# **Оглавление**

[**Оглавление** 3](#_Toc27997865)

[**Введение** 4](#_Toc27997866)

[**1. Описание работы нейро-нечёткой системы** 6](#_Toc27997867)

[**1.1. Общие положения** 6](#_Toc27997868)

[**1.2. Структура нечёткой продукционной модели** 7](#_Toc27997869)

[**1.3. Описание алгоритмов** 12](#_Toc27997870)

[**Заключение** 23](#_Toc27997871)

[**Список используемых источников** 24](#_Toc27997872)

[**Приложение. Полный исходный код** 25](#_Toc27997873)

# **Введение**

Целью курсовой работы является разработка интеллектуальной системы оценки рисков информационной безопасности.

Для реализации поставленной задачи было решено использовать адаптивную систему на основе нечёткой логики.

Средой разработки была выбрана IDE PyCharm. Для работы с данными была выбрана библиотека numpy, а для разработки системы с нечётким выводом библиотека skfuzzy.

Нечеткая логика предназначена для формализации человеческих способностей к неточным или приближенным рассуждениям, которые позволяют более адекватно описывать ситуации с неопределенностью. Классическая логика по своей сути игнорирует проблему неопределенности, поскольку все высказывания и рассуждения в формальных логических системах могут иметь только значение "истина" (И, 1) или значение "ложь" (Л, 0). В отличие от этого в нечеткой логике истинность рассуждений оценивается в некоторой степени, которая может принимать и другие отличные {И, Л} значения.

Системы нечеткого вывода являются частным случаем нечетких систем или систем нечетких правил, в которых условия и заключения правил формулируются в форме нечетких лингвистических высказываний относительно тех или иных лингвистических переменных.

Общая схема системы нечёткого вывода показана на рис. 1

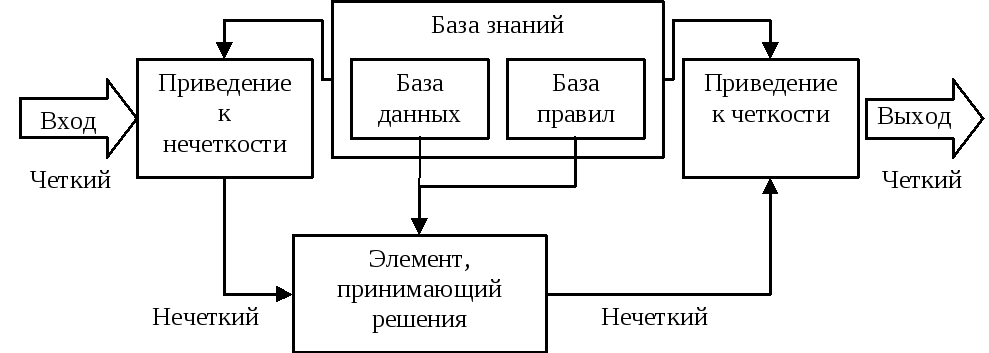


Рис. 1 Общая схема системы нечёткого вывода

В контексте нечеткой логики под приведением к нечёткости (фаззификацией) понимается не только отдельный этап нечеткого вывода, но и собственно процесс или процедура нахождения значений функции принадлежности нечетких множеств (термов) на основе обычных (не нечетких) исходных данных. Целью этапа фаззификации является установление соответствия между конкретным, обычно численным, значением отдельной входной переменной системы нечеткого вывода и значением функции принадлежности соответствующего ей терма входной лингвистической переменной.

Приведение к чёткости (дефаззификация) в системах нечеткого вывода представляет собой процедуру или процесс нахождения обычного (не нечеткого) значения для каждой из выходных лингвистических переменных. Цель дефаззификации заключается в том, чтобы, используя результаты аккумуляции всех выходных лингвистических переменных, получить обычное (не нечеткое) значение каждой из выходных переменных.

# **1. Описание работы нейро-нечёткой системы**

## **1.1. Общие положения**

Для достижения поставленной цели была использована адаптивная сеть на основе системы нечёткого вывода ANFIS (adaptive nero-fuzzy interference system), искусственная нейронная сеть, основанная на нечёткой системе вывода Такаги-Суено. Структура ANFIS состоит из пяти слоёв. Первый слой принимает входные значения и определяет функции принадлежности. Обычно это называется слоём фаззификации. Второй слой отвечает за правила, по которым распределяются входные переменные. Роль третьего слоя состоит в нормализации вычисленных значений. Четвёртый слой формирует значение выходной переменной. Последний, пятый слой выполняет дефаззификацию.

Пример ANFIS с двумя входами и двумя правилами (рис. 2):

ЕСЛИ P11(x1) И P12(x2) ТО f1(x1, x2)

ЕСЛИ P21(x1) И P22(x2) ТО f2(x1, x2)

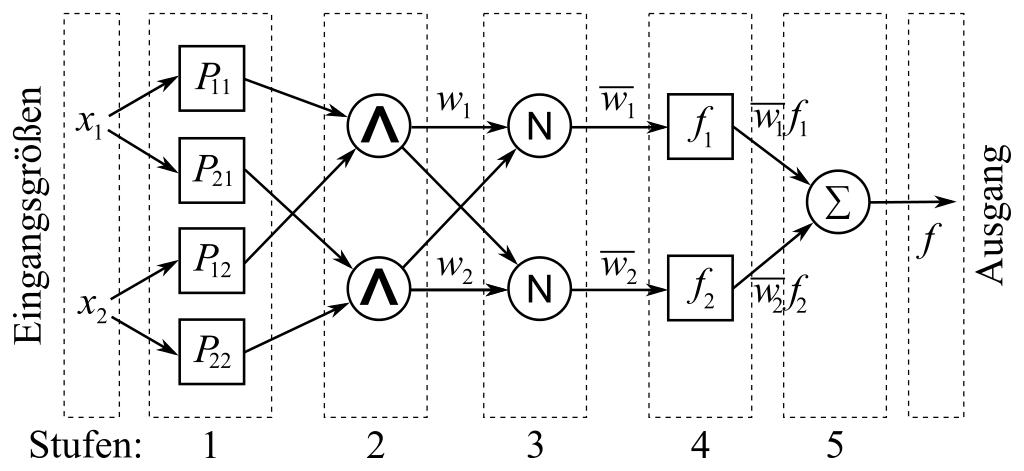


Рис. 2 Пример работы ANFIS

# **1.2. Структура нечёткой продукционной модели**

Для определения риска информационной безопасности организации были взяты семь входных лингвистических переменных (табл. 1), четыре выходных лингвистических переменных (табл. 2) и четыре базы правил (табл. 3). При формировании входных лингвистических переменных могут быть использованы следующие терм-множества, которые определяют уровни факторов:

