МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: «Моделирование нейро-нечетких сетей при анализе статистических данных»

Студент Д.Х. Буриев
(И.О. Фамилия) (личная подпись)

Руководитель к.тех.н, В.С. Климов
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

АННОТАЦИЯ

Тема бакалаврской работы: «Моделирование нейро-нечетких сетей при анализе статистических данных».

В данной бакалаврской работе исследуются практические аспекты реализации программного обеспечения для моделирования нейро-нечетких сетей ANFIS.

В бакалаврской работе проводится анализ библиотек на языке Python по машинному обучению, рассматривается математический аппарат нейро-нечетких сетей ANFIS, а также приводятся результаты разработки собственного программного класса для моделирования работы, обучение и тестирования сетей ANFIS.

Структура бакалаврской работы представлена введением, тремя главами, заключением, списком литературы.

Во введении описывается актуальность проводимого исследования, дается краткая характеристика проделанной работы.

В первой главе приведен сравнительный анализ библиотек Python по машинному обучению, на основе которого формулируется проблема исследования, ставится цель, а также формулируются задачи.

Во второй главе описывается математический аппарат нейро-нечетких сетей ANFIS, рассматривается алгоритм настройки сетей.

В третьей главе приведено описание разработанного программного обеспечения для моделирования работы, обучение и тестирования сетей ANFIS и рассматривается пример его использования.

В заключении представлены выводы по проделанной работе.

В работе использовано 2 таблицы, 18 рисунков, список литературы содержит 20 литературных источников. Общий объем выпускной квалификационной работы составляет 49 страниц.

ABSTRACT

The theme of the bachelor's work: "Modeling neural fuzzy networks in the analysis of statistical data".

In the given bachelor's work practical aspects of software realization for modelling of neurofine fuzzy networks ANFIS are investigated.

In the bachelor's work the analysis of libraries in Python language on machine learning is carried out, the mathematical device of neuro- fuzzy ANFIS networks is considered, and also results of development of own program class for modelling of work, training and testing of ANFIS networks are resulted.

The structure of the bachelor's work is represented by an introduction, three chapters, conclusion, and a list of literature.

The introduction describes the relevance of the ongoing research and gives a brief description of the work done.

The first chapter provides a comparative analysis of Python libraries on machine learning, based on which the research problem is formulated, the goal is set, and the objectives are formulated.

The second chapter describes the mathematical apparatus of neural fuzzy ANFIS networks and considers the algorithm of network configuration.

The third chapter describes the developed software for modeling, training and testing ANFIS networks and considers an example of its use.

In conclusion, the conclusions on the work done are presented.

In work 2 tables, 18 figures are used, the list of literature contains 20 literature sources. The total volume of the final qualification work is 49 pages.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Исследования в области искусственного интеллекта динамично развиваются, это приводит к появлению новых алгоритмов машинного обучения основанных на комбинировании технологий. Примером, объединения нескольких технологий является нейро-нечеткая сеть ANFIS, объединяющая в себе теорию нечетких множеств и нейросетевые технологии. Для того, чтобы изучать особенности сети ANFIS необходимо сначала ее реализовать на каком либо языке программирования.

В данной бакалаврской работе был проведен анализ библиотек на языке Python по машинному обучению: Scikit-learn, Tensorflow, Keras и pyTorch. Анализ показал, что в этих билиотеках нет реализации сети ANFIS. Поэтому целью данной работы была определена следующим образом – разработка на языке Python класса, позволяющего моделировать работу, обучение и тестирование ANFIS сети.

При выполнении бакалаврской работы решались следующие задачи:

1. Анализ библиотек на языке программирования Python по машинному обучению.

2. Математическое моделирование адаптивной нейро-нечеткой системы логического вывода ANFIS.

3. Разработка класса на языке Python, позволяющего моделировать работу, обучение и тестирование ANFIS сети.

При выполнении бакалаврской работы на языке Python был реализован класс, позволяющий моделировать работу, обучение и тестирование ANFIS сети. Описание программного кода, пример использования класса представлены в третьей главе.

Также в ходе выполнения исследований была изучены отечественные [1-10] и зарубежные [11-20] статьи об особенностях работы нейро-нечетких сетей ANFIS.

1 АНАЛИЗ БИБЛИОТЕК НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON ПО МАШИННОМУ ОБУЧЕНИЮ

Язык программирования Python в последнее время все чаще используется для анализа данных, как в науке, так и коммерческой сфере. Этому способствует простота языка, а также большое разнообразие открытых библиотек.

Особую популярность Python получил в сфере машинного обучения и анализа данных. Комбинация последовательного синтаксиса, более короткого времени разработки и гибкости делает этот язык подходящим для разработки и настройки моделей, описывающих данные различной природы.

Одно из самых больших преимуществ Python – широкий спектр библиотек. Библиотеки представляют собой наборы подпрограмм и функций, написанных на данном языке. Хороший комплект библиотек может облегчить осуществление сложных задач без необходимости написания многих строк кода.

Машинное обучение во многом основано на математике. В частности, на математической оптимизации, статистике и теории вероятности. Библиотеки Python помогают исследователям использовать методы машинного обучения, не имея значительных познаний в алгоритмизации и математике.

Исследования в области искусственного интеллекта динамично развиваются, это приводит к появлению новых алгоритмов машинного обучения основанных на комбинировании технологий. Примером, объединения нескольких технологий является адаптивная нейро-нечеткая система логического вывода ANFIS, объединяющая в себе теорию нечетких множеств и нейросетевые технологии. Для изучения особенностей сети ANFIS необходимо ее сначала программно реализовать. Рассмотрим существующие библиотеки по машинному обучению с целью определения реализованных в них алгоритмов (Таблица 1).

Таблица 1 – Сравнение функциональности библиотек по машинному обучению на языке программирования Python

Алгоритм машинного обучения	Библиотеки Python по машинному обучению			
	Scikit-learn	Tensorflow	Keras	pyTorch
k-means	+	–	+	+
k-ближайших соседей	+	–	–	–
Деревья принятия решений (классификационные)	–/+	–	–/+	–
Деревья принятия решений (регрессионные)	+	–	+	+
Рекуррентные нейронные сети	–	+	+	+
Нейронные сети прямого распространения	+	+	+	+
Глубокие нейронные сети	–	+	+	+
Генетические алгоритмы	–	–	+	+
Нейро-нечеткие сети ANFIS	–	–	–	–

В таблице 1 представлены результаты сравнительного анализа библиотек Python по машинному обучению. Как видно из анализа в библиотеках Scikit-learn, Tensorflow, Keras, pyTorch присутствуют классы, отвечающие за работу с нейронными сетями прямого распространения, на отсутствуют реализация функций для моделирования нейро-нечетких сетей ANFIS. Это можно объяснить тем, что описание работы нейро-нечетких сетей в научном сообществе появилось относительно недавно, и разработчики библиотек не успели реализовать функции для моделирования и работы с сетями ANFIS.

На преодоление этой проблемы и направленны исследования, представленные в данной бакалаврской работе.

Таким образом, актуальной можно признать цель работы – разработка на языке Python класса, позволяющего моделировать работу, обучение и тестирование ANFIS сети.

Данную цель предполагается достигнуть путем решения задач следующих задач:

1. Анализ библиотек на языке программирования Python по машинному обучению.

2. Математическое моделирование адаптивной нейро-нечеткой системы логического вывода ANFIS.

3. Разработка класса на языке Python, позволяющего моделировать работу, обучение и тестирование ANFIS сети.

В качестве выводов можно отметить следующее. В рамках анализа, представленного в первой главе работы, были сформулированы, решаемая проблема, цели и задачи бакалаврской работы.

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ СЕТИ ANFIS

2.1 Адаптивная нейро-нечеткая система логического вывода ANFIS

Математическая теория нечетких множеств (fuzzy sets) и нечеткая логика (fuzzy logic) являются обобщениями классической теории множеств и классической формальной логики. Данные понятия были впервые предложены американским ученым Лотфи Заде (Lotfi Zadeh) в 1965 г. Основной причиной появления новой теории стало наличие нечетких и приближенных рассуждений при описании человеком процессов, систем, объектов. В настоящее время экспертные системы на основе нечетких правил применяются в автомобильной, аэрокосмической и транспортной промышленности, в области изделий бытовой техники, в сфере финансов, анализа и принятия управленческих решений и многих других [1-2].

