MALAYSIA-JAPAN INTERNATIONAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF SOFTWARE ENGINEERING
BACHELOR OF COMPUTER SCIENCE (SOFTWARE ENGINEERING)

SECJ3563-15 (COMPUTATIONAL INTELLIGENCE)

SESSION 20212022/2

ALTERNATIVE ASSESMENT REPORT

SECTION 15

PREPARED FOR:

NOOR HAYATI BINTI MOHD ZAIN

PREPARED BY:

MD YUSUF BIN FORKAN (A19MJ0178)

Abstract

Alternative Assessment was provided on 28th July and to be submitted on 7th July. Question 1 presents a problem where ANN architecture needs to be designed and parameter needs to be proposed for the solution. With the proposed parameter, the architecture should be implemented in a neural network tool (WEKA, RapidMiner, MATLAB, etc). This report proposes the solution using python where implementation is done in Google Colab. Question 2 presents a problem where Fuzzy Logic is proposed to obtain a processing time based on number of plates and level of dirtiness in a dishwasher appliance. The logic for input(antecedents) and output (consequents) are provided. Fuzzy Inference System (FIS) needs to be developed using Mamdani model. TEN fuzzy rules need to implemented using AND & OR operators. This report proposes the solution using python where implementation is done in Google Colab.

Table of Contents

Question 1

1. Introduction

Intelligence Solutions Sdn. Bhd. provides intelligence system solutions worldwide. The business was recently contracted to classify wheat types. The UC Irvine Machine Learning Repository presents measurements of three wheat kinds of kernels. The experiment used 70 randomly selected Kama, Rosa, and Canadian wheat kernels. A soft X-ray method discovered high-quality internal kernel structure. Non-destructive and cheaper than scanning microscopy or laser imaging. KODAK 13x18 cm X-ray plates recorded the images. Studies were conducted utilising combine-harvested wheat grain from Lublin's Institute of Agrophysics. The task inputs describe three wheat seed kinds (as listed in UC Irvine machine learning repository). Supervised Learning is ML's classification algorithm. This method uses labelled datasets to train classification or prediction algorithms. During cross-validation, the sequential model's weights change when fresh data is added. This continues until the model is fitted. It learns a function that converts input to output from input-output pairings. 80% of the data relates to training, and 20% to seed testing. Python codes the model.

To construct the data, seven geometric parameters of wheat kernels were measured:

1. Area A
2. Perimeter P
3. Compactness C = 4*pi*A/P^2
4. Length of kernel
5. Width of kernel
6. Asymmetry coefficient
7. Length of kernel groove
All of these parameters are real-valued continuous.

2. Objective

Dataset with 7 attributes of 3 types of wheat samples are provided.

- To find a distinct relation between the physical attributes of wheat seeds
- To find the type of wheat.

3. Proposed ANN architecture

The proposed ANN architecture is provided in the image below.