- Т2 = {Низкий (Н); Высокий (В)};

- T3 = {Низкий (Н); Средний (С); Высокий (В)};

- T4 = {Очень Низкий (ОчН); Низкий (Н); Средний (С); Высокий (В)};

- T5 = {Очень Низкий (ОчН); Низкий (Н); Средний (С); Высокий (В); Очень Высокий (ОчВ)}.

|  |  |  |
| --- | --- | --- |
| Обозначение | Наименование лингвистической переменной | Вид терм-множества и интерпретация уровней факторов |
| Х1 | Программно-аппаратный уровень защиты | Т3. Н – удовлетворительная, для обеспечения начального уровня защиты;  С – достаточная, для базовой информационной защиты;  В – полностью соответствует уровню конфиденциальности информации |
| Х2 | Уровень организационной защиты | Т3. Н – слабое планирование и отсутствие мониторинга уязвимостей;  С – планирование и мониторинг уязвимостей проводятся нерегулярно;  В – своевременное планирование и мониторинг уязвимостей |
| Х3 | Уровень правовой защиты | Т3. Н – обрывочная и неполная документация;  С – документация имеется, но недостаточно детальная;  В – документация полная и синхронизированная |
| Х4 | Мотивация источника угроз (ИУ) | Т3. Н – редкое проявление заинтересованности;  С – вполне может заинтересовать;  В – скорее всего, заинтересуется |
| Х5 | Возможности источника угроз (ИУ) | Т3. Н – незначительный уровень оснащенности ИУ;  С – средний уровень оснащенности;  В – достаточно высокий уровень оснащенности |
| Х6 | Рыночная ценность информационного ресурса (ИР) | Т3. Н – ИР обладает незначительной ценностью;  С – ИР представляет коммерческую тайну;  В – высококонфиденциальные данные |
| Х7 | Объем данных информационного ресурса (ИР) организации | Т3. Н – меньшая часть;  С – половина ИР;  В – большая часть |

Табл. 1 Факторы риска ИБ организации

|  |  |  |
| --- | --- | --- |
| Обозначение | Наименование лингвистической переменной | Примечание |
| Y1 | Риск снижения эффективности защиты | Характеризует потенциальную возможность снижения / увеличения эффективности защиты по отношению к требуемой эффективности для конкретного предприятия |
| Y2 | Риск возникновения потенциальных угроз | Характеризует возможность возникновения потенциальных угроз для предприятия |
| Y3 | Риск материального ущерба | Характеризует возможность возникновения материального ущерба для предприятия при нарушениях параметров информационной безопасности предприятия |
| Y4 | Риск ИБ организации | Интегральный риск, характеризующий обеспечение информационной безопасности  предприятия |

Табл. 2 Показатели риска ИБ организации

|  |  |  |
| --- | --- | --- |
| Обозначение правила | Антецедент | Консеквент |
| П1.1 | (х1 = Н ˄ х2 = Н ˄ х3 = Н) ∨ (х1 = С ˄ х2 = Н ˄ х3 = Н) ∨ (х1 = Н ˄ х2 = С ˄ х3 =Н) | Y1 = ОчВОР |
| П1.2 | (х1 = В ˄ х2 = Н ˄ х3 = Н) ∨ (х1 = С ˄ х2 = С ˄ х3 = Н) ∨ (х1 = Н ˄ х2 = В ˄ х3 = Н) ∨ (х1 = С ˄ х2 = В ˄ х3 = Н) ∨ (х1 = Н ˄ х2 = Н ˄ х3 = С) ∨ (х1 = Н ˄ х2 = С ˄ х3 = С) ∨ (х1 = Н ˄ х2 = В ˄ х3 = С) ∨ (х1 = Н ˄ х2 = Н ˄ х3 = В) | Y1 = ВОР |
| П1.3 | (х1 = В ˄ х2 = С ˄ х3 = Н) ∨ (х1 = В ˄ х2 = В ˄ х3 = Н) ∨ (х1 = С ˄ х2 = Н ˄ х3 = С) ∨ (х1 = В ˄ х2 = Н ˄ х3 = С) ∨ (х1 = С ˄ х2 = С ˄ х3 = С) ∨ (х1 = С ˄ х2 = В ˄ х3 = С) ∨ (х1 = С ˄ х2 = Н ˄ х3 = В) ∨ (х1 = Н ˄ х2 = С ˄ х3 = В) ∨ (х1 = С ˄ х2 = С ˄ х3 = В) ∨ (х1 = Н ˄ х2 = В ˄ х3 = В) | Y1 = СОР |
| П1.4 | (х1 = В ˄ х2 = С ˄ х3 = С) ∨ (х1 = В ˄ х2 = В ˄ х3 = С) ∨ (х1 = В ˄ х2 = Н ˄ х3 = В) ∨ (х1 = В ˄ х2 = С ˄ х3 = В) ∨ (х1 = С ˄ х2 = В ˄ х3 = В) | Y1 = НОР |
| П1.5 | х1 = В ˄ х2 = В ˄ х3 = В | Y1 = ОчНОР |

Табл. 3 Нечёткие продукционные правила модели (база правил П1)

При формировании выходных лингвистических переменных могут быть использованы следующее терм-множества, которые определяют показатели риска:

- Т1 = {Низкая очевидность риска (НОР); Средняя очевидность риска (CОР); Высокая очевидность риска (ВОР)};

- Т2 = {Очень низкая очевидность риска (ОчНОР); Низкая очевидность риска (НОР); Средняя очевидность риска (СОР); Высокая очевидность риска (ВОР); Очень высокая очевидность риска (ОчВОР)}.

Структура нечеткой продукционной модели оценки рисков ИБ организации приведена на рисунке 3.

