**Department of Computer Science Engineering & Application**

**Soft Computing Lab File**

**Name – Puneet Kumar**

**Section – K**

**Roll No. – 33**

**University Roll No. – 191500612**

**Submitted To –**

**Dr. Praphula Kumar Jain**

INDEX

**Experiment 1 – Plot different types of Activation Function.**

**Linear Activation Function -**

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
def linear(x):
''' y = f(x) It returns the input as it is'''
return x
x = np.linspace(-10, 10)
plt.plot(x, linear(x))
plt.axis('tight')
plt.title('Activation Function :Linear')
plt.show()
```

**Sigmoid Activation Function**

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
def sigmoid(x):
''' It returns 1/(1+exp(-x)). where the values lies between zero and one '''
return 1/(1+np.exp(-x))
x = np.linspace(-10, 10)
plt.plot(x, sigmoid(x))
plt.axis('tight')
plt.title('Activation Function :Sigmoid')
plt.show()
```

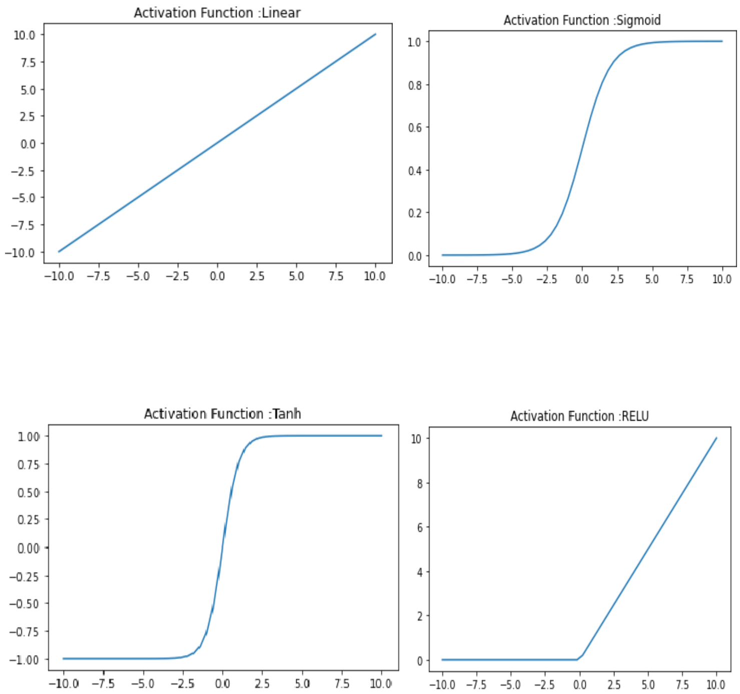
**Tanh Activation Function-**

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
def tanh(x):
''' It returns the value (1-exp(-2x))/(1+exp(-2x)) and the value returned will be lies in between -
1 to 1.'''
return np.tanh(x)
x = np.linspace(-10, 10)
plt.plot(x, tanh(x))
plt.axis('tight')
plt.title('Activation Function :Tanh')
plt.show()
```

**RELU Activation Function –**

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
def RELU(x):
''' It returns zero if the input is less than zero otherwise it returns the given input. '''
x1=[]
for i in x:
if i<0:
x1.append(0)
else:
x1.append(i)
```

```
return x1
x = np.linspace(-10, 10)
plt.plot(x, RELU(x))
plt.axis('tight')
plt.title('Activation Function :RELU')
plt.show()
```

