parameter specified by the user. Then the velocity on that dimension is limited to V_{max} .

Conclusion

Thus we learnt implementation of particle swarm optimization for benchmark function.

Assignment No. 4

Aim:

Implement basic logic gates using Mc-Culloch-Pitts or Hebbnet neural networks

Objectives:

1. To learn neuralnetwork.
2. To learn Mc-Culloch-Pitts or Hebbnet neuralnetworks

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Brain is the basic of human body which corresponds for all the functions. Neurons are responsible for the response of our body. Like the same way, artificial neurons are created which function as similar to that of biological brain. In this paper the response of the artificial neurons are obtained by using different threshold values and activation functions of logic gates. In this paper McCulloch-Pitts model is applied for the purpose of realization of logic gates.

First artificial neurons: The McCulloch-Pitts model

The McCulloch-Pitts model was an extremely simple artificial neuron. The inputs could be either a zero or a one. And the output was a zero or a one. And each input could be either excitatory or inhibitory. Now the whole point was to sum the inputs. If an input is one, and is excitatory in nature, it added one. If it was one, and was inhibitory, it subtracted one from the sum. This is done for all inputs, and a final sum is calculated. Now, if this final sum is less than some value (which you decide, say T), then the output is zero. Otherwise, the output is a one.

Every neuron model consists of a processing element with synaptic input connection and a single input. The "neurons" operated under the following assumptions:-

- i. They are binary devices ($V_i = [0,1]$)
- ii. Each neuron has a fixed threshold, θ values.
- iii. The neuron receives inputs from excitatory synapses, all having identical weights.
- iv. Inhibitory inputs have an absolute veto power over any excitatory inputs.
- v. At each time step the neurons are simultaneously (synchronously) updated by summing the weighted excitatory inputs and setting the output (V_i) to 1 if the sum is greater than or equal to the threshold and if the neuron receives no inhibitory input.

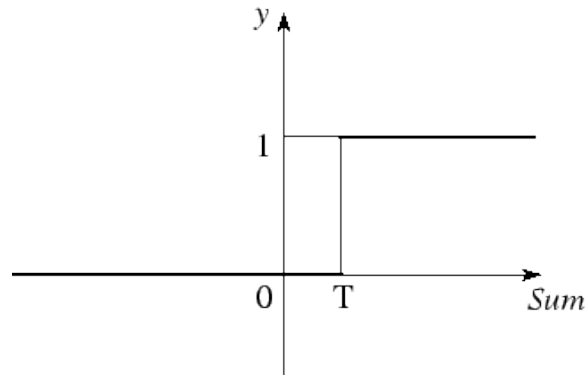
The McCulloch-Pitts Model of Neuron

The early model of an artificial neuron is introduced by Warren McCulloch and Walter Pitts in 1943. The McCulloch-Pitts neural model is also known as linear threshold gate. It is a neuron of a set of inputs $I_1, I_2, I_3, \dots, I_m$ and one output y . The linear threshold gate simply classifies the set of inputs into two different classes. Thus the output is y binary. Such a function can be described mathematically using these equations:

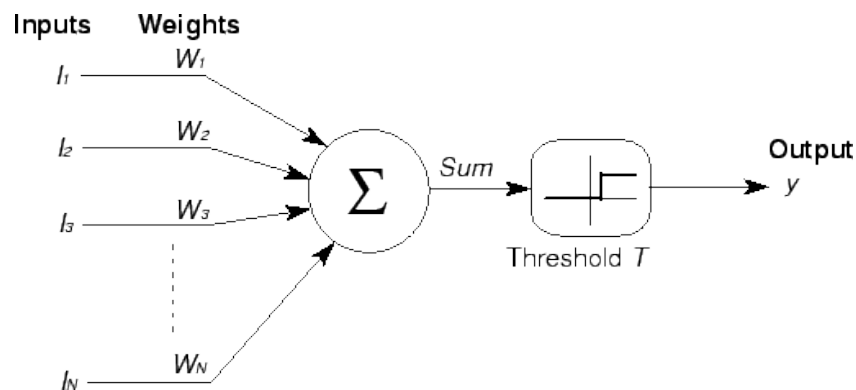
$$Sum = \sum_{i=1}^N I_i W_i, \quad (1.1)$$

$$y = f(Sum). \quad (1.2)$$

$W_1, W_2, W_3, W_4, \dots, W_m$ are weight values normalized in the range of either (0,1) or (-1,1) and associated with each input line, Sum is the weighted sum, and T is a threshold constant. The function f is a linear step function at threshold T as shown in figure. The symbolic representation of the linear threshold gate is shown in figure