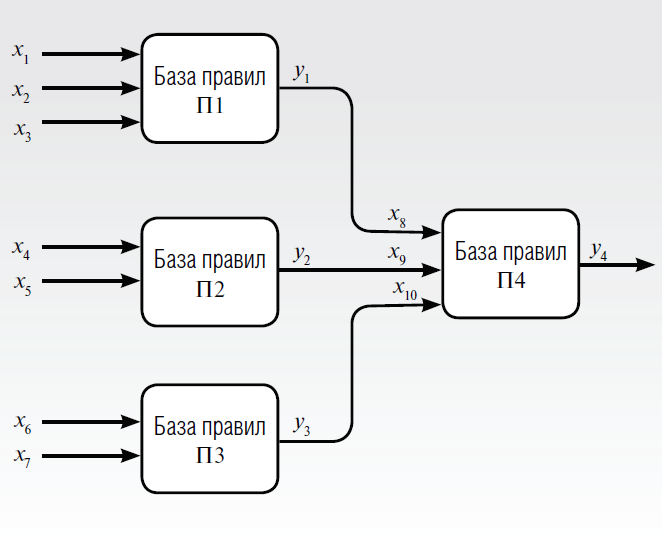


Рис. 3 Структура нечёткой продукционной модели

## **1.3. Описание алгоритмов**

Для выполнения первого этапа (первого слоя нейронов) необходимо перевести все входные параметры, которые заданы в виде вещественных чисел в нечеткий вид. Для данной цели были созданы 3 функции для каждого из выходных параметров:

1. **def** difuz\_y2(arr):
2. x1 = np.arange(0, 1.001, 0.001)
3. x2 = np.arange(0, 1.001, 0.001)
4. y1  = np.arange(0, 1.001, 0.001)

Данный фрагмент кода задает входные и выходные критерии для показателя «Риск снижения эффективности защиты» далее для каждого из них задаем функцию принадлежности, при этом превращая вещественное число в нечеткое множество, для этого разбиваем его на 3 параметра {low, middle, high}:

1. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
2. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
3. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
4. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
5. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
6. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
7. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
8. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
9. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
10. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
11. y1\_gd= fuzz.trimf(y1, [0.75, 1, 1])

Этот фрагмент кода отображает функции принадлежностей, которые выглядят следующим образом, например для параметра «Возможности источника угроз» (рис. 4):

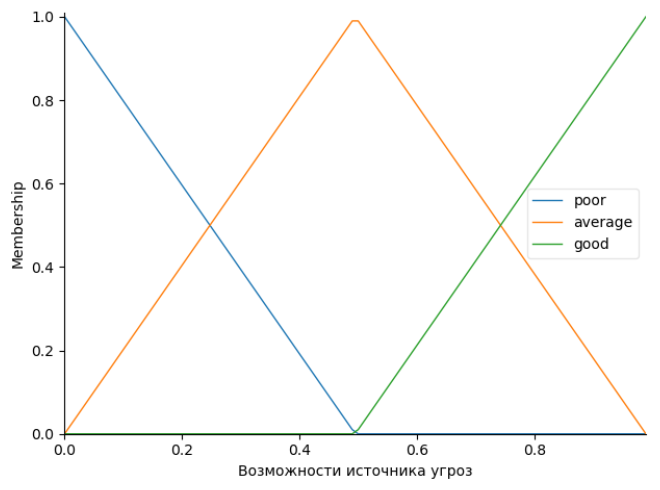


Рис. 4 Функции принадлежностей

1. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
2. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
3. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])

Метод interp\_membership (библиотека skfuzzy) осуществляет перевод вещественного числа в значение функции принадлежности, в элемент нечеткого множества при помощи t-функции принадлежности, которая описывается следующим образом:

Для второго слоя нейронов необходимо обозначить базу правил. Каждое правило представлено в виде ЕСЛИ x1 – это A’ И x2 – это B’, ТО y1 – это C’. В нечеткой логике операция И является операцией min(q(x11), q(x12)), где q – это значения функций принадлежностей, соответствующих им параметров, в точках x11 и x12. Операция ИЛИ представлена в виде max(q(x11), q(x12)).

1. active\_rule1 = np.fmin(x1\_level\_lo, x2\_level\_lo)
2. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
4. active\_rule2 = np.fmax(np.fmin(x1\_level\_lo, x2\_level\_md), np.fmin(x1\_level\_md, x2\_level\_lo))
5. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
7. active\_rule3 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_md), np.fmax(np.fmin(x1\_level\_hi, x2\_level\_lo),
8. np.fmin(x1\_level\_lo, x2\_level\_hi)))
9. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
11. active\_rule4 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_hi), np.fmin(x1\_level\_hi, x2\_level\_md))
12. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
14. active\_rule5 = np.fmin(x1\_level\_hi, x2\_level\_hi)
15. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)

Для каждого из элементов нечеткого множества y ({Очень Низкий; Низкий; Средний; Высокий; Очень Высокий}) задается свое правило в базе правил.

Дефаззификация производится методом центра масс, который описывается формулой:

Где μ′ - функция принадлежности кривой, полученной после выполнения всех правил. В нашем проекте данная опция была реализована методом skfuzz.defuzz():

1. aggregated = np.fmax(y1\_activation\_lo, np.fmax(np.fmax(y1\_activation\_md, y1\_activation\_ave),(np.fmax(y1\_activation\_dec, y1\_activation\_gd))))
2. y2\_defuz = fuzz.defuzz(y1, aggregated, 'centroid')

Наглядно этот метод изображен на следующих графиках:

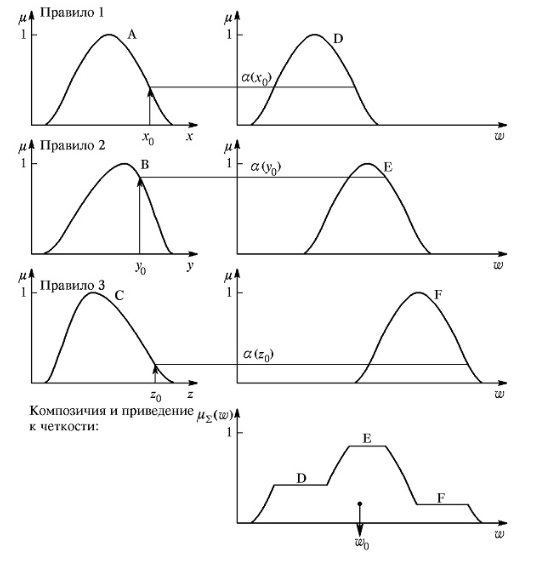


Рис. 5 Метод центра масс

Дефаззификация при помощи программных методов была использована в качестве способа нахождения идеальных ответов для последующего обучения сети. Для большей достоверности полученных ответов был также использован другой способ их получения при помощи более простых средств, находящихся в библиотеке skfuzzy:

1. **def** ideal\_stroitel\_y1(arr):
2. x1 = ctrl.Antecedent(np.arange(0, 1, 0.01), 'Программно-аппаратный уровень защиты')
3. x2 = ctrl.Antecedent(np.arange(0, 1, 0.01), 'Уровень организационной защиты')
4. x3 = ctrl.Antecedent(np.arange(0, 1, 0.01), 'Уровень правовой защиты')
5. y1 = ctrl.Consequent(np.arange(0, 1, 0.01), 'Риск снижения эффективности защиты')
7. x1.automf(3)
8. x2.automf(3)
9. x3.automf(3)
10. y1.automf(5)
11. rule1 = ctrl.Rule((x1['poor'] & x2['poor'] & x3['poor']) | (x1['average'] & x2['poor'] & x3['poor']) | (
12. x1['poor'] & x2['average'] & x3['poor']), y1['poor'] % 0.9)
13. …
14. …
15. …
16. security\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])
17. security\_es = ctrl.ControlSystemSimulation(security\_ctrl)
19. security\_es.input['Программно-аппаратный уровень защиты'] = arr[0]
20. security es.input['Уровень организационной защиты'] = arr[1]
21. security\_es.input['Уровень правовой защиты'] = arr[2]
23. security\_es.compute()

В обоих случаях выдается одинаковый ответ, например на ввод чисел [0.4, 0.9, 0.9] оба способа выдают ответ:

Риск снижения эффективности защиты

0.6102941176470577

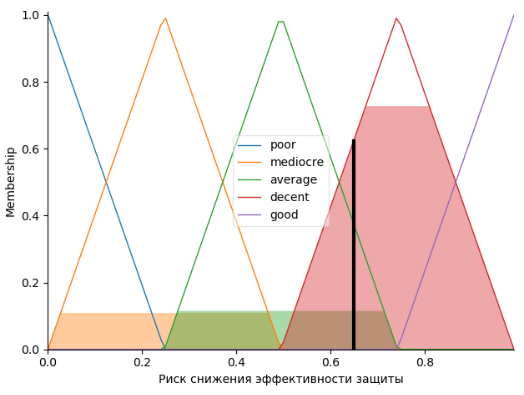


Рис. 6 График функции принадлежности

Аналогичным образом реализованы и другие функции для выходных параметров «Риск возникновения потенциальных угроз»и «Риск материального ущерба», функция для вывода «Риск ИБ организации» принимает в качестве входного параметра массив из результатов y1, y2 и y3, в остальном она работает аналогично предыдущим.

После слоя, содержащего базы правил мы получаем нечеткие переменные выхода, которые представлены в виде значений функций принадлежностей для каждого из правил. Для продолжения необходимо нормализовать наши полученные данные.

В коде это выполнено следующим образом:

1. w1 = active\_rule1 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
2. w2 = active\_rule2 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
3. w3 = active\_rule3 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
4. w4 = active\_rule4 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
5. w5 = active\_rule5 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
7. w = [w1, w2, w3, w4, w5]

Оставшимся синопсам нашей цепи мы присваиваем веса и далее реализуем алгоритм обратного разпространения ошибки для многослойного персептрона, чтобы обучить нашу нейросеть. Веса задавались для кадого из синапсов в диапазоне [0, 1] рандомным образом в качестве значений для входных синапсов были взяты нормированые значения они же степень выполнения нечеткого правила в формуле.

Данные заданы в виде массивов:

1. input\_arr\_y3 = [[round(random.random(), 3) **for** j **in** range(2)] **for** i **in** range(200)]
2. input\_arr\_y2 = [[round(random.random(), 3) **for** j **in** range(2)] **for** i **in** range(200)]
3. input\_arr\_y1 = [[round(random.random(), 3) **for** j **in** range(3)] **for** i **in** range(200)]
4. ideal\_value\_y3 = [round(difuz\_y3(input\_arr\_y3[i]), 3) **for** i **in** range(200)]
5. ideal\_value\_y2 = [round(difuz\_y2(input\_arr\_y2[i]), 3) **for** i **in** range(200)]
6. ideal\_value\_y1 = [round(difuz\_y1(input\_arr\_y1[i]), 3) **for** i **in** range(200)]

Далее входные данные проходят первые слои нейронной сети, такие как инициализация, фаззификация и совпадение с базой правил. Алгоритмы этих слоев были представлены выше, для них выполнены отдельные функции dlya2 и dlya3:

1. inputs = [(dlya\_3(input\_arr\_y1[i])) **for** i **in** range(200)]
2. inputs2 = [(dlya2(input\_arr\_y2[i])) **for** i **in** range(200)]
3. inputs3 = [(dlya2(input\_arr\_y3[i])) **for** i **in** range(200)]

Далее для каждого из синапсов представлены веса, в качестве функции активатора используется сигмоид:

1. **def** sigmoid(x):
2. **return** 1/(1 + np.exp(-x))
3. …
4. …
5. S1 = inputs[i][0]\* weight1
6. S2 = inputs[i][1] \* weight2
7. S3 = inputs[i][2] \* weight3
8. S4 = inputs[i][3] \* weight4
9. S5 = inputs[i][4] \* weight5
10. S11 = S1+S2+S3+S4+S5
11. …
12. …
13. F1 = sigmoid(S11)
14. F2 = sigmoid(S21)
15. F3 = sigmoid(S31)
16. S4 = F1\*weight16+F2\*weight17+F3\*weight18
17. F4 = sigmoid(S4)
18. D = ideal\_value\_y4[i] - F4
19. D1 = D\*weight16
20. D2 = D\*weight17
21. D3 = D\*weight18
22. weight1 = weight1 + D1\*F1\*(1-F1)\*0.01\*inputs[i][0]
23. weight2 = weight2 + D1\*F1\*(1-F1)\*0.01\*inputs[i][1]
24. weight3 = weight3 + D1 \* F1 \* (1 - F1) \* 0.01\*inputs[i][2]
25. weight4 = weight4 + D1 \* F1 \* (1 - F1) \* 0.01\*inputs[i][3]
26. weight5 = weight5 + D1 \* F1 \* (1 - F1) \* 0.01\*inputs[i][4]
27. weight6 = weight6 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][0]
28. weight7 = weight7 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][1]
29. weight8 = weight8 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][2]
30. weight9 = weight9 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][3]
31. weight10 = weight10 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][4]
32. weight11 = weight11 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][0]
33. weight12 = weight12 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][1]
34. weight13 = weight13 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][2]
35. weight14 = weight14 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][3]
36. weight15 = weight15 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][4]
37. weight16 = weight16 + D \* F4 \* (1 - F4) \* 0.01\*F1
38. weight17 = weight17 + D \* F4 \* (1 - F4) \* 0.01\*F2
39. weight18 = weight18 + D \* F4 \* (1 - F4) \* 0.01\*F3

Ошибка вычисляется по формуле: D = ideal – e, где ideal - это идеальное значение параметра, e – полученное значение параметра. После расчета ошибки, необходимо высчитать изменения весов всех синапсов. Для этого используется формула:

W’i23 = wi23 + d\*Fa’(S7)\*f\*a

Где W’i23 – это измененный вес, d- ошибка, Fa – производная из сигмоидальной функции, которая заменяется формулой: Fa\*(1-Fa).

В идеальной выборке было использовано 200 наборов начальных данных и 200 идеальных выходов. В последствии после работы сети (200 эпох) были получены ответы:

При заданных начальных данных:

«Программно-аппаратный уровень защиты» – 0.6

«Уровень организационной защиты» - 0.7

«Уровень правовой защиты» – 0.4

«Мотивация источника угроз» – 0.4

«Возможности источника угроз» – 0.7

«Рыночная ценность информационного ресурса» – 0.2

«Объем данных информационного ресурса» – 0.25

«Риск ИБ организации» (идеальный ответ):0.441.

Ответ, полученный в результате выполнения нейронной сети: 0.4472185554997701.

Блок схема, отображающая работу нейро-нечеткой сети, изображена на рис. 7

X3

X2

X1

W’ = W + d\*F’(S)\*xi\*a

F = sigmoid(S)

Обучение методом обратного распространения ошибки

Нормирование, полученных нечетких результатов

База правил

Машина нечеткого логического вывода

Фаззификация входных данных

Ydefuz

Рис. 7 Блок-схема, показывающая общую схему работы нейро-нечёткой цепи

# **Заключение**

В данной курсовой работе была продемонстрирована реализация интеллектуальной системы оценки рисков информационной безопасности в виде нейронной сети с нечётким выводом. Данная система получает на вход 7 параметров, и, основываясь на них, определяет общую безопасность информационной сети организации. Полученные результаты могут помочь специалисту по ИБ провести необходимые действия по совершенствованию безопасности информационной сети предприятия во избежание убыток при различных атаках.

# **Список используемых источников**

1. Белов А.А., Гвоздева Т.В. «Основы теории нечёткости», 2001

2. Глушенко С.А. «Адаптивная нейро-нечёткая система оценки рисков информационной безопасности организации», 2017

3. «Нечеткая логика — математические основы» [Электронный ресурс], URL: https://basegroup.ru/community/articles/fuzzylogic-math

4. «Нейронные сети для начинающих» [Электронный ресурс], URL: https://habr.com/ru/post/312450/

5. Доусон М. «Программируем на Python», Питер СПБ, 2019

# **Приложение. Полный исходный код**

1. **import** numpy as np
2. **import** skfuzzy as fuzz
3. **import** random as random
5. **def** sigmoid(x):
6. **return** 1/(1 + np.exp(-x))
8. **def** difuz\_y2(arr):
9. x1 = np.arange(0, 1.001, 0.001)
10. x2 = np.arange(0, 1.001, 0.001)
11. y1  = np.arange(0, 1.001, 0.001)
13. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
14. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
15. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
16. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
17. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
18. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
19. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
20. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
21. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
22. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
23. y1\_gd= fuzz.trimf(y1, [0.75, 1, 1])
25. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
26. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
27. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])
29. x2\_level\_lo = fuzz.interp\_membership(x2, x2\_lo, arr[1])
30. x2\_level\_md = fuzz.interp\_membership(x2, x2\_md, arr[1])
31. x2\_level\_hi = fuzz.interp\_membership(x2, x2\_hi, arr[1])

34. active\_rule1 = np.fmin(x1\_level\_lo, x2\_level\_lo)
35. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
37. active\_rule2 = np.fmax(np.fmin(x1\_level\_lo, x2\_level\_md), np.fmin(x1\_level\_md, x2\_level\_lo))
38. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
40. active\_rule3 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_md), np.fmax(np.fmin(x1\_level\_hi, x2\_level\_lo),
41. np.fmin(x1\_level\_lo, x2\_level\_hi)))
42. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
44. active\_rule4 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_hi), np.fmin(x1\_level\_hi, x2\_level\_md))
45. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
47. active\_rule5 = np.fmin(x1\_level\_hi, x2\_level\_hi)
48. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)
50. aggregated = np.fmax(y1\_activation\_lo, np.fmax(np.fmax(y1\_activation\_md, y1\_activation\_ave),(np.fmax(y1\_activation\_dec, y1\_activation\_gd))))
51. y2\_defuz = fuzz.defuzz(y1, aggregated, 'centroid')
53. **return** y2\_defuz