Нечеткие системы, широко применяемые для понимания поведения системы, очень интерпретируемы и способны моделировать человеческие знания с помощью понятных лингвистических терминов. Однако, основным их недостатком является необходимость в привлечении экспертов исследуемой области для формирования правил и функций принадлежности. Именно поэтому было решено совместить нечеткие системы с нейронными сетями, обладающими хорошими обучающими возможностями, но которым не хватает способности к интерпретации. В результате появились нейро-нечеткие системы, которые стали довольно мощным инструментом для работы с нечеткими множествами [3-5]. Наибольшее распространение в настоящее время получила архитектура нейро-нечеткой системы ANFIS, в который вывод осуществляется на основе аппарата нечеткой логики, а параметры функций принадлежности настраиваются с использованием метода обратного распространения ошибки [6].

ANFIS (adaptive neuro-fuzzy inference system) — адаптивная сеть на основе системы нечеткого вывода Такаги-Сугено. Данная система соответствует набору нечетких правил IF-THEN, которые обладают

способностью к обучению для аппроксимации нелинейных функций [7-8]. Общая форма модели вывода Такаги-Сугено может быть представлена следующим образом [9-10]:

ЕСЛИ x_1это A_1 И x_2 это A_2 И …. И x_nэто A_n, ТО y = f($x_1, x_2, \dots, x_n$).

Структура ANFIS представлена на рисунке 1.

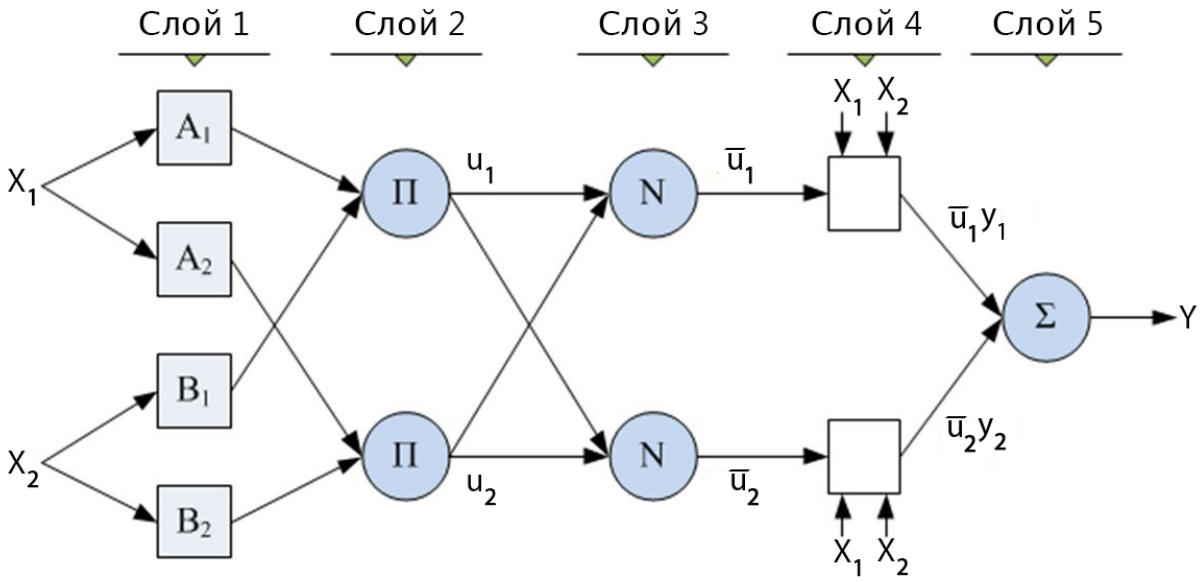

Рисунок 2.1 – Структура адаптивной нейро-нечеткой системы вывода (ANFIS)

Правила вывода Такаги-Сугено для нейро-нечеткой сети, изображенной на рисунке 2.1, будут выглядеть следующим образом:

ЕСЛИ x_1это A_1 И x_2 это B_1, ТО y = f(x_1, x_2);

ЕСЛИ x_1это A_2 И x_2 это B_2, ТО y = f(x_1, x_2).

Как видно из рисунка 2.1, данная сеть состоит из 5 слоев, каждый из который выполняет определенную функцию [11-12].

Первый слой является слоем фаззификации, роль которого — установить соответствие между численными значениями входных переменных системы нечеткого вывода и значениями функций принадлежности, соответствующих им термам лингвистических переменных. Лингвистическая переменная — это переменная, которая может принимать значения фраз из естественного или искусственного языка. Например, лингвистическая переменная «скорость» может иметь термы «высокая»,

«средняя», «очень низкая» и т. д. Таким образом, каждый нейрон на первом слое в качестве функции активации содержит функцию принадлежности, определяющую степень с которой входы соответствуют всем своим термам. Степени принадлежности каждой функции вычисляются с использованием набора параметров предпосылки {a, b, c}. В качестве функции принадлежности может быть выбрана, к примеру, гауссова функция [13-14].

$$g(x) = ae^{-\frac{(x-b)^2}{2c^2}}, \quad (1)$$

где *a* - максимальная высота, *b* - сдвиг пика от 0, *c* - ширина колокола. Однако так как значения функция принадлежности должны изменяться в диапазоне от [0; 1], то параметр *a* всегда будет равен 1.

Второй слой обозначается как «слой правил», так как количество нейронов на нем равняется количеству правил системы. Каждый узел второго слоя соединен только с теми узлами первого слоя, которые формируют посылки соответствующего правила. Результатом слоя являются антецеденты (посылки) нечетких правил, другими словами, выходом каждого узла будет являться степень истинности текущего правила. Данная степень определяется как логическое произведение (минимум) от множества степеней принадлежности [15-16].

$$u_i = \mu_{A_i}(x_1) * \mu_{B_i}(x_2), i = 1,..m, \quad (2)$$

где m-количество нечетких правил.

Пример выполнения данного произведения для двух входов и двух правил для каждого входа представлен на рисунке 2.2.