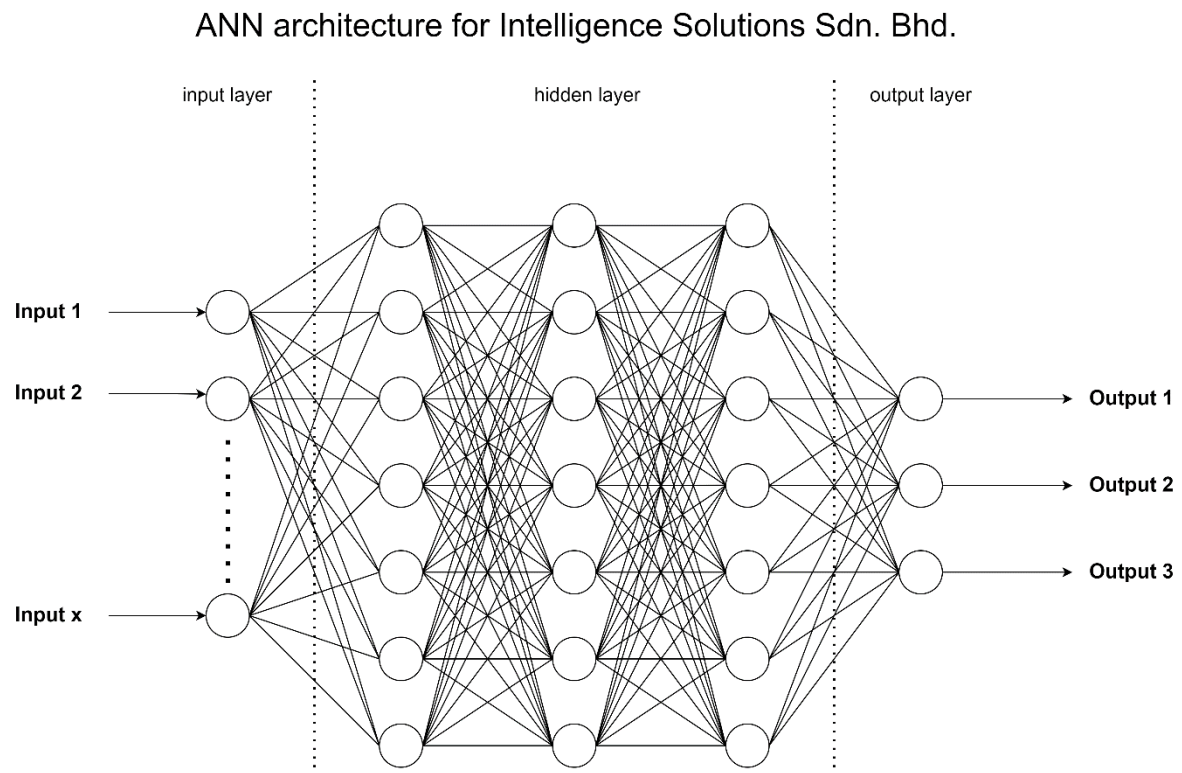

Figure 1 Proposed ANN architecture

4. Proposed ANN parameter

The proposed parameters are input layer, hidden layer, and output layer. The data from the seed_dataset is fed into x input layer which will pass through 3 hidden layer each containing 7 neurons. The 3 neurons are in the output layer to 3 types of output. Softmax classifier is used because it can predict multiple outputs.

5. Implementation

The dataset is downloaded from the UC Irvine machine learning repository and loaded into Google Colab for analysis. Implementation using python code is shown below.

```
[1]  import tensorflow as tf
     import pandas as pd
     from sklearn.model_selection import train_test_split
     import matplotlib.pyplot as plt
     import numpy as np
```

Figure 2 Import required library

```
[11] colnames=['AREA','PERIMETER','COMPACTNESS','LENGTH','WIDTH','ASSYMMETRY_COEFFICIENT','GROOVE_LENGTH','TYPE']
     df = pd.read_csv('seeds_dataset.txt',  sep='\t', names= colnames )
```

Figure 3 Name table identified from dataset accordingly

```
[3]  df.head()
```

	AREA	PERIMETER	COMPACTNESS	LENGTH	WIDTH	ASSYMMETRY_COEFFICIENT	GROOVE_LENGTH	TYPE
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1.0
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1.0
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1.0
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1.0
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1.0

Figure 4 Reading data from seed_dataset file

```
[4]  df = df.dropna()
```

```
[5]  train_set = df.iloc[:,:7]
     test_set = df.iloc[:,7:]
     test_set.head()
```

	TYPE
0	1.0
1	1.0
2	1.0
3	1.0
4	1.0

```
[6]  test_set = pd.get_dummies(test_set.TYPE)
```

```
[7]  X_train, X_test, y_train, y_test = train_test_split(train_set, test_set,train_size=0.80, test_size=0.20)
```

```
[8]  y_train.head()
```

	1.0	2.0	3.0
163	0	0	1
126	0	1	0
180	0	0	1
83	0	1	0
166	0	0	1

Figure 5 Data 80% trained and 20% Tested

The image below shows the sequential model of the proposed architecture with parameters. Here random seed set is 30. There are 3 layers of (tf.keras.layers.Dense(7, activation='relu') which depict 3 hidden layer, 7 is the neurons in the hidden layer and "ReLu" function is used. Last layer of (tf.keras.layers.Dense(3,activation='softmax') depict output layer with 3 neurons and softmax classifier is used.

```
tf.random.set_seed = 30

model_1 = tf.keras.Sequential([
              tf.keras.layers.Dense(7, activation='relu'),
              tf.keras.layers.Dense(7, activation='relu'),
              tf.keras.layers.Dense(7, activation='relu'),
              tf.keras.layers.Dense(3,activation='softmax')

])
```

