



DEPARTMENT OF COMPUTER ENGINEERING & APPLICATIONS

Institute of Engineering & Technology

LAB MANUAL

Subject Name: Cryptography and Network Lab

Subject Code: BCSE0071

Submitted To:

Dr Asheesh Tiwari

(Assistant Professor)

Submitted By:

Abhisek Verma

B.Tech (CSE)

201500027 C-(7)

```

import numpy as np
def sig(x):
    return 1/(1 + np.exp(-x))

x = 1.0
print('Applying Sigmoid Activation on (%.1f) gives %.1f' % (x, sig(x)))

x = -10.0
print('Applying Sigmoid Activation on (%.1f) gives %.1f' % (x, sig(x)))

x = 0.0
print('Applying Sigmoid Activation on (%.1f) gives %.1f' % (x, sig(x)))

x = 15.0
print('Applying Sigmoid Activation on (%.1f) gives %.1f' % (x, sig(x)))

x = -2.0
print('Applying Sigmoid Activation on (%.1f) gives %.1f' % (x, sig(x)))

```

```

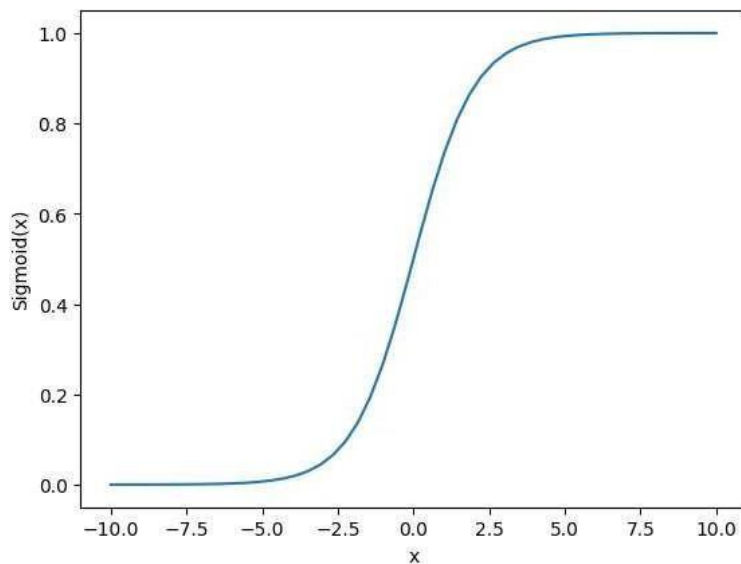
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10, 50)
p = sig(x)
plt.xlabel("x")
plt.ylabel("Sigmoid(x)")
plt.plot(x, p)
plt.show()

```

```

Applying Sigmoid Activation on (1.0) gives 0.7
Applying Sigmoid Activation on (-10.0) gives 0.0
Applying Sigmoid Activation on (0.0) gives 0.5
Applying Sigmoid Activation on (15.0) gives 1.0
Applying Sigmoid Activation on (-2.0) gives 0.1