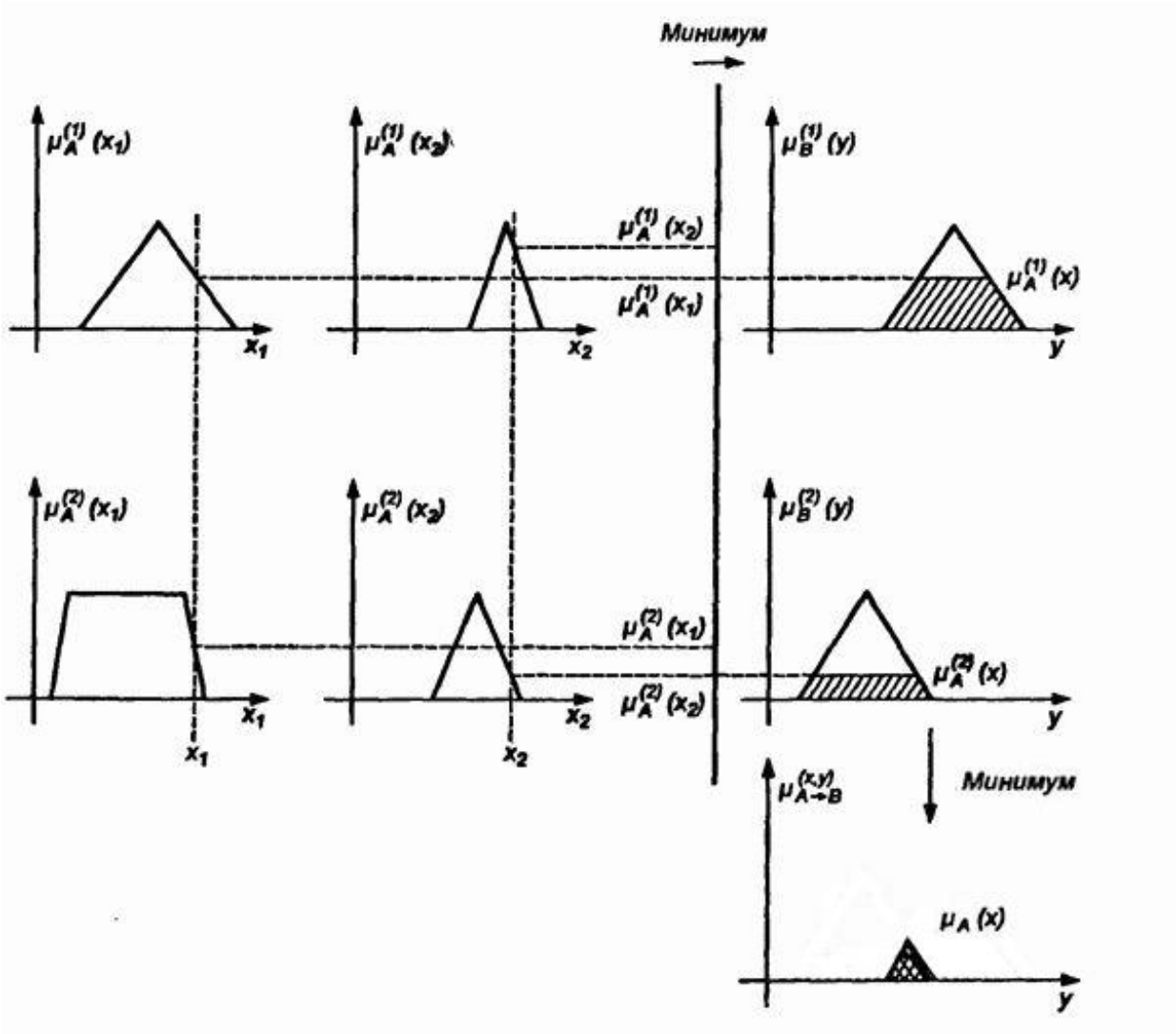

Рисунок 2.2 – Определение степени истинности правил

Третий слой выполняет нормализацию вычисленных степеней истинностей правил. Узлы этого слоя рассчитывают относительный вес выполнения нечеткого правила [17-18].

$$\bar{u}_i = \frac{u_i}{\sum_{j=1}^{m} u_j}, i = 1,..m, \quad (3)$$

где m - количество нечетких правил.

Четвертый слой дает на выходе заключения правил. Каждый узел рассчитывает вклад одного нечеткого правила в выход всей сети. Все узлы соединены с одним из выходов предыдущего слоя, а также со всеми входами сети. Кроме нормализованных значений, полученных с предыдущего слоя, данный слой оперирует набором, так называемых, параметров заключения

(следствия) $\{c_{ij}, i = 1,..m; j = 0,..m\}$.

$$\bar{u}_i y_i = \bar{u}_i(c_{i1}x_1 + c_{i2}x_2 + \cdots + c_{i0}) \quad (4)$$

,

где m - количество входов, i - количество узлов на 4 слое.

На пятом слое содержится всего один узел, который выполняет суммирование вкладов всех правил.

$$y = \sum_{i=1}^{m} \bar{u}_i y_i \quad (5)$$

Процесс обучения сети выполняется итерационно. Каждая итерация состоит из двух этапов. На первом этапе при помощи обучающей выборки находятся оптимальные параметры заключений четвертого слоя с помощью метода наименьших квадратов, после чего с помощью метода обратного распространения ошибки изменяются параметры предпосылок. Данная процедура выполняется до тех пор, пока ошибка не уменьшается до требуемого значения [19-20].

Для вычисления параметров заключений из формулы (4) с использованием метода наименьших квадратов, задача обучения представляется в виде AX=B, где X матрица, содержащая параметры заключений. Данная итерационная процедура приведена ниже.

$$S_{i+1} = S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}$$
$$X_{i+1} = X_i + S_{i+1} a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i), \quad (6)$$

где $X_0 = 0;\ S_0 = \gamma I$ (γ-большое число, I -единичная матрица); a_i^T – i-я строка матрицы A; b_i^T – i-ый элементы вектора B.

Строка матрицы A будет выглядеть следующим образом:

$$a_i^T = x_1^{(i)} * \bar{u}_1^{(i)} \quad ... \quad x_n^{(i)} * \bar{u}_1^{(i)} \quad \bar{u}_1^{(i)} \quad ... \quad x_1^{(i)} * \bar{u}_1^{(i)} \quad ... \quad x_n^{(i)} * \bar{u}_1^{(i)} \quad \bar{u}_1^{(i)}$$

(i = 1, ... m), где m – число примеров в обучающей выборке.

После получения параметров заключений можно начать настройку

параметров предпосылок с помощью метода обратного распространения ошибок. Найдем сумму квадратов ошибки сети. Для этого из ожидаемого результата выхода сети вычтем результат, полученный при прямом проходе для каждого примера обучающей выборки:

$$E = \sum_{p=1}^{P} E_p$$
$$E_p = \left(T_p - O_p^L\right)^2, \tag{7}$$

где P – размер обучающей выборки, T – действительный результат, а O^L – выход последнего слоя нейронной сети.

Чтобы реализовать процедуру обучения, выполняющую градиентный спуск в пространстве параметров, необходимо вычислить коэффициент погрешности $\frac{\partial E_p}{\partial O}$ для всех тренировочных данных и выходов каждого узла O. Коэффициент погрешности для выходного узла (L, i) можно рассчитать по следующей формуле:

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2\left(T_{i,p} - O_{i,p}^L\right), \tag{8}$$

где L – номер последнего слоя, i – номер узла.

Для внутреннего узла (k, i) коэффициент ошибок может быть выведен по правилу:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{K+1} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} * \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}, \tag{9}$$

где $1 \leq k \leq L\text{-}1$ и K – количество узлов k-го слоя. Таким образом, коэффициент ошибок внутреннего узла выражается в виде линейной комбинации коэффициентов ошибок узлов следующего слоя. Поэтому ошибку для всех узлов на каждом слое можно найти, используя формулы (8) и (9).

Теперь, если, а это параметр сети, а точнее функции принадлежности на 1 слое, то коэффициент ошибки для него высчитывается следующим образом:

$$\frac{\partial E_p}{\partial a} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} * \frac{\partial O^*}{\partial a}, \tag{10}$$

где S - набор узлов, выходные данные которых зависят от a.

Тогда общая погрешность по отношению k равна:

$$\frac{\partial E}{\partial a} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial a} \tag{11}$$

Таким образом, обновленное значение параметра a примет вид:

$$\Delta a = -\eta * \frac{\partial E}{\partial a}, \tag{12}$$

где η – скорость обучения, которая может быть посчитана по следующей формуле:

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial a}\right)^2}} \tag{13}$$

2.2 Пример корректирования параметров сети

Для лучшего понимания принципа работы нейро-нечеткой системы ANFIS будет рассмотрен пример обучения сети «с нуля». Так как данный вид нейро-нечеткой системы обладает хорошей аппроксимационной способностью нелинейных функций, то в качестве обучающей выборки будет приняты параметры функции от двух переменных и её результат.

Таблица 2 – Обучающая выборка

x1	x2	y
-10	-8	-0.00672789715494514
-8	2	0.0562263067414631
-6	4	0.00881093109436063
-4	0	-0.189200623826982
-2	-2	0.206705452607952
0	-10	-0.054402111088937
0	0	1
2	-8	0.0562263067414631
4	4	0.0357968760565192
6	-4	0.00881093109436063
8	10	-0.00672789715494514
10	6	0.00253346549549811

Таким образом, сеть будет иметь на входе две переменные x1 и x2. Пусть каждая из переменных принимает значение из множества термов {A1, A2} и {B1, B2} соответственно. В таком случае третий слой будет содержать 4 правила. Функция принадлежности A_1 имеет параметры b = -10 и c = 4, A_2: b = 10 и c = 6, B_1: b = -10 и c = 8, B_2: b = 7 и c = 6. В таком случае имеем следующие функции принадлежности:

$$\mu_{A_1}(x) = e^{-\frac{(x+10)^2}{32}}, \mu_{A_2}(x) = e^{-\frac{(x-10)^2}{72}}$$

$$\mu_{B_1}(x) = e^{-\frac{(x+10)^2}{128}}, \mu_{B_2}(x) = e^{-\frac{(x-7)^2}{72}}$$

Данные функции принадлежности представлены на рисунке 2.3.