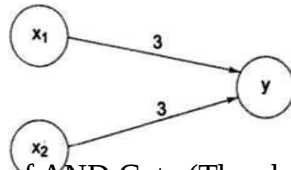


The McCulloch-Pitts model of a neuron is simple yet has substantial computing potential. It also has a precise mathematical definition. However, this model is so simplistic that it only generates a binary output and also the weight and threshold values are fixed. The neural computing algorithm has diverse features for various applications. Thus, we need to obtain the neural model with more flexible computational features.



By using McCulloch-Pitts model, we are going to solve the following logic gates.i. OR Gate ii. NOT Gate iii. AND Gate iv. NAND Gate v. XOR Gate vi. NOR Gate.

- **Implementation of McCulloch-Pitts model for ORgate**



Architecture of AND Gate (Threshold value per unit=3)

The net input is $Y_{in}=3A+3B$. The output is given by

$$Y=f(Y_{in})=\begin{cases} 1 & \text{if } Y_{in} \geq 3 \\ 0 & \text{if } Y_{in} < 3 \end{cases}$$

Results:

```

MATLAB R2012a
> e Edit Debug Parallel Desktop Window Help
> shortcuts How to Add What's New
> Command History
> Enter weights
> Weight w1=3
> Weight w2=3
> Enter Threshold value
> theta=3
> Output of Net
> 0 1 1 1
>
> Mccullotch-pitts Net for OR function
> Weights of Neuron
> 3
> 3
>
> Threshold value
> 3
  
```

- **Implementation of McCulloch-Pitts model for NOTgate**

Architecture of AND Gate (Threshold value=1)

Activation function= $Y=f(y_{in}) = 1$ if $y_{in} < 1$
 0 if $y_{in} \geq 1$



Results:

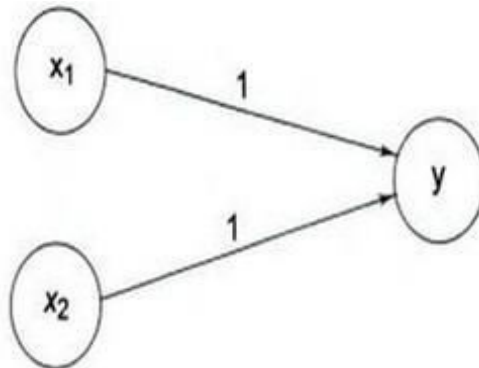
```
MATLAB R2012a
File Edit Debug Parallel Desktop Window Help
[Icons]
Shortcuts [How to Add] [What's New]
Command History

Enter weights
Weight w1=1
Enter Threshold value
theta=1
Output of Net
1 0

McCulloch-pitts Net for NOT function
Weights of Neuron
1

Threshold value
1
```

- Implementation of McCulloch-Pitts model for ANDgate



Architecture of AND Gate (Threshold value=2)

Net input is $y_{in}=A+B$. Output is given by $Y=f(y_{in})$

Activationfunction=
$$\begin{cases} 1 & \text{if } y_{in} \geq 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$$

Results:

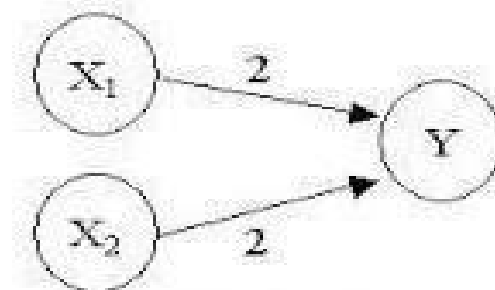
```

MATLAB R2012a
File Edit Debug Parallel Desktop Window Help
Shortcuts How to Add What's New
Enter weights
Weight w1=1
Weight w2=1
Enter Threshold value
theta=2
Output of Net
0 0 0 1

McCulloch-pitts Net for AND function
Weights of Neuron
1
1
Threshold value
2

```

- **Implementation of McCulloch-Pitts model for NANDgate**



Architecture of NAND Gate (Threshold value=4)

Net input is $y_{in}=x_1-x_2$. Output activation function is

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 4 \\ 0 & \text{if } y_{in} < 4 \end{cases}$$

Results:

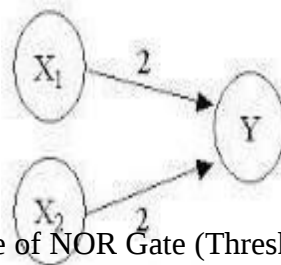
```

MATLAB R2012a
File Edit Debug Parallel Desktop Window Help
Shortcuts How to Add What's New
Command History
Enter weights
Weight w1=2
Weight w2=2
Enter Threshold value
theta=4
Output of Net
1 1 1 0

McCulloch-pitts Net for NAND function
Weights of Neuron
2
2
Threshold value
4

```

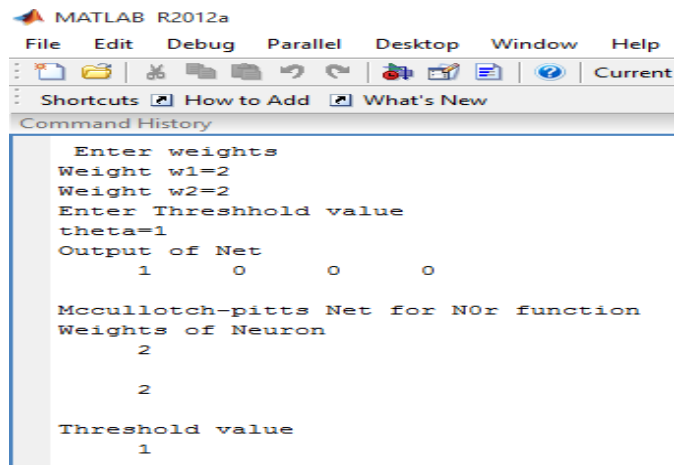
- **Implementation of McCulloch-Pitts model for NORgate**



Architecture of NOR Gate (Threshold value=1)

Activation function= 1 if $y_{in} \geq 1$
 0 if $y_{in} < 1$

Results:



```
MATLAB R2012a
File Edit Debug Parallel Desktop Window Help
...
Shortcuts How to Add What's New
Command History
Enter weights
Weight w1=2
Weight w2=2
Enter Threshold value
theta=1
Output of Net
1 0 0 0

McCulloch-pitts Net for NOR function
Weights of Neuron
2
2
Threshold value
1
```

Limitations of McCulloch-Pitts model

- i. Weights and thresholds are analytically determined.
- ii. Very difficult to minimize size of a network.

Conclusion

Hence we have studied and implemented basic logic gates using McCulloch-Pitts model.

Code:

```
t1=2  
t2=1  
yin=0
```

```
defAND():  
    w1=1  
    w2=1  
    x1=int(input("Enter first value : " ))  
    x2=int(input("Enter second value : "))  
  
    yin=x1*w1+x2*w2  
    if yin>=t1:  
        print("Your answer is 1")  
    else:  
        print("Your answer is 0")
```

```
defOR():  
  
    w1=1  
    w2=1  
    x1=int(input("Enter first value : " ))  
    x2=int(input("Enter second value : "))  
  
    yin=x1*w1+x2*w2  
    if yin>=t2:  
        print("Your answer is 1")  
    else:  
        print("Your answer is 0")
```



```

defNOT():
    w1=1
    x1=int(input("Enter value : " ))

    yin=x1*w1
    if yin<t2:
        print("Your answer is 1 ")

    else:
        print("Your answer is 0")


defANDNOT():

    w1=1
    w2=-1
    x1=int(input("Enter first value : " ))
    x2=int(input("Enter second value : "))

    yin=x1*w1+x2*w2
    if yin>=t2:
        print("Your answer is 1")
    else:
        print("Your ans is 0")


defdefault():
    print("Invalid choice")

ch='no'
while(ch!='yes'):
    switcher={
        1: AND,
        2: OR,
        3: NOT,
        4: ANDNOT,
        5: exit
    }
    defswitch(option):
        returnswitcher.get(option,default())

```

```
x=int(input("1: AND\n2: OR\n3: NOT\n4: NANDD\n5: Exit\n\nChoice : "))  
print(switch(x))
```

Output:

```
E:\CLG\SEM 8\SCOA\Outputs\Assignment 3>"C:/Program Files/Python37/python.exe"  
"e:/CLG/SEM 8/SCOA/Outputs/Assignment 3/Mc_culoch_logic_gates.py"
```

```
1: AND  
2: OR  
3: NOT  
4: NANDD  
5: Exit
```