```



```

#tanh or Hyperbolic:
import matplotlib.pyplot as plt
import numpy as np

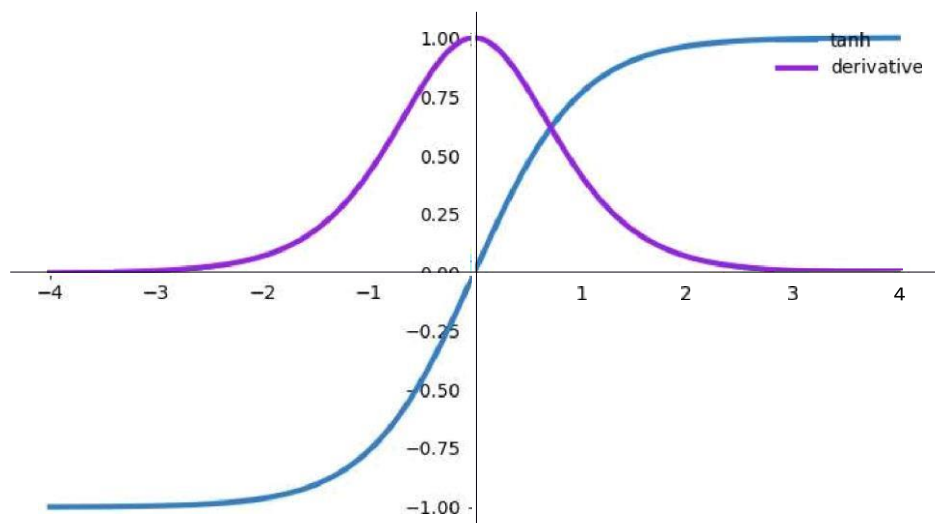
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return t,dt
z=np.arange(-4,4,0.01)
tanh(z[0].size,tanhz)[I.size
# Setup centered axes
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['right'].set_color('none')

```

```

ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(z, tanh(z)[1], color="#307EC7", linewidth=3, label="tanh")
ax.plot(z, tanh(z)[0], color="#9621E2", linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
fig.show()

```



- To write a program to create a perceptron network using command 'newp'

```
#@title To write a program to create a perceptron network using command 'newp'
#Ignore this command as it belongs to Matlab
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def load_data():
    URL_='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
    data = pd.read_csv(URL_, header = None)
    print(data)

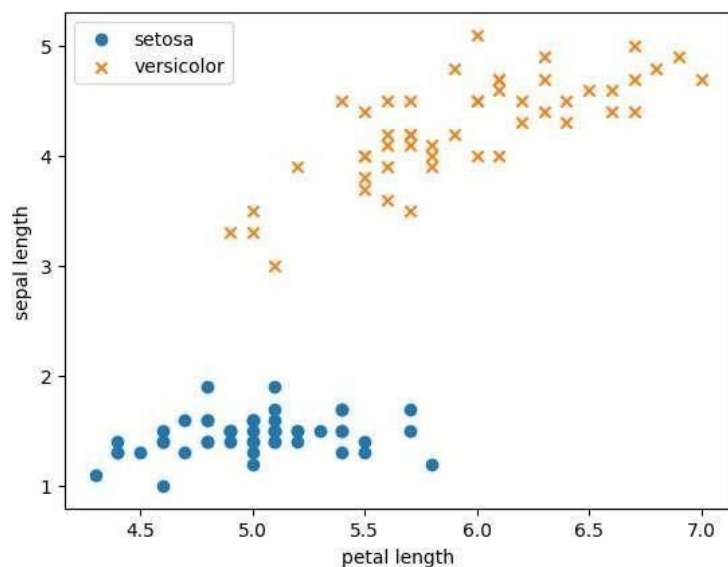
    # make the dataset linearly separable
    data = data[:100]
    data[4] = np.where(data.iloc[:, -1]=='Iris-setosa', 0, 1)
    data = np.asmatrix(data, dtype = 'float64')
    return data
data = load_data()
```

```
0      0      1      2      3      4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa

145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica
```