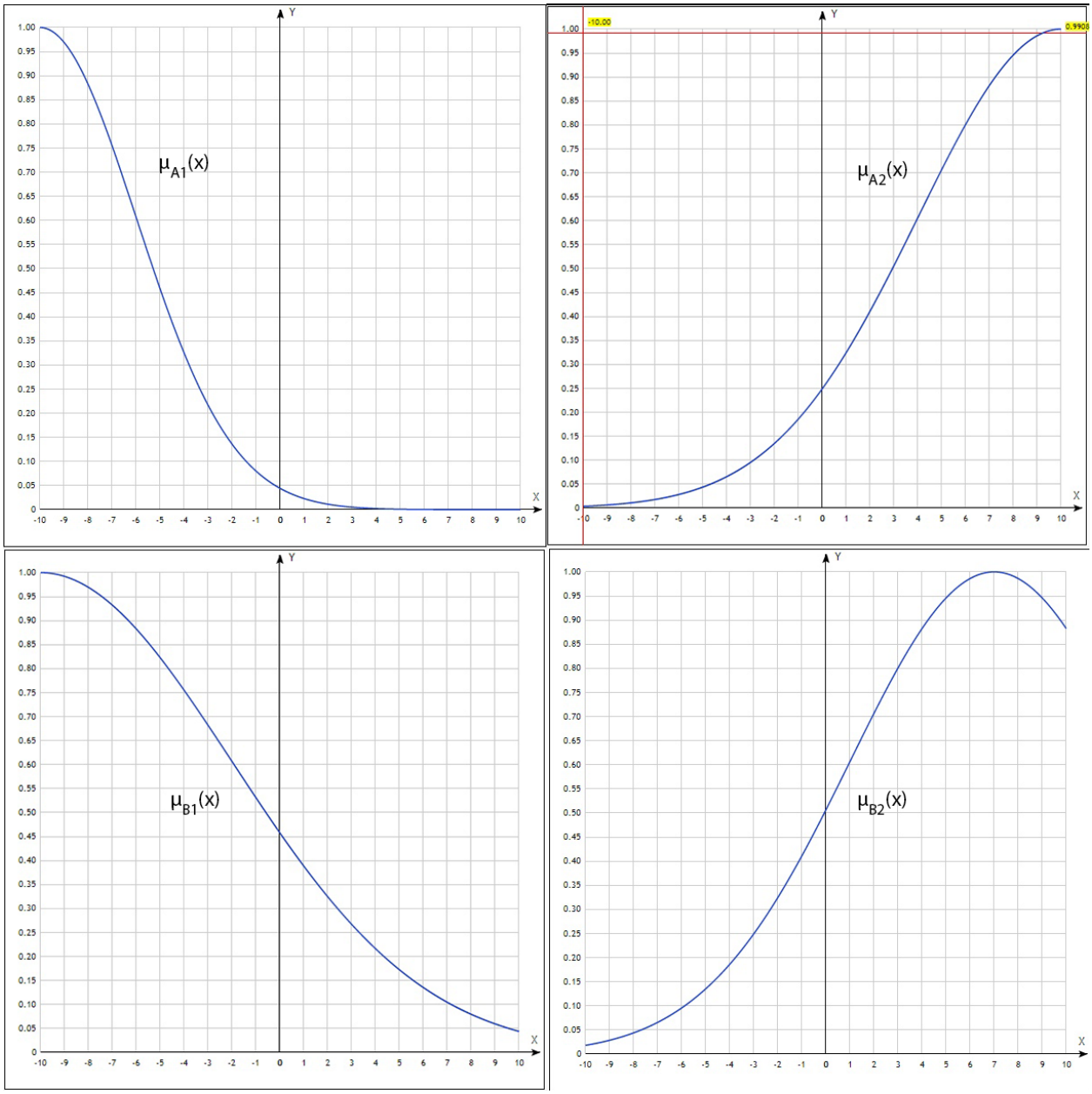

Рисунок 2.3 – Функции принадлежности слоя фаззификации

Вычислим выходные значения первого слоя для первого примера обучающей выборки $x_1 = -10, x_2 = -8$.

$\mu_{A_1}(x_1) = 1$

$\mu_{A_2}(x_1) = 0.003866$

$\mu_{B_1}(x_2) = 0.969233$

$\mu_{B_2}(x_2) = 0.043938$

Вычислим выходные значения второго слоя:

$u_1 = \mu_{A_1}(x_1) * \mu_{B_1}(x_2) = 1 * 0.969233 = 0.969233$

$$u_2 = \mu_{A_2}(x_1) * \mu_{B_2}(x_2) = 0.003866 * 0.043938 = 0.00017$$

Вычислим выходные значения третьего слоя:

$$\bar{u}_1 = \frac{u_1}{u_1 + u_2} = \frac{0.969233}{0.969233 + 0.00017} = 0.999825$$

$$\bar{u}_2 = \frac{u_2}{u_1 + u_2} = \frac{0.00017}{0.969233 + 0.00017} = 0.000175$$

Следующим шагом будет вычисление выхода четвертого слоя, для этого необходимо найти параметры заключений по формуле (6).

Для расчета матрицы A необходимо для каждого примера из обучающей выборки вычислить выходные значения первого, второго и третьего слоев. Таким образом, матрица A примет вид:

$$\begin{pmatrix} -9.9982478 & -7.99859826 & 0.99982478 & -0.001752178 & -0.0014017424 & 0.0001752178 \\ -7.9464572 & -1.986614298 & 0.993307149 & -0.0535428074 & -0.0133857019 & 0.006692851 \\ -5.9310695 & -3.9540463348 & 0.9885115837 & -0.0689304978 & -0.0459536652 & 0.0114884163 \\ -3.2682241416 & 0.0 & 0.8170560354 & -0.7317758584 & 0.0 & 0.1829439646 \\ -1.3027097293 & -1.3027097293 & 0.6513548647 & -0.6972902707 & -0.6972902707 & 0.3486451353 \\ 0.0 & -9.0701980548 & 0.9070198054 & 0.0 & -0.9298019452 & 0.0929801945 \\ 0.0 & 0.0 & 0.1374295321 & 0.0 & 0.0 & 0.8625704679 \\ 0.7469390018 & -2.9877560074 & 0.3734695009 & 1.2530609982 & -5.0122439926 & 0.6265304991 \\ 0.0035321834 & 0.0035321834 & 0.0008830458 & 3.9964678166 & 3.9964678166 & 0.9991169542 \\ 0.0101690241 & -0.0067793494 & 0.0016948375 & 5.9898309759 & -3.9932206506 & 0.9983051626 \\ 0.0000168695 & 0.0000210868 & 0.0000021087 & 7.9999831305 & 9.9999789132 & 0.99999789132 \\ 0.00000511401 & 0.0000030684 & 0.0000005114 & 9.999994886 & 5.9999969316 & 0.9999994886 \end{pmatrix}^T$$

Вычислим S_1:

$$S_1 = S_0 - \frac{S_0 a_1 a_1^T S_0}{1 + a_1^T S_0 a_1}$$

Матрицу S_0 зададим как:

$$S_0 = \begin{pmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{pmatrix}$$

Для начала посчитаем числитель:

$$S_0 a_1 a_1^T S_0 = \begin{pmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{pmatrix} * \begin{pmatrix} -9.99825 \\ -7.9986 \\ 0.999825 \\ -0.00175 \\ -0.0014 \\ 0.000175 \end{pmatrix} * \begin{pmatrix} -9.99825 \\ -7.9986 \\ 0.999825 \\ -0.00175 \\ -0.0014 \\ 0.000175 \end{pmatrix}^T *$$

$$* \begin{pmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{pmatrix} =$$

$$= \begin{pmatrix} 99965003.1 & 79972002.4 & -9996500.31 & 17496.9375 & 13997.55 & -1749.69375 \\ 79972002.4 & 63977602 & -7997200.24 & 13997.55 & 11198.04 & -1399.755 \\ -9996500.31 & -7997200.24 & 999650.031 & -1749.69375 & -1399.755 & 174.969375 \\ 17\,496.9375 & 13\,997.55 & -1\,749.69375 & 3.0625 & 2.45 & -0.30625 \\ 13997.55 & 11198.04 & -1399.755 & 2.45 & 1.96 & -0.245 \\ -1749.69375 & -1399.755 & 174.969375 & -0.30625 & -0.245 & 0.030625 \end{pmatrix}$$

Рассчитаем знаменатель:

$$1 + a_1^T S_0 a_1 = 1 + \begin{pmatrix} -9.99825 \\ -7.9986 \\ 0.999825 \\ -0.00175 \\ -0.0014 \\ 0.000175 \end{pmatrix}^T * \begin{pmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{pmatrix}$$

$$* \begin{pmatrix} -9.99825 \\ -7.9986 \\ 0.999825 \\ -0.00175 \\ -0.0014 \\ 0.000175 \end{pmatrix} = 164943.2601$$

Теперь можно рассчитать S_1, используя результаты предыдущих вычислений.