57. **def** difuz\_y3(arr):
58. x1 = np.arange(0, 1.001, 0.001)
59. x2 = np.arange(0, 1.001, 0.001)
60. y1 = np.arange(0, 1.001, 0.001)
62. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
63. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
64. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
65. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
66. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
67. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
68. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
69. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
70. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
71. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
72. y1\_gd = fuzz.trimf(y1, [0.75, 1, 1])
74. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
75. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
76. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])
78. x2\_level\_lo = fuzz.interp\_membership(x2, x2\_lo, arr[1])
79. x2\_level\_md = fuzz.interp\_membership(x2, x2\_md, arr[1])
80. x2\_level\_hi = fuzz.interp\_membership(x2, x2\_hi, arr[1])
82. active\_rule1 = np.fmin(x1\_level\_lo, x2\_level\_lo)
83. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
85. active\_rule2 = np.fmax(np.fmin(x1\_level\_lo, x2\_level\_md), np.fmin(x1\_level\_md, x2\_level\_lo))
86. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
88. active\_rule3 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_md), np.fmax(np.fmin(x1\_level\_hi, x2\_level\_lo),
89. np.fmin(x1\_level\_lo, x2\_level\_hi)))
90. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
92. active\_rule4 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_hi), np.fmin(x1\_level\_hi, x2\_level\_md))
93. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
95. active\_rule5 = np.fmin(x1\_level\_hi, x2\_level\_hi)
96. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)
98. aggregated = np.fmax(y1\_activation\_lo, np.fmax(np.fmax(y1\_activation\_md, y1\_activation\_ave),
99. (np.fmax(y1\_activation\_dec, y1\_activation\_gd))))
101. y3\_defuz = fuzz.defuzz(y1, aggregated, 'centroid')
103. **return** y3\_defuz
105. **def** dlya2(arr):
106. x1 = np.arange(0, 1.001, 0.001)
107. x2 = np.arange(0, 1.001, 0.001)
108. y1 = np.arange(0, 1.001, 0.001)
110. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
111. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
112. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
113. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
114. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
115. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
116. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
117. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
118. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
119. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
120. y1\_gd = fuzz.trimf(y1, [0.75, 1, 1])
122. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
123. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
124. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])
126. x2\_level\_lo = fuzz.interp\_membership(x2, x2\_lo, arr[1])
127. x2\_level\_md = fuzz.interp\_membership(x2, x2\_md, arr[1])
128. x2\_level\_hi = fuzz.interp\_membership(x2, x2\_hi, arr[1])
130. active\_rule1 = np.fmin(x1\_level\_lo, x2\_level\_lo)
131. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
133. active\_rule2 = np.fmax(np.fmin(x1\_level\_lo, x2\_level\_md), np.fmin(x1\_level\_md, x2\_level\_lo))
134. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
136. active\_rule3 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_md), np.fmax(np.fmin(x1\_level\_hi, x2\_level\_lo),
137. np.fmin(x1\_level\_lo, x2\_level\_hi)))
138. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
140. active\_rule4 = np.fmax(np.fmin(x1\_level\_md, x2\_level\_hi), np.fmin(x1\_level\_hi, x2\_level\_md))
141. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
143. active\_rule5 = np.fmin(x1\_level\_hi, x2\_level\_hi)
144. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)
146. w1 = active\_rule1 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
147. w2 = active\_rule2 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
148. w3 = active\_rule3 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
149. w4 = active\_rule4 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
150. w5 = active\_rule5 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
152. w = [w1, w2, w3, w4, w5]
153. **return** w
155. **def** difuz\_y1(arr):
156. x1 = np.arange(0, 1.001, 0.001)
157. x2 = np.arange(0, 1.001, 0.001)
158. x3 = np.arange(0, 1.001, 0.001)
159. y1 = np.arange(0, 1.001, 0.001)
161. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
162. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
163. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
164. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
165. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
166. x3\_hi = fuzz.trimf(x2, [0.5, 1, 1])
167. x3\_lo = fuzz.trimf(x2, [0, 0, 0.5])
168. x3\_md = fuzz.trimf(x2, [0, 0.5, 1])
169. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
170. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
171. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
172. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
173. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
174. y1\_gd = fuzz.trimf(y1, [0.75, 1, 1])
176. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
177. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
178. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])
180. x2\_level\_lo = fuzz.interp\_membership(x2, x2\_lo, arr[1])
181. x2\_level\_md = fuzz.interp\_membership(x2, x2\_md, arr[1])
182. x2\_level\_hi = fuzz.interp\_membership(x2, x2\_hi, arr[1])
184. x3\_level\_lo = fuzz.interp\_membership(x3, x3\_lo, arr[2])
185. x3\_level\_md = fuzz.interp\_membership(x3, x3\_md, arr[2])
186. x3\_level\_hi = fuzz.interp\_membership(x3, x3\_hi, arr[2])
188. active\_rule1 = np.fmax(np.fmax(np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_lo)),
189. np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_lo))),
190. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_lo)))
191. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
193. active\_rule2 = np.fmax(np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_lo)),
194. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_md))),
195. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_md))),
196. np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_lo)),
197. np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_lo))),
198. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_lo)))),
199. np.fmax(np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_md)),
200. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_hi))))
201. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
203. active\_rule3 = np.fmax(np.fmax(np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_lo)),
204. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_hi, x3\_level\_lo))),
205. np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_md))),
206. np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_md)),
207. np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_md))),
208. np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_md)))),
209. np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_hi)),
210. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_hi)))),
211. np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_hi)),
212. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_hi))))
213. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
215. active\_rule4 = np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_md)),
216. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_hi, x3\_level\_md))),
217. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_hi))),
218. np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_hi)),
219. np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_hi))))
220. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
222. active\_rule5 = np.fmin(x3\_level\_hi, np.fmin(x1\_level\_hi, x2\_level\_hi))
223. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)
225. aggregated = np.fmax(y1\_activation\_lo, np.fmax(np.fmax(y1\_activation\_md, y1\_activation\_ave),
226. (np.fmax(y1\_activation\_dec, y1\_activation\_gd))))
228. y1\_defuz = fuzz.defuzz(y1, aggregated, 'centroid')