[150 rows x 5 columns]

```
plt.scatter(np.array(data[:50,0]), np.array(data[:50,2]), marker='o', label='setosa')
plt.scatter(np.array(data[50:,0]), np.array(data[50:,2]), marker='x', label='versicolor')
plt.xlabel('petal length')
plt.ylabel('sepal length')
plt.legend()
plt.show()
```



```
def perceptron(data, num_iter):
    features = data[:, :-1]
    labels = data[:, -1]

    # set weights to zero
    w = np.zeros(shape=(1, features.shape[1]+1))

    misclassified_ = []
```

```

for epoch in range(num_iter):
    misclassified = 0
    for x, label in zip(features, labels):
        x = np.insert(x,0,1)
        y = np.dot(w, x.transpose())
        target = 1.0 if (y > 0) else 0.0

        delta = (label.item(0,0) - target)

        if(delta): # misclassified
            misclassified += 1
            w += (delta * x)

    misclassified_.append(misclassified)
return (w, misclassified_)

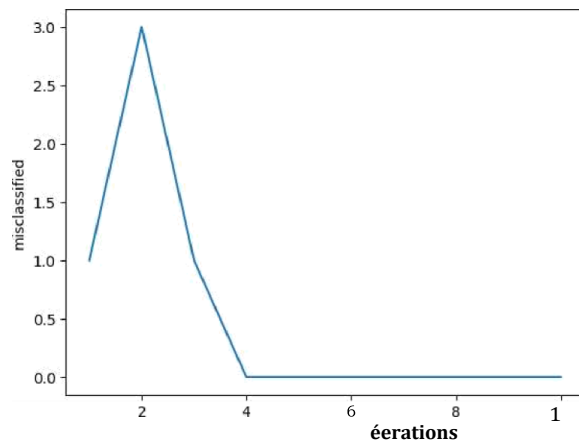
num_iter = 10
w, misclassified_ = perceptron(data, num_iter)

```

```

epochs = np.arange(1, num_iter+1)
plt.plot(epochs, misclassified_)
plt.xlabel('iterations')
plt.ylabel('misclassified')
plt.show()

```



- To implement single perceptron

#@title To implement single perceptron

```
import numpy as np
import pandas as pd
```

```
data=pd.read_csv('Iris.csv')
data.columns=['Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Type']
data.head(10)
```

	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Type
0	4.9	3.0	1.4	0.1	Setosa
1	4.7	3.2	1.3	0.4	Setosa
2	4.6	3.1	1.5	0.2	Setosa
3	5.0	3.6	1.4	0.2	Setosa
4	5.4	3.9	1.7	0.4	Setosa
5	4.6	3.4	1.4	0.3	Setosa
6	5.0	3.4	1.5	0.2	Setosa
7	4.4	2.9	1.4	0.3	Setosa
8	4.9	3.1	1.5	0.1	Setosa
9	5.4	4.4	1.5	0.4	Setosa

- Use Sigmoid function as the activation function

#@title Use Sigmoid function as the activation function

```
def activation_func(value): #Tangent Hypotense
    #return (1/(1+np.exp(-value)))
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

```
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yu=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+' : '+str(weights))
    print('Difference: '+str(weights-original))
    return weights
```

- Testing and Score

#@title Testing and Score

```
def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
```

```

        y[key]=1
    elif a<=(-0.7):
        y[key]=-1
    return y

def score(result, labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))

#@ Main code

# Dividing DataFrame "data" into "d_train" (60%) and "d_test" (40%)
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[-divider]