$$S_1 = S_0 - \frac{S_0 a_1 a_1^T S_0}{1 + a_1^T S_0 a_1} =$$

$$\begin{pmatrix} 393.943 & -484.8456 & 60.6057 & -0.106080 & -0.08486 & 0.010607 \\ -484.84553 & 612.12358 & 48.48455 & -0.08486282 & -0.06789 & 0.0084863 \\ 60.60569 & 48.484553 & 993.93943 & 0.0106079 & 0.0084863 & -0.0010608 \\ -0.106079 & -0.084863 & 0.010608 & 999.99998 & -0.000014854 & 0.0000018567 \\ -0.084863 & -0.0678902 & 0.0084863 & -0.000014853 & 999.99999 & 0.00000148536 \\ 0.01060786 & 0.0084863 & -0.0010608 & 0.0000018567 & 0.00000148536 & 1000 \end{pmatrix}$$

После расчета матрицы S можно получить X_1:

$$X_1 = X_0 + S_1 a_1 (b_1^T - a_1^T X_0)$$

$$b_1^T - a_1^T X_0 = -0.006728 - \begin{pmatrix} -9.99825 \\ -7.9986 \\ 0.999825 \\ -0.00175 \\ -0.0014 \\ 0.000175 \end{pmatrix}^T * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 0.00253347$$

$$S_1 a_1 = \begin{pmatrix} 0.07300494 \\ 0.043802965 \\ 0.0073005 \\ 0.0000127781 \\ 0.00000766686 \\ 0.00000127781 \end{pmatrix}$$

$$X_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.07300494 \\ 0.043802965 \\ 0.0073005 \\ 0.0000127781 \\ 0.00000766686 \\ 0.00000127781 \end{pmatrix} * 0.00253347 = \begin{pmatrix} 0.000407820227 \\ 0.000326256181 \\ -0.0000407820227 \\ 0.0000000713810314 \\ 0.0000000571048251 \\ -0.00000000713810314 \end{pmatrix}$$

Таким образом, повторяя данную процедуру итерационно, получим вектор X_{12}, содержащий в себе коэффициенты заключений “c” из формулы (4).

$$X_{12} = \begin{pmatrix} -0.027101 \\ -0.015111 \\ -0.317969 \\ -0.105891 \\ 0.014854 \\ 0.752904 \end{pmatrix}$$

$$\bar{u}_1 y_1 = \bar{u}_1((-0.027101 * -10) + (-0.015111 * -8) + (-0.317969))$$
$$= 0.999825 * 0.07393211 = 0.07391915$$

$$\bar{u}_2 y_2 = \bar{u}_2((-0.105891 * -10) + (0.014854 * -8) + 0.752904)$$
$$= 0.000175 * 1.69298363 = 0.00029664$$

Вычислим выход сети, для этого необходимо просуммировать значения, полученные на 4 слое:

$$y = \sum_{i=1}^{m} \bar{u}_i y_i = 0.07391915 + 0.00029664 = 0.07421579$$

Таким образом, получен выход пятого слоя для первого примера обучающей выборки. Для того, чтобы начать процедуру изменения параметров предпосылок, необходимо рассчитать выходы для всех примеров обучающей выборки. При прямом проходе для вычисления остальных выходов сети используются ранее вычисленные параметры заключения.

Выход сети для всей обучающей выборки примет следующий вид:

$$\vec{y} = \begin{pmatrix} 0.07421579 \\ -0.05995294 \\ -0.07856074 \\ 0.04400183 \\ 0.17385534 \\ -0.09514847 \\ 0.60573438 \\ 0.17073071 \\ 0.38798223 \\ 0.05732934 \\ 0.0543135 \\ -0.21688417 \end{pmatrix}$$

Следующим шагом будет расчет суммы квадратов ошибок сети по формуле (7). Квадрат ошибки для первой примера из обучающей выборки выглядит следующим образом:

$$e_1^2 = (-0.00672789715494514 - 0.07421579)^2 = 0.00655188$$

Таким же образом рассчитаем квадраты ошибки для остальных примеров и получим сумму квадратов ошибок для всех примеров обучающей выборки:

$$E^2 = \sum_{i=1}^{P} e_i^2 = 0.431621432424$$

Для того, чтобы откорректировать параметр b функции принадлежности A_1 (с параметрами b = -10, c = 4) необходимо воспользоваться формулой (12). Для начала необходимо вычислить $\frac{\partial E}{\partial b}$ - сумму всех поправок для этого параметра, которая находится по формуле (11). Каждая из поправок $\frac{\partial E_p}{\partial b}$, составляющих $\frac{\partial E}{\partial b}$, находится по следующей формуле:

$$\frac{\partial E_p}{\partial b} = \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b} \right) \right) \right) \right) \right)$$

Первым делом рассчитаем частные производные функции принадлежности по b и по c:

$$\frac{\partial g}{\partial b}(x) = \frac{(-2b + 2x)^2 * e^{\frac{-(x-b)^2}{2c^2}}}{2c^2}$$

$$\frac{\partial g}{\partial c}(x) = \frac{(-b+x)^2 * e^{\frac{-(x-b)^2}{2c^2}}}{c^3}$$

Теперь рассчитаем отдельно частные производные в $\frac{\partial E_p}{\partial b}$:

$$\frac{\partial E_p}{\partial e} = -2\left(T_p - O_p^5\right)$$

$$\frac{\partial e}{\partial y}, \frac{\partial y}{\partial(\bar{u}_i y_i)} = 1$$

$$\frac{\partial(\bar{u}_i y_i)}{\partial \bar{u}_i} = c_{i1} * x_1 + c_{i2} * x_2 + c_{i0}$$

$$\frac{\partial \bar{u}_i}{\partial u_i} = -\frac{u_i}{(u_1+u_2)^2} * \frac{1}{u_1+u_2}$$

$$\frac{\partial \bar{u}_i}{\partial u_j} = -\frac{u_i}{\left(u_i+u_j\right)^2}$$

$$\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b} = \mu_{B_1} * \frac{(-2b+2x)^2 * e^{\frac{-(x-b)^2}{2c^2}}}{2c^2}$$

Вычислим поправку параметра b для первой функции принадлежности на основе первого набора из обучающей выборки:

$$\frac{\partial E_1}{\partial b} = \frac{\partial E_1}{\partial e} * \sum\left(\frac{\partial e}{\partial y} * \sum\left(\frac{\partial y}{\partial(\bar{u}_i y_i)} * \sum\left(\frac{\partial(\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum\left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum\left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b}\right)\right)\right)\right)\right)$$

Найдем все частные производные в $\frac{\partial E_1}{\partial b}$:

$$\frac{\partial E_1}{\partial e} = -2(-0.080943687) = 0.161887374$$

$$\frac{\partial(\bar{u}_1 f_1)}{\partial \bar{u}_1} = c_{11} * x_1 + c_{12} * x_2 + c_{10} = (-0.027101 * -10) + (-0.015111 * -8) + (-0.317969) = 0.07393211$$

$$\frac{\partial(\bar{u}_2 f_2)}{\partial \bar{u}_2} = c_{21} * x_1 + c_{22} * x_2 + c_{20} = (-0.105891 * -10) + (0.014854 * -8) + 0.752904 = 1.6929836$$

$$\frac{\partial \bar{u}_1}{\partial u_1} = -\frac{u_1}{(u_1+u_2)^2} * \frac{1}{u_1+u_2} = -\frac{0.969233}{(0.969233+0.00017)^2} * \frac{1}{0.969233+0.00017} = 1.063935$$

$$\frac{\partial \bar{u}_1}{\partial u_2} = -\frac{u_1}{(u_1+u_2)^2} = -\frac{0.969233}{(0.969233+0.00017)^2} = 1.031382$$

$$\frac{\partial \bar{u}_2}{\partial u_1} = -\frac{u_2}{(u_1+u_2)^2} = -\frac{0.00017}{(0.969233+0.00017)^2} = 0.000181$$

$$\frac{\partial \bar{u}_2}{\partial u_2} = -\frac{u_2}{(u_1+u_2)^2} * \frac{1}{u_1+u_2} = -\frac{0.00017}{(0.969233+0.00017)^2} * \frac{1}{0.969233+0.00017} = 0.0001866$$

$$\frac{\partial u_1}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b} = \mu_{B_1}(x_2) * \frac{(20+2*x_1)^2 * e^{\frac{-(x_1+10)^2}{32}}}{64} = \mu_{B_1}(-8) * 0 = 0$$