Figure 6 Sequential Model with proposed parameters

The sequential model is compiled and accuracy loss function for trained data is calculated. RMSprop is used for optimization with a learning rate 0.001 to measure the accuracy. The model is fitted with training data to train the model with epoch of 49 to get the optimal accuracy.

```
model_1.compile(loss= tf.keras.losses.CategoricalCrossentropy(),
                optimizer = tf.keras.optimizers.RMSprop(learning_rate= 0.001),
                metrics = ['accuracy'])

history_1 = model_1.fit(X_train,y_train, epochs=49)
```

Figure 7 Accuracy for trained data

6. Result Analysis

After trial and error 49 epoch on training dataset which showed 81.13% accuracy to get the optimal graph shown below.

```
Epoch 1/49
5/5 [==============================] - 1s 3ms/step - loss: 1.1071 - accuracy: 0.1321
Epoch 2/49
5/5 [==============================] - 0s 3ms/step - loss: 1.0617 - accuracy: 0.1698
Epoch 3/49
5/5 [==============================] - 0s 3ms/step - loss: 1.0365 - accuracy: 0.3082
Epoch 4/49
5/5 [==============================] - 0s 4ms/step - loss: 1.0152 - accuracy: 0.3585
Epoch 5/49
5/5 [==============================] - 0s 4ms/step - loss: 0.9943 - accuracy: 0.4151
Epoch 6/49
5/5 [==============================] - 0s 4ms/step - loss: 0.9732 - accuracy: 0.4528
Epoch 7/49
5/5 [==============================] - 0s 3ms/step - loss: 0.9572 - accuracy: 0.4780
Epoch 8/49
5/5 [==============================] - 0s 3ms/step - loss: 0.9432 - accuracy: 0.4717
Epoch 9/49
5/5 [==============================] - 0s 4ms/step - loss: 0.9327 - accuracy: 0.4528
Epoch 10/49
5/5 [==============================] - 0s 3ms/step - loss: 0.9210 - accuracy: 0.4528
Epoch 11/49
5/5 [==============================] - 0s 3ms/step - loss: 0.9089 - accuracy: 0.4277
Epoch 12/49
5/5 [==============================] - 0s 4ms/step - loss: 0.8989 - accuracy: 0.4403
Epoch 13/49
...
Epoch 48/49
5/5 [==============================] - 0s 4ms/step - loss: 0.5784 - accuracy: 0.8365
Epoch 49/49
5/5 [==============================] - 0s 3ms/step - loss: 0.5692 - accuracy: 0.8113
```

Figure 8 Loss function and accuracy of model trained

The model is tested, and the accuracy is 90% which is higher than the model trained.

```
model_1.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 5ms/step - loss: 0.5971 - accuracy: 0.9000
[0.5971323847770691, 0.8999999761581421]
```

Figure 9 Model tested for accuracy

```
pd.DataFrame(history_1.history).plot(figsize=(10,7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f837b0fcf90>
```