# Dividing d_train into data and labels/targets
d_train_y=d_train['Type']
d_train_X=d_train.drop(['Type'],axis=1)

# Dividing d_test into data and labels/targets
d_test_y=d_test['Type']
d_test_X=d_test.drop(['Type'],axis=1)

# Learning rate
alpha = 0.01

# Train
weights = perceptron_train(d_train_X, d_train_y, alpha)

```

```

Iteration 0: [0.62085819 0.24880642 0.49123374 0.11735
Iteration 1: [0.66785019 0.28080642 0.50423374 0.11935
Iteration 2: [0.71785019 0.31680642 0.51823374 0.12135
Iteration 3: [0.76785019 0.35080642 0.53323374 0.12335
Iteration 4: [0.81685019 0.38180642 0.54823374 0.12435
Iteration 5: [0.87085019 0.41880642 0.56323374 0.12635
Iteration 6: [0.91885019 0.45280642 0.57923374 0.12835
Iteration 7: [0.96685019 0.48280642 0.59323374 0.12935
Iteration 8: [1.02485019 0.52280642 0.60523374 0.13135
Iteration 9: [1.08185019 0.56680642 0.62023374 0.13535
Iteration 10: [1.13585819 0.60580642 0.63323374 0.139
Iteration 11: [1.18685019 0.64080642 0.64723374 0.142
Iteration 12: [1.23785019 0.67880642 0.66223374 0.145
Iteration 13: [1.28885019 0.71580642 0.67723374 0.149
Iteration 14: [1.33485019 0.75180642 0.68723374 0.151
Iteration 15: [1.38585019 0.78480642 0.70423374 0.156
Iteration 16: [1.43385019 0.81880642 0.72323374 0.158
Iteration 17: [1.48385019 0.84880642 0.73923374 0.160
Iteration 18: [1.53585019 0.88380642 0.75423374 0.162
Iteration 19: [1.58785019 0.91780642 0.76823374 0.164
Iteration 20: [1.63485019 0.94980642 0.78423374 0.166
Iteration 21: [1.68885019 0.98380642 0.79923374 0.170
Iteration 22: [1.74085019 1.02480642 0.81423374 0.171
Iteration 23: [1.79585019 1.06680642 0.82823374 0.173
Iteration 24: [1.84485019 1.09780642 0.84323374 0.174
Iteration 25: [1.89485019 1.12980642 0.85523374 0.176
Iteration 26: [1.94985019 1.16480642 0.86823374 0.178
Iteration 27: [1.99885019 1.19580642 0.88323374 0.179
Iteration 28: [2.04285019 1.22580642 0.89623374 0.181
Iteration 29: [2.09285019 1.26080642 0.90923374 0.184
Iteration 30: [2.13685019 1.29280642 0.92223374 0.186
Iteration 31: [2.18685019 1.32780642 0.93823374 0.192
Iteration 32: [2.23785019 1.36580642 0.95423374 0.194
Iteration 33: [2.29085019 1.40280642 0.96923374 0.196
Iteration 34: [2.34085019 1.43580642 0.98323374 0.198
Iteration 35: [2.48085019 1.49980642 1.07723374 0.226
Iteration 36: [2.61885019 1.56180642 1.17523374 0.256

```

```
Iteration42: [3.31485019 1.87780642 1.66323374 0.40435913]
Iteration43: [3.44885019 1.93980642 1.75123374 0.43235913]
Iteration44: [3.56085019 1.99980642 1.84123374 0.46235913]
Iteration45: [3.67685019 2.05380642 1.92323374 0.48235913]
Iteration46: [3.78885019 2.10380642 2.00123374 0.50435913]
Iteration47: [3.90685019 2.16780642 2.09723374 0.54035913]
Iteration48: [4.03285019 2.21780642 2.19523374 0.57035913]
Iteration49: [4.15485019 2.27380642 2.28923374 0.59435913]
Iteration50: [4.29085019 2.32980642 2.38523374 0.62235913]
Iteration51: [4.42485019 2.38980642 2.48523374 0.65635913]
Iteration52: [4.53885019 2.44180642 2.55523374 0.67635913]
Iteration53: [4.64885019 2.48980642 2.63123374 0.69835913]
Iteration54: [4.75885019 2.53780642 2.70523374 0.71835913]
Iteration55: [4.87485019 2.59180642 2.78323374 0.74235913]
Iteration56: [4.99485019 2.65980642 2.87323374 0.77435913]
Iteration57: [5.12885019 2.72180642 2.96723374 0.80435913]
```


- To implement OR and AND functions using ADALINE with bipolar inputs and output

#@title To implement OR and AND functions using ADALINE with bipolar inputs and output
[#https://www.geeksforgeeks.org/implementing-and-not-gate-using-adaline-network/](https://www.geeksforgeeks.org/implementing-and-not-gate-using-adaline-network/)

- OR gate implementation

```
#@title OR gate implementation
import numpy as np

# Define the Adaline class
class AdalineGD(object):
    def __init__(self, learning_rate=0.01, n_iter=50):
        self.learning_rate = learning_rate
        self.n_iter = n_iter

    def fit(self, X, y):
        self.weights_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.weights_[1:] += self.learning_rate * X.T.dot(errors)
            self.weights_[0] += self.learning_rate * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.weights_[1:]) + self.weights_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return np.where(self.activation(X) >= 0.0, 1, 0)