Отдельно рассчитаем значения всех сумм:

$$\sum\left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b}\right) = 0$$

$$\sum\left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum\left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b}\right)\right) = (1.063935 + 1.031382 + 0.000181 + 0.0001866) * 0 = 0$$

$$\sum\left(\frac{\partial(\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum\left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum\left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b}\right)\right)\right) = (0.07393211 + 1.69298363) * 0 = 0$$

$$\sum\left(\frac{\partial y}{\partial(\bar{u}_i y_i)} * \sum\left(\frac{\partial(\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum\left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum\left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b}\right)\right)\right)\right) = (1 + 1) * 0 = 0$$

$$\sum\left(\frac{\partial e}{\partial y} * \sum\left(\frac{\partial y}{\partial(\bar{u}_i y_i)} * \sum\left(\frac{\partial(\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum\left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum\left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b}\right)\right)\right)\right)\right) = 1 * 0 = 0$$

$$\frac{\partial E_1}{\partial b} = 0.161887374 * 0 = 0$$

Таким образом, была вычислена поправка параметра b для первой функции принадлежности на основе первой записи из обучающей выборки. Для того, чтобы вычислить общую поправку b для первой функции, необходимо проделать описанные выше вычисления для каждого примера обучающей выборки.

$$\frac{\partial E_1}{\partial b} = 0;\ \frac{\partial E_2}{\partial b} = -0.24706724;\ \frac{\partial E_3}{\partial b} = -0.33666892;$$

$$\frac{\partial E_4}{\partial b} = 1.79544692;\ \frac{\partial E_5}{\partial b} = -0.26820813;\ \frac{\partial E_6}{\partial b} = -0.87490483;$$

$$\frac{\partial E_7}{\partial b} = -0.46142541;\ \frac{\partial E_8}{\partial b} = 0.78290409;\ \frac{\partial E_9}{\partial b} = -0.00030598;$$

$$\frac{\partial E_{10}}{\partial b} = -0.000916;\ \frac{\partial E_{11}}{\partial b} = -0.000000804;\ \frac{\partial E_{12}}{\partial b} = 0.000001013$$

После сложения данных поправок будет получена общая поправка параметра a. Результат вычисления данной поправки приведен ниже:

$$\frac{\partial E}{\partial b} = \sum \frac{\partial E_p}{\partial b} = 0.3888547$$

Таким образом, значение обновления параметра b можно получить по формуле (12), для этого необходимо умножить полученную общую поправку на скорость обучения η. Однако для расчета скорости обучения необходимо рассчитать общие поправки $\frac{\partial E}{\partial b}, \frac{\partial E}{\partial c}$ для всех функций принадлежностей на основе всей обучающей выборки, количество которых будет 8.

Общие поправки для параметров функции принадлежности A_1:

$$\frac{\partial E^{A_1}}{\partial b} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial b} \right) \right) \right) \right) \right)$$
$$= 0.388854706439$$

$$\frac{\partial E^{A_1}}{\partial c} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{A_1}} * \frac{\partial \mu_{A_1}}{\partial c} \right) \right) \right) \right) \right)$$
$$= 0.699704754982$$

Общие поправки для параметров функции принадлежности A_2:

$$\frac{\partial E^{A_2}}{\partial b} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{A_2}} * \frac{\partial \mu_{A_2}}{\partial b} \right) \right) \right) \right) \right) = 0.77490066056$$

$$\frac{\partial E^{A_2}}{\partial c} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{A_2}} * \frac{\partial \mu_{A_2}}{\partial c} \right) \right) \right) \right) \right) = -1.0618156462$$

Общие поправки для параметров функции принадлежности B_1:

$$\frac{\partial E^{B_1}}{\partial b} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{B_1}} * \frac{\partial \mu_{B_1}}{\partial b} \right) \right) \right) \right) \right)$$
$$= 0.261148096916$$

$$\frac{\partial E^{B_1}}{\partial c} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{B_1}} * \frac{\partial \mu_{B_1}}{\partial c} \right) \right) \right) \right) \right)$$
$$= 0.399160851306$$

Общие поправки для параметров функции принадлежности B_2:

$$\frac{\partial E^{B_2}}{\partial b} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{B_2}} * \frac{\partial \mu_{B_2}}{\partial b} \right) \right) \right) \right) \right)$$
$$= 0.236040139039$$

$$\frac{\partial E^{B_2}}{\partial c} = \sum \frac{\partial E_p}{\partial e} * \sum \left(\frac{\partial e}{\partial y} * \sum \left(\frac{\partial y}{\partial (\bar{u}_i y_i)} * \sum \left(\frac{\partial (\bar{u}_i y_i)}{\partial \bar{u}_i} * \sum \left(\frac{\partial \bar{u}_i}{\partial u_i} * \sum \left(\frac{\partial u_i}{\partial \mu_{B_2}} * \frac{\partial \mu_{B_2}}{\partial c} \right) \right) \right) \right) \right) = 0.37743237762$$

Рассчитаем параметр скорости обучения по формуле (12):

$$\eta = \frac{k}{\sqrt{\sum_\alpha \left(\frac{\partial E}{\partial a} \right)^2}} = \frac{0.01}{\sqrt{\frac{\partial E^{A_1}}{\partial b} + \frac{\partial E^{A_1}}{\partial c} + \frac{\partial E^{A_2}}{\partial b} + \frac{\partial E^{A_2}}{\partial c} + \frac{\partial E^{B_1}}{\partial b} + \frac{\partial E^{B_1}}{\partial c} + \frac{\partial E^{B_2}}{\partial b} + \frac{\partial E^{B_2}}{\partial c}}}$$
$$= \frac{0.01}{\sqrt{2.075425940662}} = \frac{0.01}{1.4406338676645083} = 0.00694$$

Используя значение скорости обучения, вычислим значения обновления для параметров предпосылок:

$$\Delta b^{A_1} = -\eta * \frac{\partial E^{A_1}}{\partial b} = 0.00269;\ \Delta c^{A_1} = -\eta * \frac{\partial E^{A_1}}{\partial c} = 0.004856$$

$$\Delta b^{A_2} = -\eta * \frac{\partial E^{A_2}}{\partial b} = 0.005378;\ \Delta c^{A_2} = -\eta * \frac{\partial E^{A_2}}{\partial c} = -0.007369$$

$$\Delta b^{B_1} = -\eta * \frac{\partial E^{B_1}}{\partial b} = 0.001812;\ \Delta c^{B_1} = -\eta * \frac{\partial E^{B_1}}{\partial c} = 0.00277$$

$$\Delta b^{B_2} = -\eta * \frac{\partial E^{B_2}}{\partial b} = 0.001638\Delta;\ c^{B_2} = -\eta * \frac{\partial E^{B_2}}{\partial c} = 0.002619$$

Тогда обновленные параметры предпосылок примут следующий вид:

$$b^{A_1} = b^{A_1} + \Delta b^{A_1} = -10 + 0.002699 = -9.997301$$

$$c^{A_1} = c^{A_1} + \Delta c^{A_1} = 4 + 0.004856 = 4.004856$$

$$b^{A_2} = b^{A_2} + \Delta b^{A_2} = 10 + 0.005378 = 10.005378$$

$$c^{A_2} = c^{A_2} + \Delta c^{A_2} = 6 - 0.007369 = 5.992631$$

$$b^{B_1} = b^{B_1} + \Delta b^{B_1} = -10 + 0.001812 = -9.998188$$

$$c^{B_1} = c^{B_1} + \Delta c^{B_1} = 8 + 0.00277 = 8.00277$$

$$b^{B_2} = b^{B_2} + \Delta b^{B_2} = 7 + 0.001638 = 7.001638$$

$$c^{B_2} = c^{B_2} + \Delta c^{B_2} = 6 + 0.002619 = 6.002619$$

Таким образом, была проделана одна итерация процесса обучения данной нейро-нечеткой сети. На каждом последующем шаге выполняются аналогичные вычисления, в результате которых постепенно изменяются функции принадлежности. Процесс обучения продолжается до тех пор, пока ошибка выхода сети превышает заранее установленное значение.

Был рассмотрен математический аппарат сети ANFIS. В рамках бакалаврской работы на языке Python был реализован класс, позволяющей моделировать работу, обучение и тестирование ANFIS сети. Особенности программной реализации рассмотрены в следующей главе работы.