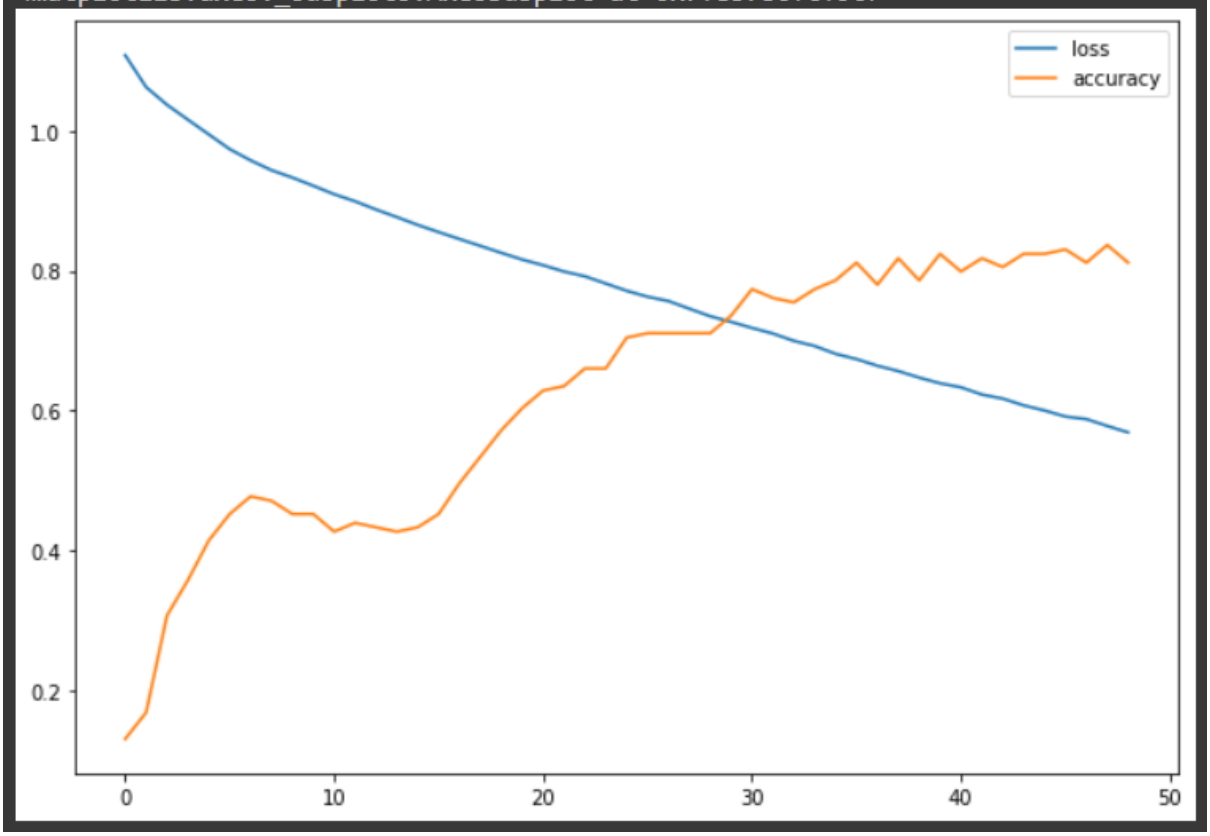

Figure 10 Graph model for loss function and accuracy

```
y_probs = model_1.predict(X_test)
y_preds = y_probs.argmax(axis=1)
y_preds
```

```
array([1, 1, 2, 0, 0, 0, 2, 2, 2, 0, 0, 2, 0, 0, 1, 0, 2, 2, 2, 2, 0, 2,
       2, 2, 1, 2, 0, 1, 1, 2, 2, 2, 2, 2, 0, 0, 2, 2, 1, 2])
```

Figure 11 Probability of Prediction

Question 2

1. Introduction

It is proposed that a Fuzzy Logic control system be implemented in a dishwasher appliance to acquire a processing time that is dependent on the number of plates and the level of dirtiness on those plates. The conventional method required human intervention at regular intervals to make decisions regarding the appropriate amount of processing time appropriate for the various dish conditions. To put it another way, the ability to perform analysis and make decisions has been programmed into the machine, which confers much increased intelligence on the latter.

2. The Tipping Problem

Antecedents (Inputs)

Number of dishes

- Universe: How many numbers of plate on a scale of 0 to 50?
- Fuzzy set: LOW, AVERAGE, HIGH, EXTRA HIGH

Level of dirtiness

- Universe: How dirty are the dishes on a scale of 0 to 100 percent
- Fuzzy set: LIGHT, MEDIUM, HEAVY, VEY HEAVY

Consequents (Outputs)

Wash Time

- Universe: How long is the wash time going to be on a scale of 0 to 90 minutes
- Fuzzy set: SHORT, AVERAGE, LONG

3. Implementation

The dataset is downloaded from the UC Irvine machine learning repository and loaded into Google Colab for analysis. Implementation using python code is shown below.