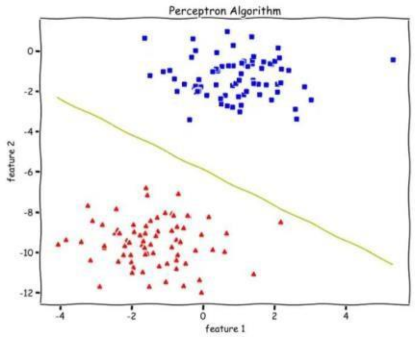


**Experiment 2- Program to create Single Layer Perceptron.**

```
def perceptron(X, y, lr, epochs):
# X --&gt; Inputs.
# y --&gt; labels/target.
# lr --&gt; learning rate.
# epochs --&gt; Number of iterations.
# m-&gt; number of training examples
# n-&gt; number of features
m, n = X.shape
# Initializing parapeters(theta) to zeros.
# +1 in n+1 for the bias term.
theta = np.zeros((n+1,1))
# Empty list to store how many examples were
# misclassified at every iteration.
n_miss_list = []
# Training.
for epoch in range(epochs):
# variable to store #misclassified.
n_miss = 0
# looping for every example.
for idx, x_i in enumerate(X):
# Insering 1 for bias, X0 = 1.
x_i = np.insert(x_i, 0, 1).reshape(-1,1)
# Calculating prediction/hypothesis.
y_hat = step_func(np.dot(x_i.T, theta))
# Updating if the example is misclassified.
if (np.squeeze(y_hat) - y[idx]) != 0:
theta += lr*((y[idx] - y_hat)*x_i)
# Incrementing by 1.
n_miss += 1
# Appending number of misclassified examples
# at every iteration.
n_miss_list.append(n_miss)
return theta, n_miss_list
# Plotting Features
def plot_decision_boundary(X, theta):
# X --&gt; Inputs
# theta --&gt; parameters
# The Line is y=mx+c
# So, Equate mx+c = theta0.X0 + theta1.X1 + theta2.X2
```

```
# Solving we find m and c
x1 = [min(X[:,0]), max(X[:,0])]
m = -theta[1]/theta[2]
c = -theta[0]/theta[2]
x2 = m*x1 + c
# Plotting
fig = plt.figure(figsize=(10,8))
plt.plot(X[:, 0][y==0], X[:, 1][y==0], &quot;r^&quot;)
plt.plot(X[:, 0][y==1], X[:, 1][y==1], &quot;bs&quot;)
plt.xlabel(&quot;feature 1&quot;)
plt.ylabel(&quot;feature 2&quot;)
plt.title('Perceptron Algorithm') plt.plot(x1, x2, &#39;y-&#39;)
# Training &amp; Plotting
theta, miss_l = perceptron(X, y, 0.5, 100)
plot_decision_boundary(X, theta)
```

**OUTPUT-**



**Experiment 3 - To implement AND function using ADALINE with bipolar input and output**

**Algorithm:**

**1. Initialize weight and bias to 0**

**2. Accept learning rate, alpha and threshold, theta**

**3. For each input calculate yin = b+x(1)*w(1)+x(2)*w(2)**

**4. Apply activation function**

**5. If calculated output ≠ target output**

**i) update weight and bias**

**ii) Go to step 3**

**6. Display final weight matrix and bias value**

**Program:**

```
% Perceptron for AND function
clear;
clc;
x=[1 1 - 1 -1; 1 -1 1 - 1];
t=[1 -1 -1 -1];
w=[0 0];
b=0;
alpha=input(‘Enter Learning rate=’);
theta=input(‘Enter Threshold Value=’);
con = 1;
epoch = 0;
while con
con=0;
for i=1:4
yin=b+x(1,i)*w(1)+x(2,i)*w(2);
if yin>theta
y=1;
end
if yin<=theta & yin>= -theta
y=0;
end
if yin < -theta
y = -1;
end
if y-t(i)
con=1;
for j=1:2
w(j)=w(j)+alpha*t(i)*x(j,i);
end
b=b+alpha*t(i);
```

```
end
epoch=epoch+1;
end
disp('Perceptron for AND Function');
disp('Final Weight Matrix');
disp(w);
disp('Final Bias');
disp(b);
```