# Define the OR gate input and outputs
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

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

# Create the Adaline and train it
adaline = AdalineGD(learning_rate=0.01, n_iter=10)
adaline.fit(X, y)

# Test the Adaline network
print("Predictions:")
for xi in X:
    print(xi, "->", adaline.predict(xi))

Predictions:

[0 0] -> 0
[0 1] -> 1
[1 0] -> 1
[1 1] -> 1
```

- AND gate implementation

```
#@title AND gate implementation
import numpy as np

class AdalineGD(object):
    def __init__(self, learning_rate=0.01, n_iter=50):
        self.learning_rate = learning_rate
        self.n_iter = n_iter

    def fit(self, X, y):
```

```

self.weights_ = np.zeros(1 + X.shape[1])
self.cost_ = []

for i in range(self.n_iter):
    output = self.net_input(X)
    errors = (y - output)
    self.weights_[1:] += self.learning_rate * X.T.dot(errors)
    self.weights_[0] += self.learning_rate * errors.sum()
    cost = (errors**2).sum() / 2.0
    self.cost_.append(cost)
return self

def net_input(self, X):
    return np.dot(X, self.weights_[1:]) + self.weights_[0]

def activation(self, X):
    return self.net_input(X)

def predict(self, X):
    return np.where(self.activation(X) >= 0.0, 1, 0)

# Define the OR gate input and outputs
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

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

# Create an instance of the Adaline class
ad = Adaline(e=0.01, n_iter=10)
adaline.fit(X, y)

# Test the Adaline network
print("Predictions:")
for xi in X:
    print(xi, "->", adaline.predict(xi))

```

- Backpropagation algorithm with multilayer perceptron network

```
#@title Backpropagation algorithm with multilayer perceptron network
```

```
from math import exp
from random import seed
from random import random

# Initialize a network
def initialize_network(u_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
    network.append(output_layer)
    return network

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    ennon += (neuron['weights'][j] * neuron['delta'])
                errors.append(ennon)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(neuron['output'] - expected[j])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] -= l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] -= l_rate * neuron['delta']
```

```
# Train a network for a fixed number of epochs
def ttrain_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for now in train:
            outputs = forward_propagate(network, row)
            expected = [B for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
```

```
# Test training backprop algorithm
```

```
seed(1)
```

```
dataset = [[2.7810836,2.550537003,0],
[1.465489372,2.362125076,0],
[3.396561688,4.400293529,0],
[1.38807019,1.850220317,0],
[3.06407232,3.005305973,0],
[7.627531214,2.759262235,1],
[5.332441248,2.088626775,1],
[6.922596716,1.77106367,1],
[8.675418651,-0.242068655,1],
[7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 3, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

```
>epoch=0, lrate=0.500, error=6.880
>epoch=1, lrate=0.500, error=5.740
>epoch=2, lrate=0.500, error=5.326
>epoch=3, lrate=0.500, error=5.338
>epoch=4, lrate=0.500, error=5.299
>epoch=5, lrate=0.500, error=5.136
>epoch=6, lrate=0.500, error=4.924
>epoch=7, lrate=0.500, error=4.692
>epoch=8, lrate=0.500, error=4.419
>epoch=9, lrate=0.500, error=4.112
>epoch=10, lrate=0.500, error=3.785
>epoch=11, lrate=0.500, error=3.456
>epoch=12, lrate=0.500, error=3.138
>epoch=13, lrate=0.500, error=2.838
>epoch=14, lrate=0.500, error=2.563
>epoch=15, lrate=0.500, error=2.313
>epoch=16, lrate=0.500, error=2.089
>epoch=17, lrate=0.500, error=1.889
>epoch=18, lrate=0.500, error=1.710
>epoch=19, lrate=0.500, error=1.552
[{'weights': [0.2663982735658251, 0.7860076057871438, 0.7603367533444867], 'output': 0.9961594902966306, 'delta': -0.000112669012226361f
[{'weights': [-0.7317965482405842, 2.3716621057165623, -0.5690560485083125, -0.03474552839897625], 'output': 0.23494719808744585, 'delta':
```