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

Figure 12 Imported required library

```
dish = ctrl.Antecedent(np.arange(0, 50, 1), 'dish')
dirty = ctrl.Antecedent(np.arange(0, 100, 1), 'dirty')
washTime = ctrl.Consequent(np.arange(0, 90, 1), 'washTime')
```

Figure 13 Universe variables

4. Membership functions design

```
dish['LOW'] = fuzz.trimf(dish.universe, [0, 0, 13])
dish['AVERAGE'] = fuzz.trimf(dish.universe, [0, 13, 25])
dish['HIGH'] = fuzz.trimf(dish.universe, [13, 25, 40])
dish['EXTRA HIGH'] = fuzz.trimf(dish.universe, [40, 50, 50])

dirty['LIGHT'] = fuzz.trimf(dirty.universe, [0, 0, 25])
dirty['MEDIUM'] = fuzz.trimf(dirty.universe, [0, 25, 50])
dirty['HEAVY'] = fuzz.trimf(dirty.universe, [25, 50, 75])
dirty['VERY HEAVY'] = fuzz.trimf(dirty.universe, [75, 100, 100])

washTime['SHORT'] = fuzz.trimf(washTime.universe, [0, 0, 30])
washTime['AVERAGE'] = fuzz.trimf(washTime.universe, [0, 30, 60])
washTime['LONG'] = fuzz.trimf(washTime.universe, [60, 90, 90])
```