Choice : 1

```
Enter first value : 2  
Enter second value : -1  
Your answer is 0
```

```
None  
1: AND  
2: OR  
3: NOT  
4: NANDD  
5: Exit
```

Choice : 1
Enter first value : 2
Enter second value : 3
Your answer is 1

```
None  
1: AND  
2: OR  
3: NOT  
4: NANDD  
5: Exit
```

Choice : 2
Enter first value : 1
Enter second value : 3
Your answer is 1

```
None  
1: AND  
2: OR  
3: NOT
```

4: NANDD

5: Exit

Choice : 3

Enter value : 5

Your answer is 0

None

1: AND

2: OR

3: NOT

4: NANDD

5: Exit

Choice : 3

Enter value : -1

Your answer is

None

1: AND

2: OR

3: NOT

4: NANDD

E:\CLG\SEM 8\SCOA\Outputs\Assignment 3>"C:/Program Files/Python37/python.exe"
"e:/CLG/SEM

8/SCOA/Outputs/Assignment 3/Mc_culoch_logic_gates.py"

1: AND

2: OR

3: NOT

4: NANDD

5: Exit

Choice : 1

Enter first value : 2

Enter second value : -1

Your answer is 0

None

1: AND

2: OR

3: NOT

4: NANDD

5: Exit

Choice : 1

Enter first value : 2
Enter second value : 3
Your answer is 1
None
1: AND
2: OR
3: NOT
4: NANDD
5: Exit

Choice : 2
Enter first value : 1
Enter second value : -2
Your answer is 0
None
1: AND
2: OR
3: NOT
4: NANDD
5: Exit

Choice : 3
Enter value : 5
Your answer is 0

4: NANDD

E:\CLG\SEM 8\SCOA\Outputs\Assignment 3>"C:/Program Files/Python37/python.exe"
"e:/CLG/SEM

Choice : 3
Enter value : -5
Your answer is 1
None
1: AND
2: OR
3: NOT
4: NANDD
5: Exit

Choice : 4
Enter first value : 2
Enter second value : 3
Your ans is 0
None
1: AND

2: OR
3: NOT
4: NANDD
5: Exit

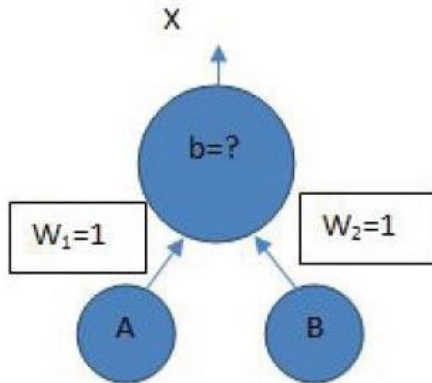
Choice : 4
Enter first value : -2
Enter second value : -3
Your answer is 1
None

1: AND
2: OR
3: NOT
4: NANDD
5: Exit

Assignment No. 5

Aim:

Write a program to find the Boolean function to implement following single layer perceptron. Assume all activation functions to be the threshold function which is 1 for all input values greater than zero and 0, otherwise.



Objectives:

1. To learn single layer perceptron.
2. To learn Boolean logic implementation using perceptron.

Software Requirements:

Ubuntu 16.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

The most common Boolean functions are AND, OR, NOT. The Boolean logic AND only returns 1 if both inputs are 1 else 0, Boolean logic OR returns 1 for all inputs with 1, and will only return 0 if both input is 0 and lastly logic NOT returns the invert of the input, if the input is 0 it returns 1, if the input is 1 it returns 0. To make it clear the image below shows the truth table for the basic BooleanFunction.

Truth table of AND gate:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of OR gate:

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

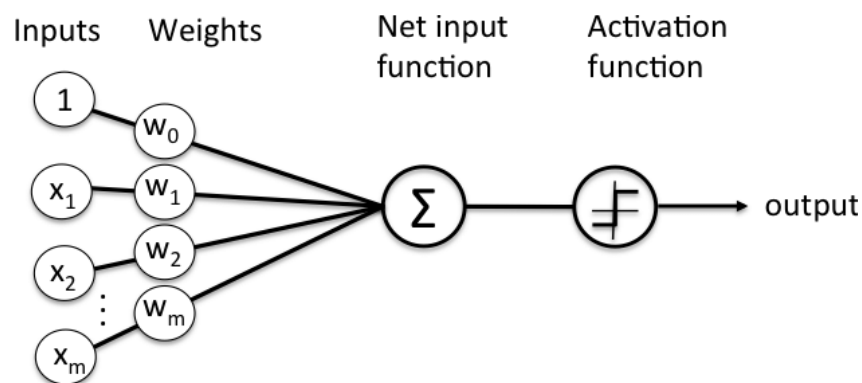
Truth table of NOT gate:

x	x^1
0	1
1	0

The columns X and Y are the inputs and last is the output. So, for the inputs $X = 0$, $Y = 0$ the output is 0.