- Program for fuzzy set with properties and operations.

```
#@title Program for fuzzy set with properties and operations.
```

Example to Demonstrate the Union of Two Fuzzy Sets

```
#@title Example to Demonstrate the Union of Two Fuzzy Sets
A = dict()
B = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)

for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]

    if A_value > B_value:
        Y[A_key] = A_value
    else:
        Y[B_key] = B_value

print('Fuzzy Set Union is :', Y)

The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Union is : {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
```

Example to Demonstrate Intersection of Two Fuzzy Sets

```
#@title Example to Demonstrate Intersection of Two Fuzzy Sets
A = dict()
B = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)

for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]

    if A_value < B_value:
        Y[A_key] = A_value
    else:
        Y[B_key] = B_value

print('Fuzzy Set Intersection is :', Y)

The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Intersection is : {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}
```

- Example to Demonstrate the Difference Between Two Fuzzy Sets

```
#@title Example to Demonstrate the Difference Between Two Fuzzy Sets
A = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

print('The Fuzzy Set is :', A)
```

```

for A_key in A:
    Y[A_key]= 1-A[A_key]

print('Fuzzy Set Complement is :', Y)

The Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
Fuzzy Set Complement is : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}

```

- Example to Demonstrate the Difference Between Two Fuzzy Sets

```

#@title Example to Demonstrate the Difference Between Two Fuzzy Sets
A = dict()
B = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)

for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]
    B_value = 1 - B_value

    if A_value < B_value:
        Y[A_key] = A_value
    else:
        Y[B_key] = B_value

print('Fuzzy Set Difference is :', Y)

The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Difference is : {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.6, 'd': 0.5}

```

- Program for composition on Fuzzy and Crips relations

```
#@title Program for composition on Fuzzy and Crips relations
```

```
Inport numpy as np
```

```
# Max-Min Composition given by Zadeh
```

```
def maxN1n(x, y):
```

```
    l-or x1 in x:
        for y1 in y.T:
            z.append(max(np.minimum(xl, yl) ))
    return np.array(z).reshape((x.shape[0], y.shape[1]))
```

```
# Max-Product Composition given by Roseufeld
```

```
de-F- maxProduct(x, y) :
```

```
    for x1 in x:
        for y1 in y.T:
            z.append(max(np.multiply(x1, yl)))
    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.annay([[1, 0, .7], [0, 1, 0], [.7, 0, 1]])
```

```
print ("R1oR2 => Max-M1n : \n" + str(maxM1n(r1, r2) ) + "\n")
print ("R1oR2 => Max-Product : \n" + str(maxProduct(r1, r2)) + "\n\n")
```

```
print ("R1oR3 => Nax-N1n : \n" + str(rraxH1n(r1, r3)) + "\n")
print ("R1oR3 => Max-Product : \n" + str(maxProduct(r1, r3)) + "\n\n")
```

```
print ("R1oR2oR3 => Max-M1n : \n" + str(maxM1n(r1, maxM1n(r2, r3) )) + "\n")
print ("R1oR2oR3 => Max-Product : \n" + str(maxProduct(r1, maxProduct(r2, r3))) + "\n\n")
```

```
➡ R1oR2 => Max-M1n
[[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-M1n :
[[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 ]]
```

- Example of Defuzzification operation

```
#@title Example of Defuzzification operation
```

```
!pip install -U scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
    994.0/994.0 kB 6.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (1.25.2)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (1.11.4)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (3.3)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=894078 sha256=5de7ba1f887f7a65c1a8d49aed3f53cbbfa93:
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bcebd95d3fedc7be7944db022094bc8
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