3 РАЗРАБОТАННОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ МОДЕЛИРОВАНИЯ СЕТИ ANFIS

3.1 Описание разработанного программного обеспечения

Для работы программы, реализующей построение, обучение и тестирование ANFIS сети необходимо импортировать следующие библиотеки:

- «numpy» – предназначена для работы с многомерными массивами, поддерживает высокоуровневые математические функции;
- «scikit-fuzzy» также известная как «skfuzzy» - набор инструментов для работы с нечеткой логикой, из этой библиотеки импортируем функции принадлежности, такие как: «gaussmf» - Гауссова функция принадлежности (Gaussian), «gbellmf» - колоколообразная функция принадлежности (Generalized bell), «sigmf» - сигмоидальная функция принадлежности (Sigmoid);
- «itertools» - модуль для создания собственных итераторов;
- «copy» - модуль для поверхностного или глубокого копирования объектов: списков, словарей и других;
- «matplotlib.pyplot» - инструмент для построения графиков.

На рисунке 3.1 представлен код, импортирующий библиотеки.

```
import numpy as np
from skfuzzy import gaussmf, gbellmf, sigmf
import itertools
import copy
import matplotlib.pyplot as plt
```

Рисунок 3.1 – Импорт библиотек

Рассмотрим вспомогательный класс, использующийся для удобной работы с функциями принадлежности, который отображен на рисунке 3.2. Класс содержит словарь, ключи в котором — это строки с названием функций принадлежности, а значения – соответствующие функции

принадлежности из библиотеки «skfuzzy». Полем класса является список, содержащий название и значение параметров функций принадлежности, например, для Гауссовой функции принадлежности параметрами являются «sigma» - стандартное отклонение и «mean» - середина.

Функция этого класса используется для вычисления степеней принадлежности к нечетким множествам. Для этого в функцию передаются входные переменные (x1, x2), затем происходит проверка совпадения количества входных переменных с количеством наборов функций принадлежности.

Функция возвращает список из двух подсписков, первый состоит из значений функций принадлежности для переменной x1, второй для x2. Название очередной функции берется из списка функций принадлежности и по этому названию из словаря подставляется соответствующая функция, ей в аргументы передается входная переменная, а также с помощью оператора «**» распаковываются параметры, такие как sigma и mean для Гауссовой функции.

Для каждой из входных переменных определен свой набор функций принадлежности.

```
class MemFuncs:
    'Common base class for all employees'
    funcDict = {'gaussmf': gaussmf, 'gbellmf': gbellmf, 'sigmf': sigmf}


    def __init__(self, MFList):
        self.MFList = MFList

    def evaluateMF(self, rowInput):
        if len(rowInput) != len(self.MFList):
            print("Number of variables does not match number of rule sets")

        return [[self.funcDict[self.MFList[i][k][0]](rowInput[i],**self.MFList[i][k]
```

Рисунок 3.2 – Класс для работы с функциями принадлежности

Перейдем к рассмотрению основного класса, реализующего сеть ANFIS, конструктор которого представлен на рисунке 3.3. При создании

объекта класса в качестве параметров передаются наборы значений переменных x1, x2 из обучающей выборки и значений y, а также объект класса «MemFuncs». Списки значений переменных копируются и переводятся в «numpy»-массивы, создаётся переменная, хранящая размер списка X. Переданный объект полностью копируется в следующее поля класса, в следующее поле «memFuncs» записывается непосредственно список функций принадлежностей из объекта класса «MemFuncs». Далее создается поле, хранящее номера функций принадлежности из списка по переменным, для этого создается список из двух подсписков, в которые записываются номера функций принадлежности, относящиеся к соответствующей входной переменной.

Например, если для x1 и x2 созданы по 4 функции принадлежности — это поле будет хранить два списка: [0,1,2,3], [0,1,2,3]. Поле с правилами создается следующим образом: с помощью библиотеки «itertools» функцией «product» перебираются элементы из предыдущего поля и получается список: [[0,0], [0,1], [0,2], …, [3,3]], который переводится в «numpy»-массив. Поле «consequents» будет содержать параметры заключения, которые находятся методом наименьших квадратов при прямом проходе, при создании объекта создается список размером: количество значений Y в одном тренировочном наборе * количество правил * (количество значений X в одном тренировочном наборе + 1), единица добавляется исходя из уравнения 4 слоя: c1*x1+c2*x2+c0, для того чтобы учесть коэффициент c0 и добавляется единица. Список заполняется нулями. В условиях примера размер списка будет 48 элементов.

Создается массив для ошибок и поле, проверяющее то что для каждой входной переменной сопоставлено одинаковое количество функций принадлежности, принимающее значение «True» в случае если это так и «False», если иначе. Создается поле для указания типа обучения.

```
def __init__(self, X, Y, memFunction):
    self.X = np.array(copy.copy(X))
    self.Y = np.array(copy.copy(Y))
    self.XLen = len(self.X)
    self.memClass = copy.deepcopy(memFunction)
    self.memFuncs = self.memClass.MFList
    self.memFuncsByVariable = [[x for x in range(len(self.memFuncs[z]))] for z in range(
    self.rules = np.array(list(itertools.product(*self.memFuncsByVariable)))
    self.consequents = np.empty(self.Y.ndim * len(self.rules) * (self.X.shape[1] + 1))
    self.consequents.fill(0)
    self.errors = np.empty(0)
    self.memFuncsHomo = all(len(i)==len(self.memFuncsByVariable[0]) for i in self.memFun
    self.trainingType = 'Not trained yet'
```

Рисунок 3.3 – Класс, реализующий сеть ANFIS

Рассмотрим функцию, реализующую первую половину прямого прохода до 4 слоя, то есть систему нечеткого вывода, код которой отображен на рисунке 3.4. Аргументами функции являются объекта класса «ANFIS» и список переменных X. В этой функции цикл проходит по всем наборам X из таблицы выборки. На первом слое вычисляются значения функций принадлежности с помощью соответствующей функции и в переменную записывается список, состоящий из двух подсписков со значениями функций принадлежности для соответствующего x (x1 и x2). Затем в цикле совершается проход по всем правилам и номера функций в правилах сопоставляются с их значениями, полученными на предыдущем слое, в итоге получается список правил с вычисленными значениями функций принадлежности. На следующем шаге необходимо вычислить степень истинности правил, для этого выполняется перемножение двух значений функций принадлежности в каждом правиле, получается список, содержащий степени истинности всех правил. На следующем слое необходим нормировать полученные степени истинности правил. Для этого с помощью функции «np.sum» производим суммирование степеней истинности всех правил, полученные на предыдущем слое. Затем если это первый набор входных переменных, то вычисляются нормированные степени истинности для этого набора и записываются в переменную, на последующих итерациях

степени записываются в стек, в итоге после всех итераций в стеке хранятся массивы степеней истинности правил для всех входных наборов переменных X. Производим подготовку данных для 4 слоя. Для этого добавляем в список x1, x2 и 1, после чего умножаем на степень истинности соответствующего правила. Все эти списки объединяем в один список, после чего добавляем этот список в общий список. В итоге после всех итераций разделяем общий список на количество подсписков равное количеству наборов в обучающей выборке и получаем матрицу A, которая используется для вычисления параметров заключения и выхода сети.

```
def forwardHalfPass(ANFISObj, Xs):
    layerFour = np.empty(0,)
    wSum = []

    for pattern in range(len(Xs[:,0])):
        #layer one
        layerOne = ANFISObj.memClass.evaluateMF(Xs[pattern,:])

        #layer two
        miAlloc = [[layerOne[x][ANFISObj.rules[row][x]] for x in range(len(ANFISObj.rule
        layerTwo = np.array([np.product(x) for x in miAlloc]).T
        if pattern == 0:
            w = layerTwo
        else:
            w = np.vstack((w,layerTwo))

        #layer three
        wSum.append(np.sum(layerTwo))
        if pattern == 0:
            wNormalized = layerTwo/wSum[pattern]
        else:
            wNormalized = np.vstack((wNormalized,layerTwo/wSum[pattern]))

        #prep for layer four (bit of a hack)
        layerThree = layerTwo/wSum[pattern]
        rowHolder = np.concatenate([x*np.append(Xs[pattern,:],1) for x in layerThree])
        layerFour = np.append(layerFour,rowHolder)

    w = w.T
    wNormalized = wNormalized.T

    layerFour = np.array(np.array_split(layerFour,pattern + 1))

    return layerFour, wSum, w
```