Figure 14 Membership function

```
dish.view()
```

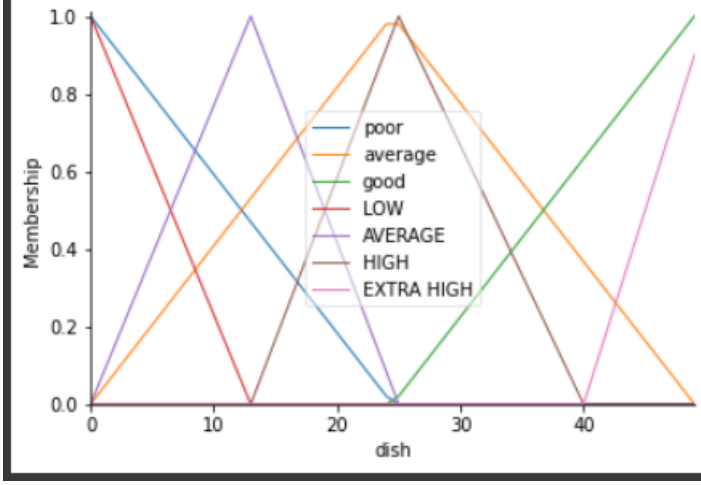

Figure 15 Graphical view for dish membership

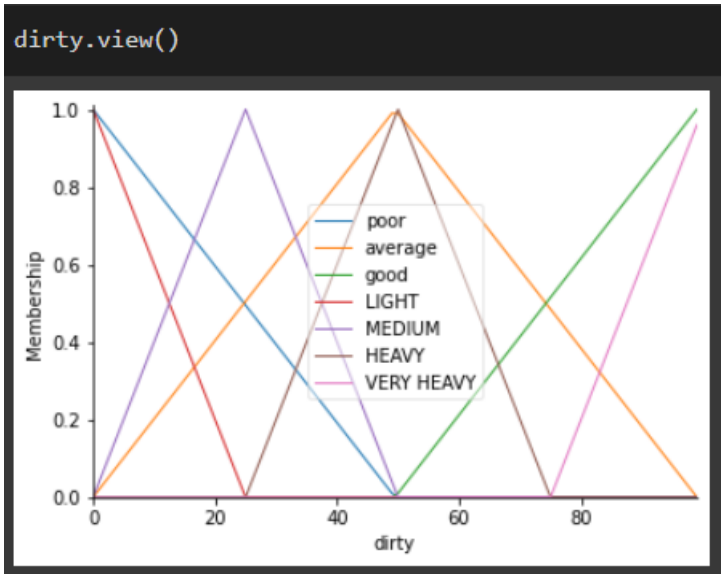

Figure 16 Graphical view for dirty membership

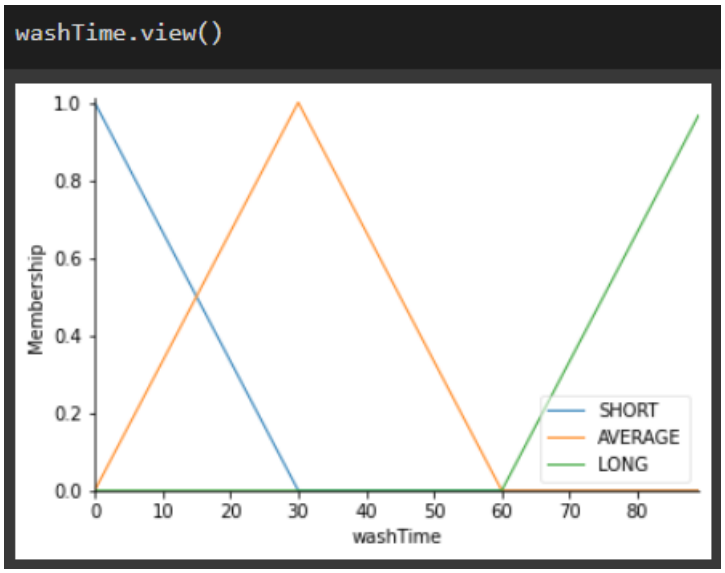

Figure 17 Graphical view for washTime membership

5. Fuzzy rules list

10 rules are :

1. If the number of dishes are *LOW* **OR** dirty level is *LIGHT*, then wash time will be *SHORT*
2. If the number of dishes are *AVERAGE* **OR** dirty level is *MEDIUM* then then the wash time will be *AVERAGE*
3. If the number of dishes are *HIGH* **OR** dirty level is *HEAVY*, then the wash time will be *LONG*
4. If the number of dishes are *EXTRA HIGH* **OR** dirty level is *VERY HEAVY*, then the wash time will be *LONG*
5. If the number of dishes are LOW **AND** dirty level is *MEDIUM*, then the wash time will be *SHORT*
6. If the number of dishes are *AVERAGE* **AND** dirty level is *HEAVY*, then the wash time will be *AVERAGE*
7. If the number of dishes are *HIGH* **AND** dirty level is *VERY HEAVY*, then the wash time will be *LONG*
8. If the number of dishes are *EXTRA HIGH* **AND** dirty level is *MEDIUM*, then the wash time will be *LONG*
9. If the number of dishes are *AVERAGE* **AND** dirty level is very *LIGHT*, then the wash time will be *SHORT*
10. If the number of dishes are *LOW* **AND** dirty level is *VERY HEAVY*, then the wash time will be *AVERAGE*

Figure 18 TEN fuzzy rules with mixture of AND & OR operators

6. Simulation input and output produced by FIS

```
rule1 = ctrl.Rule(dish['LOW'] | dirty['LIGHT'], washTime['SHORT'])
rule2 = ctrl.Rule(dish['AVERAGE'] | dirty['MEDIUM'], washTime['AVERAGE'])
rule3 = ctrl.Rule(dish['HIGH'] | dirty['HEAVY'], washTime['LONG'])
rule4 = ctrl.Rule(dish['EXTRA HIGH'] | dirty['VERY HEAVY'], washTime['LONG'])
rule5 = ctrl.Rule(dish['LOW'] & dirty['MEDIUM'], washTime['SHORT'])
rule6 = ctrl.Rule(dish['AVERAGE'] & dirty['HEAVY'], washTime['AVERAGE'])
rule7 = ctrl.Rule(dish['HIGH'] & dirty['VERY HEAVY'], washTime['LONG'])
rule8 = ctrl.Rule(dish['EXTRA HIGH'] & dirty['MEDIUM'], washTime['LONG'])
rule9 = ctrl.Rule(dish['AVERAGE'] & dirty['LIGHT'], washTime['SHORT'])
rule10 = ctrl.Rule(dish['LOW'] & dirty['HEAVY'], washTime['AVERAGE'])
```

Figure 19 Fuzzy rules implemented

```
rule1.view()
```

```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f19e0982e90>)
```

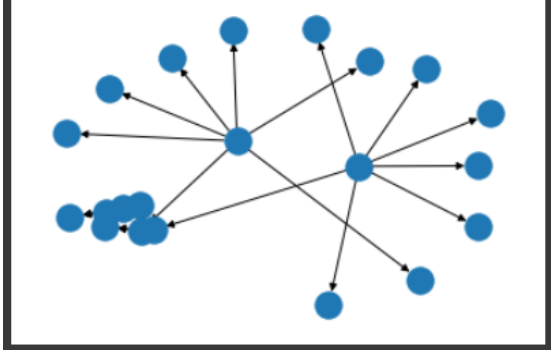

Figure 20 View of rule 1 using OR operator

```
rule10.view()
```

```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f19e088b790>)
```