```
import Humpy 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)
mf1 = fuzz.trapmf(x, [2, 2.5, 3, 4.5])
```

```
# Defuzzify this membership function five ways
defuzz_centroid = fuzz.defuzz(x, mf1, 'centroid') # Same as skfuzzy.centroid
defuzz_bisector = fuzz.defuzz(x, mf1, 'bisector')
defuzz_mom = fuzz.defuzz(x, mf1, 'mom')
defuzz_som = fuzz.defuzz(x, mf1, 'som')
defuzz_lom = fuzz.defuzz(x, mf1, 'lom')
```

```
defuzz_som
```

```
2.5
```

```
# Collect info for vertical lines
labels = ['centroid', 'bisector', 'mean of maximum', 'mid 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, mf1, i) for i in xvals]
```

```
defuzz_som
```

```
2.5
```

```
plt.figure(figsize=(12, 8))
```

```
<Figure size 1200x800 with 0 Axes>
<Figure size 1200x800 with 0 Axes>
```

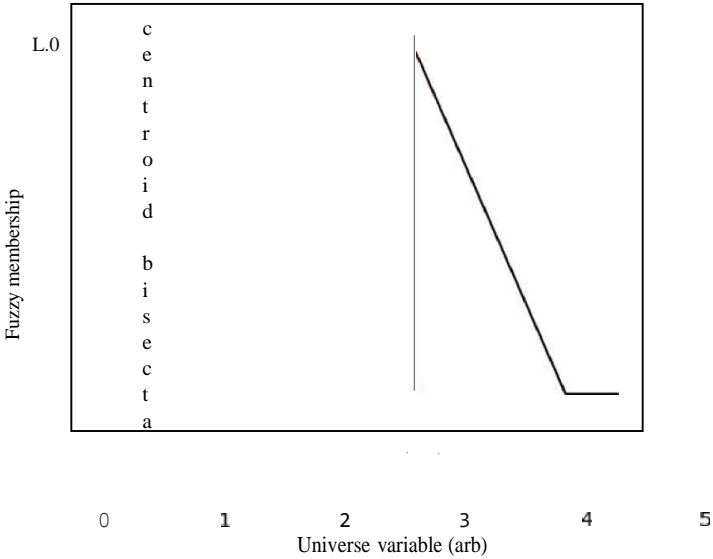
```
plt.plot(x, mf1, 'k')
for xv, y, label, color in zip(xvals, ymax, labels, colors):
    plt.vlines(xv, 0, y, label=label, color=color)
plt.ylabel('Fuzzy membership')
plt.xlabel('Universe variable (arb)')
```



```
plt.ylim(-0.1, 1.1)

plt.legend(loc=2)

plt.show()
```



- Program for design of fuzzy inference system using membership funtion

```
#@title Program for design of fuzzy inference system using membership funtion
```

```
pip install scikit-fuzzy scikit-learn
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
    994.0/994.0 kB 8.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (1.25.2)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (1.11.4)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (3.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.8.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=894078 sha256=b921463f4a4ecfe655442cb493e9bc55fdc57j
  Stored in directory: /root/.cache/pip/wheels/4f/86/1b/dfd97134a2c8313e519bcebd95d3fedc7be7944db022094bc8
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