Perceptron

A perceptron is the basic part of a neural network. A perceptron represents a single neuron on a human's brain, it is composed of the dataset (X_m), the weights (W_m) and an activation function, that will then produce an output and abias.



The datasets (inputs) are converted into an ndarray which is then matrix multiplied to another ndarray that holds the weights. Summing up all matrix multiply and adding a bias will create the

net input function, the output would then be passed into an activation function that would determine if the neuron needs to fire an output or not.

most layers of neural networks, it still is widely used for final classifications). The perceptron is simply separating the input into 2 categories, those that cause a fire, and those that don't. It does this by looking at (in the 2-dimensional case):

$$w_1 I_1 + w_2 I_2 < t$$

If the LHS is $< t$, it doesn't fire, otherwise it fires. That is, it is drawing the line:

$$w_1 I_1 + w_2 I_2 = t$$

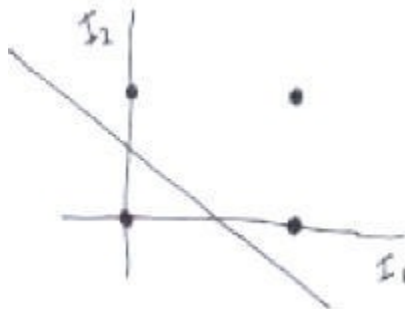
and looking at where the input point lies. Points on one side of the line fall into 1 category, points on the other side fall into the other category. And because the weights and thresholds can be anything, this is just *any line* across the 2 dimensional input space.

So what the perceptron is doing is simply drawing a line across the 2-d input space. Inputs to one side of the line are classified into one category, inputs on the other side are classified into another.

e.g. the OR perceptron, $w_1=1$, $w_2=1$, $t=0.5$, draws the line:

$$I_1 + I_2 = 0.5$$

across the input space, thus separating the points (0,1),(1,0),(1,1) from the point (0,0):



As you might imagine, not every set of points can be divided by a line like this. Those that can be, are called *linearly separable*.

In 2 input dimensions, we draw a 1 dimensional line. In n dimensions, we are drawing the $(n-1)$ dimensional *hyperplane*:

$$w_1 I_1 + \dots + w_n I_n = t$$

Conclusion

Thus we learned implementation of single layer perceptron for AND, OR and NOT Boolean function.

Code:

```
x1=[]
x2=[]
t=[]
w1=0
w2=0
b=0
y=0
alp=0
b=0
yin=0

def active(y):
    if(y>1):
        return 1
    else:
        return 0

def ino():

    n=int(input("Enter no of inputs : "))
    print("Enter x1 inputs : ")
    for i in range(n):
        x1.append(float(input()))

    print("Enter x2 inputs : ")
    for i in range(n):
        x2.append(float(input()))
    print("Enter threshold inputs : ")
    for i in range(n):
        t.append(float(input()))

    w1=float(input("Enter w1 : "))
    w2=float(input("Enter w2 : "))
    alp=float(input("Enter learning rate : "))
    b=float(input("Enter bias : "))
    for i in range(n):
        yin=(w1*x1[i] + w2*x2[i])+b
        y=active(yin)
        print("\nInput"+str(i)+" , y = "+str(y))
        if(y!=t[i]):
            w1=w1+(alp*t[i]*x1[i])
```

```
w2=w2+(alp*t[i]*x2[i])
```

```
b=b+alp*t[i]
```

```
print("\nNew w1 = " + str(w1) + " \nNew w2 = " + str(w2)+"\n New bias = "+str(b))
```

```
ino()
```

Output:

Enter no of inputs : 3

Enter x1 inputs :

0

1

0

Enter x2 inputs :

1

0

1

Enter threshold inputs :

1

0

1

Enter w1 : 1

Enter w2 : 1

Enter learning rate : 0.1

Enter bias : 0

Input0, y = 0

New w1 = 1.0

New w2 = 1.1

New bias = 0.1

Input1, y = 1

New w1 = 1.0

New w2 = 1.1

New bias = 0.1

Input2, y = 1