232. **return** y1\_defuz
234. **def** dlya\_3(arr):
235. x1 = np.arange(0, 1.001, 0.001)
236. x2 = np.arange(0, 1.001, 0.001)
237. x3 = np.arange(0, 1.001, 0.001)
238. y1 = np.arange(0, 1.001, 0.001)
240. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
241. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
242. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
243. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
244. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
245. x3\_hi = fuzz.trimf(x2, [0.5, 1, 1])
246. x3\_lo = fuzz.trimf(x2, [0, 0, 0.5])
247. x3\_md = fuzz.trimf(x2, [0, 0.5, 1])
248. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
249. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
250. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
251. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
252. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
253. y1\_gd = fuzz.trimf(y1, [0.75, 1, 1])
255. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
256. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
257. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])
259. x2\_level\_lo = fuzz.interp\_membership(x2, x2\_lo, arr[1])
260. x2\_level\_md = fuzz.interp\_membership(x2, x2\_md, arr[1])
261. x2\_level\_hi = fuzz.interp\_membership(x2, x2\_hi, arr[1])
263. x3\_level\_lo = fuzz.interp\_membership(x3, x3\_lo, arr[2])
264. x3\_level\_md = fuzz.interp\_membership(x3, x3\_md, arr[2])
265. x3\_level\_hi = fuzz.interp\_membership(x3, x3\_hi, arr[2])
267. active\_rule1 = np.fmax(np.fmax(np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_lo)),
268. np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_lo))),
269. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_lo)))
270. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
272. active\_rule2 = np.fmax(np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_lo)),
273. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_md))),
274. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_md))),
275. np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_lo)),
276. np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_lo))),
277. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_lo)))),
278. np.fmax(np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_md)),
279. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_hi))))
280. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
282. active\_rule3 = np.fmax(np.fmax(np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_lo)),
283. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_hi, x3\_level\_lo))),
284. np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_md))),
285. np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_md)),
286. np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_md))),
287. np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_md)))),
288. np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_hi)),
289. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_hi)))),
290. np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_hi)),
291. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_hi))))
292. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
294. active\_rule4 = np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_md)),
295. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_hi, x3\_level\_md))),
296. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_hi))),
297. np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_hi)),
298. np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_hi))))
299. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
301. active\_rule5 = np.fmin(x3\_level\_hi, np.fmin(x1\_level\_hi, x2\_level\_hi))
302. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)
304. w1 = active\_rule1/(active\_rule1+active\_rule2+active\_rule3+active\_rule4+active\_rule5)
305. w2 = active\_rule2 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
306. w3 = active\_rule3 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
307. w4 = active\_rule4 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
308. w5 = active\_rule5 / (active\_rule1 + active\_rule2 + active\_rule3 + active\_rule4 + active\_rule5)
310. w = [w1, w2, w3, w4, w5]
311. **return** w
313. **def** difuz\_y4(arr):
314. x1 = np.arange(0, 1.001, 0.001)
315. x2 = np.arange(0, 1.001, 0.001)
316. x3 = np.arange(0, 1.001, 0.001)
317. y1 = np.arange(0, 1.001, 0.001)
319. x1\_lo = fuzz.trimf(x1, [0, 0, 0.5])
320. x1\_md = fuzz.trimf(x1, [0, 0.5, 1])
321. x1\_hi = fuzz.trimf(x1, [0.5, 1, 1])
322. x2\_lo = fuzz.trimf(x2, [0, 0, 0.5])
323. x2\_md = fuzz.trimf(x2, [0, 0.5, 1])
324. x3\_hi = fuzz.trimf(x2, [0.5, 1, 1])
325. x3\_lo = fuzz.trimf(x2, [0, 0, 0.5])
326. x3\_md = fuzz.trimf(x2, [0, 0.5, 1])
327. x2\_hi = fuzz.trimf(x2, [0.5, 1, 1])
328. y1\_lo = fuzz.trimf(y1, [0, 0, 0.25])
329. y1\_md = fuzz.trimf(y1, [0, 0.25, 0.5])
330. y1\_ave = fuzz.trimf(y1, [0.25, 0.5, 0.75])
331. y1\_dec = fuzz.trimf(y1, [0.5, 0.75, 1])
332. y1\_gd = fuzz.trimf(y1, [0.75, 1, 1])
334. x1\_level\_lo = fuzz.interp\_membership(x1, x1\_lo, arr[0])
335. x1\_level\_md = fuzz.interp\_membership(x1, x1\_md, arr[0])
336. x1\_level\_hi = fuzz.interp\_membership(x1, x1\_hi, arr[0])
338. x2\_level\_lo = fuzz.interp\_membership(x2, x2\_lo, arr[1])
339. x2\_level\_md = fuzz.interp\_membership(x2, x2\_md, arr[1])
340. x2\_level\_hi = fuzz.interp\_membership(x2, x2\_hi, arr[1])
342. x3\_level\_lo = fuzz.interp\_membership(x3, x3\_lo, arr[2])
343. x3\_level\_md = fuzz.interp\_membership(x3, x3\_md, arr[2])
344. x3\_level\_hi = fuzz.interp\_membership(x3, x3\_hi, arr[2])
346. active\_rule1 = np.fmax(np.fmax(np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_lo)),
347. np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_lo))),
348. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_lo)))
349. y1\_activation\_lo = np.fmin(active\_rule1, y1\_lo)
351. active\_rule2 = np.fmax(np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_lo)),
352. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_md))),
353. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_md))),
354. np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_lo)),
355. np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_lo))),
356. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_lo)))),
357. np.fmax(np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_md)),
358. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_lo, x3\_level\_hi))))
359. y1\_activation\_md = np.fmin(active\_rule2, y1\_md)
361. active\_rule3 = np.fmax(np.fmax(np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_lo)),
362. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_hi, x3\_level\_lo))),
363. np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_md))),
364. np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_md)),
365. np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_md))),
366. np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_md)))),
367. np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_lo, x3\_level\_hi)),
368. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_md, x3\_level\_hi)))),
369. np.fmax(np.fmin(x1\_level\_md, np.fmin(x2\_level\_md, x3\_level\_hi)),
370. np.fmin(x1\_level\_lo, np.fmin(x2\_level\_hi, x3\_level\_hi))))
371. y1\_activation\_ave = np.fmin(active\_rule3, y1\_ave)
373. active\_rule4 = np.fmax(np.fmax(np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_md)),
374. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_hi, x3\_level\_md))),
375. np.fmin(x1\_level\_hi, np.fmin(x2\_level\_lo, x3\_level\_hi))),
376. np.fmax(np.fmin(x1\_level\_hi, np.fmin(x2\_level\_md, x3\_level\_hi)),
377. np.fmin(x1\_level\_md, np.fmin(x2\_level\_hi, x3\_level\_hi))))
378. y1\_activation\_dec = np.fmin(active\_rule4, y1\_dec)
380. active\_rule5 = np.fmin(x3\_level\_hi, np.fmin(x1\_level\_hi, x2\_level\_hi))
381. y1\_activation\_gd = np.fmin(active\_rule5, y1\_gd)
383. aggregated = np.fmax(y1\_activation\_lo, np.fmax(np.fmax(y1\_activation\_md, y1\_activation\_ave),
384. (np.fmax(y1\_activation\_dec, y1\_activation\_gd))))
386. y4\_defuz = fuzz.defuzz(y1, aggregated, 'centroid')
388. **return** y4\_defuz