**Sample Input and Output:**

**Enter Learning rate = 1**

**Enter Threshold Value =0.5**

**Perceptron for AND Function**

**Final Weight Matrix**

**1 1**

**Final Bias**

**-1**

**Experiment 4: To Construct and test Auto Associative Network for input vector using HEBB's Rule**

```
%Auotassociative net to store the vector

 clc;

clear;

x = [1 1 –1 –1];

 w=zeros (4, 4);

w=x'*x;

yin=x*w;

for i=1:4

if yin(i)>0

y(i)=1;

else

y(i) = –1;

 end

end

disp ('Weight matrix');

disp (w);

 if x == y disp ('The vector is a Known Vector');

 Else

 disp ('The vector is a Unknown Vector');

End
```

**OUTPUT:**

```
Weight matrix

1  1 –1 –1

1  1 –1 –1

–1 –1  1  1

–1 –1 1 1

The vector is a known vector.
```

**Experiment 5: Create a multilayer Perceptron & WAP for Back-propagation Network**

```
Import
numpy as np

# X = (hours sleeping, hours studying), y = test score of the student
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

# scale units
X = X/np.amax(X, axis=0) #maximum of X array
y = y/100 # maximum test score is 100


class NeuralNetwork(object):
  def __init__(self):
    #parameters
    self.inputSize = 2
    self.outputSize = 1
    self.hiddenSize = 3

    #weights
    self.W1 = np.random.randn(self.inputSize, self.hiddenSize) # (3x2) weight
matrix from input to hidden layer
    self.W2 = np.random.randn(self.hiddenSize, self.outputSize) # (3x1)
weight matrix from hidden to output layer

  def feedForward(self, X):
    #forward propogation through the network
    self.z = np.dot(X, self.W1) #dot product of X (input) and first set of
weights (3x2)
    self.z2 = self.sigmoid(self.z) #activation function
    self.z3 = np.dot(self.z2, self.W2) #dot product of hidden layer (z2) and
second set of weights (3x1)
    output = self.sigmoid(self.z3)
    return output

def sigmoid(self, s, deriv=False):
```

```
        if (deriv == True):
            return s * (1 - s)
        return 1/(1 + np.exp(-s))

    def backward(self, X, y, output):
        #backward propogate through the network
        self.output_error = y - output # error in output
        self.output_delta = self.output_error * self.sigmoid(output, deriv=True)

        self.z2_error = self.output_delta.dot(self.W2.T) #z2 error: how much our hidden layer weights contribute to output error
        self.z2_delta = self.z2_error * self.sigmoid(self.z2, deriv=True) #applying derivative of sigmoid to z2 error

        self.W1 += X.T.dot(self.z2_delta) # adjusting first set (input -> hidden) weights
        self.W2 += self.z2.T.dot(self.output_delta) # adjusting second set (hidden -> output) weights

    def train(self, X, y):
        output = self.feedForward(X)
        self.backward(X, y, output)

NN = NeuralNetwork()


for i in range(1000): #trains the NN 1000 times
    if (i % 100 == 0):
        print("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))
    NN.train(X, y)

print("Input: " + str(X))
print("Actual Output: " + str(y))
print("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))
```