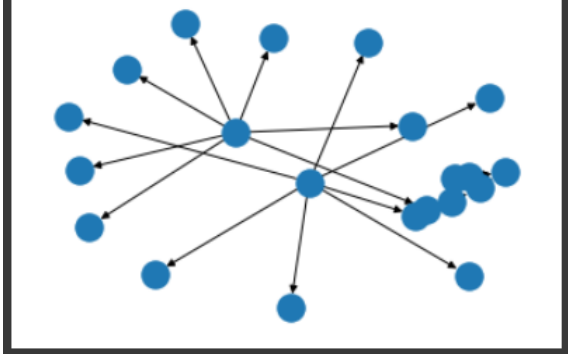

Figure 21 View of rule 10 using AND operator

Control System

```
washTime_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10])
```

Control system for simulation

```
[17] WashTime = ctrl.ControlSystemSimulation(washTime_ctrl)
```

Figure 22 All 10 fuzzy logics are implemented into control system and simulation

Number of plates (10) **LOW** and dirty level (70%) **HEAVY** then washing time is (36.30 minutes) **SHORT**

```
[60] WashTime.input['dish'] = 10
     WashTime.input['dirty'] = 70
```

```
[61] time_taken = WashTime.compute()
```

Result can be viewed

```
[62] print(WashTime.output['washTime'])
     washTime.view(sim=WashTime)
```

```
36.30047586285973
```

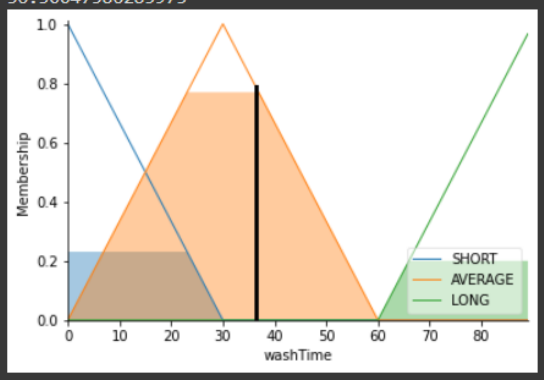

Figure 23 washTime calculated for LOW no. of plates and HEAVY dirty level

Number of plates (17) **AVERAGE** and dirty level (62%) **HEAVY** then washing time is (44.02 minutes) **AVERAGE**

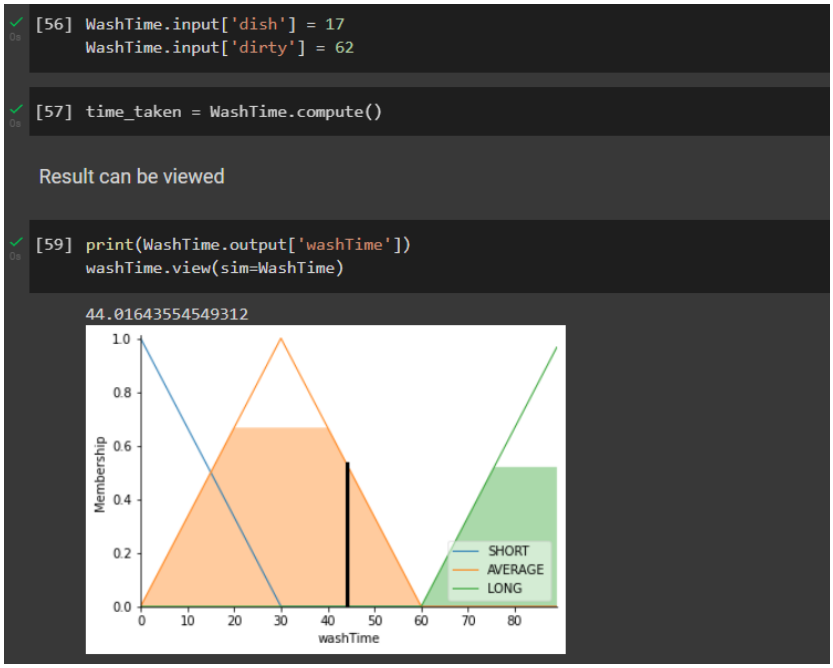

Figure 24 washTime calculated for AVERAGE no. of plates and HEAVY dirty level