391. input\_arr\_y3 = [[round(random.random(), 3) **for** j **in** range(2)] **for** i **in** range(200)]
392. input\_arr\_y2 = [[round(random.random(), 3) **for** j **in** range(2)] **for** i **in** range(200)]
393. input\_arr\_y1 = [[round(random.random(), 3) **for** j **in** range(3)] **for** i **in** range(200)]
394. ideal\_value\_y3 = [round(difuz\_y3(input\_arr\_y3[i]), 3) **for** i **in** range(200)]
395. ideal\_value\_y2 = [round(difuz\_y2(input\_arr\_y2[i]), 3) **for** i **in** range(200)]
396. ideal\_value\_y1 = [round(difuz\_y1(input\_arr\_y1[i]), 3) **for** i **in** range(200)]
398. ideal\_value\_y4 = [round(difuz\_y4([ideal\_value\_y1[i], ideal\_value\_y2[i], ideal\_value\_y3[i]]), 3) **for** i **in** range(200)]
400. weight1 = np.random.random()
401. weight2 = np.random.random()
402. weight3 = np.random.random()
403. weight4 = np.random.random()
404. weight5 = np.random.random()
405. weight6 = np.random.random()
406. weight7 = np.random.random()
407. weight8 = np.random.random()
408. weight9 = np.random.random()
409. weight10 = np.random.random()
410. weight11 = np.random.random()
411. weight12 = np.random.random()
412. weight13 = np.random.random()
413. weight14 = np.random.random()
414. weight15 = np.random.random()
415. weight16 = np.random.random()
416. weight17 = np.random.random()
417. weight18 = np.random.random()


421. inputs = [(dlya\_3(input\_arr\_y1[i])) **for** i **in** range(200)]
422. inputs2 = [(dlya2(input\_arr\_y2[i])) **for** i **in** range(200)]
423. inputs3 = [(dlya2(input\_arr\_y3[i])) **for** i **in** range(200)]
424. inputs4 = [(dlya\_3([ideal\_value\_y1[i], ideal\_value\_y2[i], ideal\_value\_y3[i]])) **for** i **in** range(200)]
426. #for i in range(200):
427. #    for j in range(200):
428. #       input\_layer = np.array(inputs[i])
429. #      output = sigmoid(np.dot(input\_layer,weights))
430. #
431. #       err = ideal\_value\_y1[i] - output
432. #      adjustmens = err \* (output \* (1 - output))
433. #     weights += adjustmens
435. **for** i **in** range(200):
436. **for** j **in** range(200):
437. S1 = inputs[i][0]\* weight1
438. S2 = inputs[i][1] \* weight2
439. S3 = inputs[i][2] \* weight3
440. S4 = inputs[i][3] \* weight4
441. S5 = inputs[i][4] \* weight5
442. S11 = S1+S2+S3+S4+S5
443. S6 = inputs2[i][0] \* weight6
444. S7 = inputs2[i][1] \* weight7
445. S8 = inputs2[i][2] \* weight8
446. S9 = inputs2[i][3] \* weight9
447. S10 = inputs2[i][4] \* weight10
448. S21 = S6+S7+S8+S9+S10
449. S11 = inputs3[i][0] \* weight11
450. S12 = inputs3[i][1] \* weight12
451. S13 = inputs3[i][2] \* weight13
452. S14 = inputs3[i][3] \* weight14
453. S15 = inputs3[i][4] \* weight15
454. S31 = S11+S12+S13+S14+S15
455. F1 = sigmoid(S11)
456. F2 = sigmoid(S21)
457. F3 = sigmoid(S31)
458. S4 = F1\*weight16+F2\*weight17+F3\*weight18
459. F4 = sigmoid(S4)
460. D = ideal\_value\_y4[i] - F4
461. D1 = D\*weight16
462. D2 = D\*weight17
463. D3 = D\*weight18
464. weight1 = weight1 + D1\*F1\*(1-F1)\*0.01\*inputs[i][0]
465. weight2 = weight2 + D1\*F1\*(1-F1)\*0.01\*inputs[i][1]
466. weight3 = weight3 + D1 \* F1 \* (1 - F1) \* 0.01\*inputs[i][2]
467. weight4 = weight4 + D1 \* F1 \* (1 - F1) \* 0.01\*inputs[i][3]
468. weight5 = weight5 + D1 \* F1 \* (1 - F1) \* 0.01\*inputs[i][4]
469. weight6 = weight6 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][0]
470. weight7 = weight7 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][1]
471. weight8 = weight8 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][2]
472. weight9 = weight9 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][3]
473. weight10 = weight10 + D2 \* F2 \* (1 - F2) \* 0.01\*inputs2[i][4]
474. weight11 = weight11 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][0]
475. weight12 = weight12 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][1]
476. weight13 = weight13 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][2]
477. weight14 = weight14 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][3]
478. weight15 = weight15 + D3 \* F3 \* (1 - F3) \* 0.01\*inputs3[i][4]
479. weight16 = weight16 + D \* F4 \* (1 - F4) \* 0.01\*F1
480. weight17 = weight17 + D \* F4 \* (1 - F4) \* 0.01\*F2
481. weight18 = weight18 + D \* F4 \* (1 - F4) \* 0.01\*F3
483. **print**(ideal\_value\_y4)
484. y\_idol = ideal\_value\_y4[0]
485. y\_new = inputs4[0]
486. **print**(y\_idol)
487. **print**(y\_new)
488. **print**(F4)