**Experiment 6 - Program for fuzzy set operation and properties**

```
class FzSets:
def __init__(self):
        self.A = dict()
        self.B = dict()
        self.complement_A = dict()
        self.complement_B = dict()
        self.union_AB = dict()
        self.intersection_AB = dict()
        self.differenceAB = dict()
        self.differenceBA = dict()
        self.change_union = False
        self.change_intersection = False
        self.change_complement = False
def __init_(self,A,nA,B,nB):
        self.A = A
        self.B = B
        self.Aname = nA
        self.Bname = nB
        self.complement_A = dict()
        self.complement_B = dict()
        self.union_AB = dict()
        self.intersection_AB = dict()
        self.differenceAB = dict()
        self.differenceBA = dict()
        self.change_union = False
        self.change_intersection = False
        self.change_complement = False
def unionOp(self):
        if self.change_union:
        print('Result of UNION operation :',self.union_AB)
        else:
        #unionSet = set(self.A.keys()).union(self.B.keys())
        sa = set(self.A.keys())
        sb = set(self.B.keys())
        intersectionSet = set(self.A.keys()).intersection(self.B.keys())
        for i in intersectionSet:
        self.union_AB[i] = max(self.A[i],self.B[i])
        for i in sa-intersectionSet:
        self.union_AB[i] = self.A[i]
        for i in sb-intersectionSet:
```

```
self.union_AB[i] = self.B[i]
print('Result of UNION operation :',self.union_AB)
        def intersectionOp(self):
        if self.change_intersection:
        print('Result of INTERSECTION operation :\n\t\t',self.intersection_AB)
        else:
        #unionSet = set(self.A.keys()).union(self.B.keys())
        sa = set(self.A.keys())
        sb = set(self.B.keys())
        intersectionSet = set(self.A.keys()).intersection(self.B.keys())
        for i in intersectionSet:
                self.intersection_AB[i] = min(self.A[i],self.B[i])
        for i in sa-intersectionSet:
                self.intersection_AB[i] = 0.0
        for i in sb-intersectionSet:
                self.intersection_AB[i] = 0.0
        print('Result of INTERSECTION operation :\n\t\t',self.intersection_AB)
        self.change_intersection = True
def complementOp(self):
        if self.change_complement:
        print('Result of COMPLEMENT on ',self.Aname,' operation :',self.complement_A)
        print('Result of COMPLEMENT on ',self.Bname,' operation :',self.complement_B)
        else:
        for i in self.A:
                self.complement_A[i] = 1 - A[i]
        for i in self.B:
                self.complement_B[i] = 1 - B[i]
print('Result of COMPLEMENT on ',self.Aname,' operation :',self.complement_A)
        print('Result of COMPLEMENT on ',self.Aname,' operation :',self.complement_B)
self.change_complement = True
def _oneMinustwo(self,L,R):
        minus_d = dict()
        Rcomp = dict()
        for i in R:
        Rcomp[i] = 1 - R[i]
        sa = set(L.keys())
        sb = set(R.keys())
        intersectionSet = sa.intersection(sb) # min( A , complement(B) )
        # l - r OR a - b
        for i in intersectionSet:
        minus_d[i] = min(L[i],Rcomp[i])
```

```
for i in sa-intersectionSet:
        minus_d[i] = 0.0
        for i in sb-intersectionSet:
        minus_d[i] = 0.0
        return minus_d
        def AminusB(self):
        self.differenceAB = self.__oneMinustwo(self.A,self.B)
        print('Result of DIFFERENCE ',self.Aname,' | ',self.Bname,' operation
:\n\t\t',self.differenceAB)
def BminusA(self):
        self.differenceBA = self.__oneMinustwo(self.B,self.A)
        print('Result of DIFFERENCE ',self.Bname,' | ',self.Aname,' operation
:\n\t\t',self.differenceBA)
def change_Setz(self,A,B):
        self.A = A
        self.B = B
print('\nSet ',self.Aname,' :',self.A)
        print('Set ',self.Bname,' :',self.B,end='')
self.change_union = True
        self.change_intersection  =  True
        self.change_complement = True
        print('\t\t\t Cache Reset')
def displaySets(self):
        print('\nSet ',self.Aname,' :',self.A)
        print('Set ',self.Bname,' :' ,self.B)
```

**Experiment 7 - Program to find relation using max-min composition, enter the two vectors who's relation is to be find.**