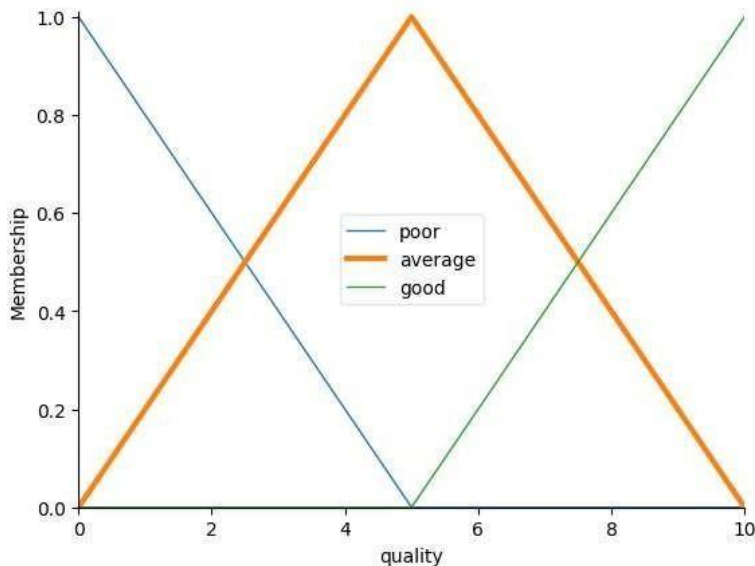
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

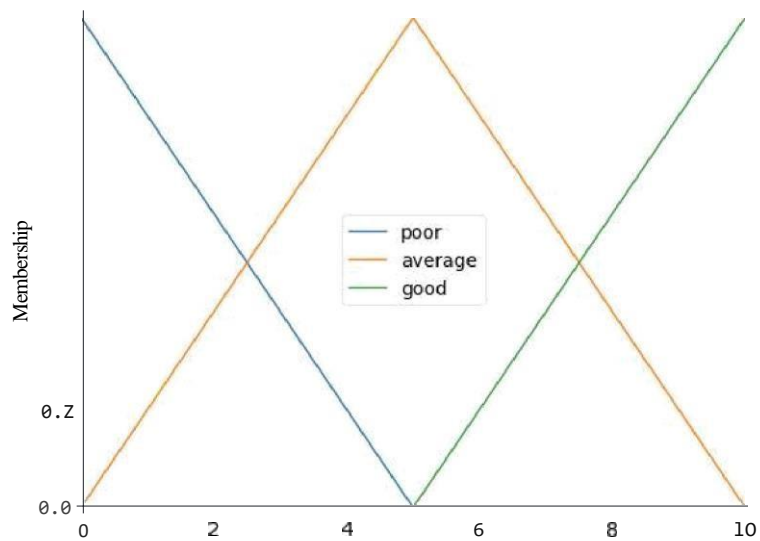
# Auto-membership function population is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

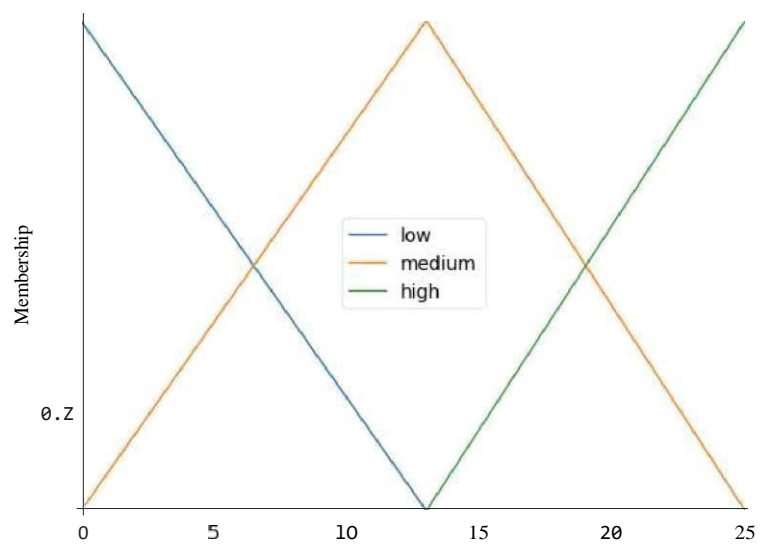
# You can see how these look with .view()
quality['average'].view()
```



```
service.view()
```



```
tip.view()
```



```
rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()
```

```
(Figure size 640x480 with 1 Axes, Axes: >)  
  
tippin = ctrl.  
ctrl.ControlSystem([rule1,  
rule2, rule3])  
=
```