**Program -**

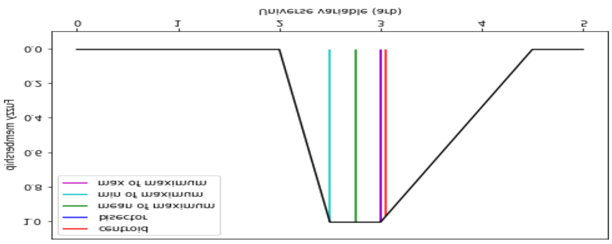
```
import numpy as np
# Max-Min Composition
def maxMin(x, y):
   z = []
   for x1 in x:
      for y1 in y.T:
         z.append(max(np.minimum(x1, y1)))
   return np.array(z).reshape((x.shape[0], y.shape[1]))
# Max-Product Composition given by Rosenfeld
def maxProduct(x, y):
   z = []
   for x1 in x:
      for y1 in y.T:
         z.append(max(np.multiply(x1, y1)))
   return np.array(z).reshape((x.shape[0], y.shape[1]))
# 3 arrays for the example
r1 = np.array([[1, 0, .7], [.3, .2, 0], [0, .5, 1]])
r2 = np.array([[.6, .6, 0], [0, .6, .1], [0, .1, 0]])
r3 = np.array([[1, 0, .7], [0, 1, 0], [.7, 0, 1]])
print "R1oR2 => Max-Min :\n" + str(maxMin(r1, r2)) + "\n"
print "R1oR2 => Max-Product :\n" + str(maxProduct(r1, r2)) + "\n\n"
print "R1oR3 => Max-Min :\n" + str(maxMin(r1, r3)) + "\n"
print "R1oR3 => Max-Product :\n" + str(maxProduct(r1, r3)) + "\n\n"
print "R1oR2oR3 => Max-Min :\n" + str(maxMin(r1, maxMin(r2, r3))) + "\n"
print "R1oR2oR3 => Max-Product :\n" + str(maxProduct(r1, maxProduct(r2, r3))) + "\n\n"
```

```
R1oR2 => Max-Min :
[[ 0.6  0.6  0. ]
 [ 0.3  0.3  0.1]
 [ 0.   0.5  0.1]]

R1oR2 => Max-Product :
[[ 0.6   0.6   0.  ]
 [ 0.18  0.18  0.02]
 [ 0.    0.3   0.05]]


R1oR3 => Max-Min :
[[ 1.   0.   0.7]
 [ 0.3  0.2  0.3]
 [ 0.7  0.5  1. ]]

R1oR3 => Max-Product :
[[ 1.    0.    0.7 ]
 [ 0.3   0.2   0.21]
 [ 0.7   0.5   1.  ]]


R1oR2oR3 => Max-Min :
[[ 0.6  0.6  0.6]
 [ 0.3  0.3  0.3]
 [ 0.1  0.5  0.1]]

R1oR2oR3 => Max-Product :
[[ 0.6    0.6    0.42 ]
 [ 0.18   0.18   0.126]
 [ 0.035  0.3    0.05 ]]
```

**Experiment 8 - Method Of Defuzzification**

**Program -**

```
pip install -U scikit-fuzzy
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz
# Generate trapezoidal membership function on range [0, 1]
x = np.arange(0, 5.05, 0.1)
mfx = fuzz.trapmf(x, [2, 2.5, 3, 4.5])
# Defuzzify this membership function five ways
defuzz_centroid = fuzz.defuzz(x, mfx, 'centroid')  # Same as skfuzzy.centroid
defuzz_bisector = fuzz.defuzz(x, mfx, 'bisector')
defuzz_mom = fuzz.defuzz(x, mfx, 'mom')
defuzz_som = fuzz.defuzz(x, mfx, 'som')
defuzz_lom = fuzz.defuzz(x, mfx, 'lom')
# Collect info for vertical lines
labels = ['centroid', 'bisector', 'mean of maximum', 'min of maximum',
          'max of maximum']
xvals = [defuzz_centroid,
         defuzz_bisector,
         defuzz_mom,
         defuzz_som,
         defuzz_lom]
colors = ['r', 'b', 'g', 'c', 'm']
ymax = [fuzz.interp_membership(x, mfx, i) for i in xvals]
# Display and compare defuzzification results against membership function
plt.figure(figsize=(8, 5))
plt.plot(x, mfx, 'k')
for xv, y, label, color in zip(xvals, ymax, labels, colors):
    plt.vlines(xv, 0, y, label=label, color=color)
plt.ylabel('Fuzzy membership')
plt.xlabel('Universe variable (arb)')
plt.ylim(-0.1, 1.1)
plt.legend(loc=2)
plt.show()
```



**Experiment 9: Program for complete Genetic Algorithm Cycle**

```
from random import randint
import random as rn
class Sample:
def __init_(self,s,score):
        self.s = s
        self.score = score
class GeneticEvolution:
def __init__(self,s):
self.s = s
        #97 122
        self.population = []
        self.mutation_rate = 0.1
        for i in range(100):
            sample = ''
            for i in range(len(s)):
                sample+= chr(int(randint(97,122)))
            self.population.append(Sample(sample,self.fitness(sample)))
    def fitness(self,sample):
        score=0
        for i,j in zip(list(sample),list(self.s)):
            if i==j:
                score+=1
        return score
    def selection(self):
        new_population = []
      self.population.sort(key = lambda x: x.score,reverse=True)
        new_population = self.population[:40]
       r_sample = rn.sample(self.population[40:],10)
        for i in r_sample:
             new_population.append(i)
        self.population = new_population
    def crossover(self):
        for i in range(100-len(self.population)):
```

```
            parents = rn.sample(self.population,2)
    child = ''
    for i,j in zip(list(parents[0].s),list(parents[1].s)):
                if rn.random()<0.5:
                    child+=i
                else:
                    child+=j
          self.population.append(Sample(child,self.fitness(child)))
          def mutation(self):
            new_population = []
        for sample in self.population:
            mutated = ''
            for i in sample.s:
                if rn.random()<self.mutation_rate:
                    mutated += chr(int(randint(97,122)))
                else:
                    mutated += i
        new_population.append(Sample(mutated,self.fitness(mutated)))
        self.population = new_population
      def evolution(self):
          generation = 0
          while True:
            generation+=1
            print(generation)
            self.selection()
            #condition
            if self.population[0].score==self.fitness(self.s):
                print(self.population[0].s,self.population[0].score)
                return self.population[0]
            self.crossover()
            self.mutation()
            if generation>1000:
                return None
         g = GeneticEvolution('hel')
```

```
        g.evolution()
def brute_force(s):
    count = 0
    while True:
        e = ''
        for i in range(len(s)):
            e+=chr(int(randint(97,122)))
        if e==s:
            return count
        print(e)
        if count>10000:
            return -1
        count+=1
```

**OUTPUT:**

**1**

**2**

**3**

**4**

**Experiment 10 - Write a program in MATLAB to implement De-Morgan’s Law.**

**De-Morgan’s Law c(i(u,v)) = max(c(u),c(v))**

**c(u(u,v)) = min(c(u),c(v))**

```
%Enter Data
u=input('Enter First Matrix');
v=input('Enter Second Matrix');
%To Perform Operations
w=max(u,v);
p=min(u,v);
q1=1-u;
q2=1-v;
x1=1-w;
x2=min(q1,q2);
y1=1-p;
y2=max(q1,q2);
%Display Output
display('Union Of Two Matrices');
display(w);
display('Intersection Of Two Matrices');
display(p);
display('Complement Of First Matrix');
display(q1);
display('Complement Of Second Matrix');
display(q2);
display('De-Morgans Law');
display('LHS');
display(x1);
display('RHS');
display(x2);
display('LHS1');
display(y1);
display('RHS1');
display(y2);
```

**Output:-**

**Enter First Matrix [0.3 0.4]**

**Enter Second Matrix [0.2 0.5]**

**Union Of Two Matrices**

**w =0.3000 0.5000**

**Intersection Of Two Matrices**

**p =0.2000 0.4000**

**Complement Of First Matrix**

**q1 =0.7000 0.6000**

**Complement Of Second Matrix**

**q2 =0.8000 0.5000**

**De-Morgans Law**

**LHS**

**x1 = 0.7000 0.5000**

**RHS**

**x2 = 0.7000 0.5000**

**LHS1**

**y1 =0.8000 0.6000**

**RHS1**

**y2 = 0.8000 0.6000**