Рисунок 3.4 – Функция, реализующая прямой проход до 4 слоя

Перейдем к рассмотрению функции реализующей нахождение параметров заключения с помощью метода наименьших квадратов, код данной функции представлен на рисунке 3.5. Аргументами функции являются: матрица A, полученная как выход предыдущей рассмотренной функции, список значений Y из обучающей выборки, и гамма. Создается матрица S, необходимая для вычисления по итерационной формуле, размер матрицы S: (размер строки матрицы A * размер строки матрицы A). Полученная единичная матрица умножается на гамму. Затем создается вектор x, в который будут помещаться параметры заключения. Его размер равен размеру строки матрицы A. Вектор заполняется нулями. После чего в цикле происходит проход по всем строкам матрицы A и параметры заключения вычисляются по итерационным формулам с использованием функций из библиотеки «numpy». Выходом функции является список, состоящий из параметров заключения, для каждого уравнения по три параметра.

```
def LSE(self, A, B, initialGamma = 1000.):
    coeffMat = A
    rhsMat = B
    S = np.eye(coeffMat.shape[1])*initialGamma
    x = np.zeros((coeffMat.shape[1],1)) # need to correct for multi-dim B
    for i in range(len(coeffMat[:,0])):
        a = coeffMat[i,:]
        b = np.array(rhsMat[i])
        S = S - (np.array(np.dot(np.dot(np.dot(S,np.matrix(a).transpose()),np.mat
        x = x + (np.dot(S,np.dot(np.matrix(a).transpose(),(np.matrix(b)-np.dot(np
    return x
```

Рисунок 3.5 – Функция, реализующая метод наименьших квадратов

Рассмотрим функцию обучения сети ANFIS, часть которой отображена на рисунке 3.6. Аргументами функции являются: количество итераций обучения, критерий ошибки, при достижении которого останавливается обучение, гамма для метода наименьших квадратов, размер шага для

градиентного спуска - длина каждого градиентного перехода в пространстве параметров.

В начале устанавливается тип обучения, переменная для остановки обучения, переменная для номера итерации. Затем запускается основной цикл обучения, который будет работать пока не выполнит указанное количество итераций или пока не будет достигнут критерий остановки. Находим матрицу A, с помощью функции, проходящей по системе нечеткого вывода.

С помощью функции, реализующей метод наименьших квадратов, получаем список с параметрами заключения, который преобразуем в «numpy»-массив, после чего записываем его в поле класса для дальнейшего использования при прогнозе.

Вычисляем выход сети для это производим скалярное умножение матрицы A и матрицы, состоящей из одного столбца со значениями параметров заключения.

Во время умножения матриц, параметры заключения умножаются на соответствующие степени истинности, перемноженные с x1 и x2, после чего полученные значения складываются и получается выходное значения для данного входного набора. В итоге получается список, состоящий из выходных значений для каждого входного набора переменных. После этого происходит вычисление ошибки – сумма квадратов ошибок.

Для этого суммируется квадрат разности значения y из обучающей выборки и значения, выданного сетью. Текущая ошибка выводится в консоль. Вычисляется средняя абсолютная ошибка (MAE), для этого находится среднее из модулей разности значений y и значений выхода сети.

В поле объекта – массив ошибок добавляется текущая ошибка. Затем происходит проверка: получилась ли последняя вычисленная ошибка меньше критерия ошибки, если да, то обучение останавливается в начале следующей итерации.

```
def trainHybridJangOffLine(self, epochs=5, tolerance=1e-5, initialGamma=1000, k=0.01):

    self.trainingType = 'trainHybridJangOffLine'
    convergence = False
    epoch = 1

    while (epoch < epochs) and (convergence is not True):

        #layer four: forward pass
        [layerFour, wSum, w] = forwardHalfPass(self, self.X)

        #layer five: least squares estimate
        layerFive = np.array(self.LSE(layerFour,self.Y,initialGamma))
        self.consequents = layerFive
        layerFive = np.dot(layerFour,layerFive)

        #error
        error = np.sum((self.Y-layerFive.T)**2)
        print('current error: '+ str(error))
        average_error = np.average(np.absolute(self.Y-layerFive.T))
        self.errors = np.append(self.errors,error)

        if len(self.errors) != 0:
            if self.errors[len(self.errors)-1] < tolerance:
                convergence = True
```

Рисунок 3.6 – Часть функции обучения сети ANFIS

Перед тем как дальше рассматривать функцию обучения необходимо разобрать функцию, реализующую метод обратного распространения ошибки в сети, первая часть кода которой отображена на рисунке 3.7. Алгоритм обратного распространения ошибки используется в гибридной сети ANFIS для того, чтобы корректировать параметры функций принадлежности. Для того чтобы выполнить корректировку необходимо вычислить ошибку и провести её через предыдущие слои нейронов. Для поправки коэффициентов с целью уменьшения ошибки используется градиентный спуск. Для разработки функции обучения, реализующей градиентный спуск через пространство параметров, необходимо вычислить коэффициент ошибок $\frac{\partial E_p}{\partial O}$ для p наборов обучающих данных и для каждого выходного узла O. Коэффициент погрешности для выходного узла (L, i) можно вычислить по формуле: $\frac{\partial E_p}{\partial {O_{i,p}}^L} = -2(T_{i,p} - {O_{i,p}}^L)$, где T – вектор

значений y из обучающей выборки, O – вектор значений, полученный с помощью прямого прохода сети. Для узлов (k, i) внутренних слоев расчет происходит по следующему цепному правилу: $\frac{\partial E_p}{\partial {O_{i,p}}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial {O_{m,p}}^{k+1}} \frac{\partial {O_{m,p}}^{k+1}}{\partial {O_{i,p}}^k}$, где #(k+1) количество узлов зависимых от параметра предпосылки, $1 \le k \le L-1$. После вычисления поправок для каждого входного набора они суммируются. Рассмотрим код функции.

```
def backprop(ANFISObj, columnX, columns, theWSum, theW, theLayerFive):

    paramGrp = [0]* len(ANFISObj.memFuncs[columnX])
    for MF in range(len(ANFISObj.memFuncs[columnX])):

        parameters = np.empty(len(ANFISObj.memFuncs[columnX][MF][1]))
        timesThru = 0
        for alpha in sorted(ANFISObj.memFuncs[columnX][MF][1].keys()):

            bucket3 = np.empty(len(ANFISObj.X))
            for rowX in range(len(ANFISObj.X)):
                varToTest = ANFISObj.X[rowX,columnX]
                tmpRow = np.empty(len(ANFISObj.memFuncs))
                tmpRow.fill(varToTest)

                bucket2 = np.empty(ANFISObj.Y.ndim)
                for colY in range(ANFISObj.Y.ndim):

                    rulesWithAlpha = np.array(np.where(ANFISObj.rules[:,columnX]==MF))[0] #Номера правил,
                    adjCols = np.delete(columns,columnX) # Удаление обрабатываемого столбца

                    senSit = partial_dMF(ANFISObj.X[rowX,columnX],ANFISObj.memFuncs[columnX][MF],alpha)
                    # produces d_ruleOutput/d_parameterWithinMF
                    dW_dAplha = senSit * np.array([np.prod([ANFISObj.memClass.evaluateMF(tmpRow)[c][ANFIS(
```

Рисунок 3.7 – Первая часть кода функции, реализующей обратное распространение ошибки в сети

Аргументами функции являются: объект класса ANFIS, один из столбцов матрицы выборки (x1 или x2), номера всех столбцов, массив со суммой степеней истинности правил для всех входных строк, массив со степенями истинности всех правил для все входных строк, значения выходного слоя нейронной сети. Задаётся список под вычисляемые для параметров предпосылок коэффициенты погрешности размером равный количеству функций принадлежности для переменной заданного столбца (x1 или x2). Далее начинается цикл по всем функциям принадлежности для

переменной заданного столбца. Объявляется массив под вычисляемые параметры функций принадлежности, для Гауссовой функции это «sigma» и «mean». Далее запускаем цикл, в котором по очереди высчитываем поправки для этих параметров. Создаем массив, который будет использоваться для вычисления поправок, размером равным количеству входных строк из обучающей выборки. Запускаем цикл по этим строкам. В переменную «varToTest» записывается очередное значение параметра из перебираемых строк обучающей выборки и указанного столбца. Создаем массив размером с количество наборов функций принадлежности и заполняем его значениями «varToTest», так как функция вычисляющая степень принадлежности принимает массив, размер которого равен количеству входных переменных, для примера 2 (x1, x2). Создаем еще один массив размером с количество переменных y, в нашем случае – размером 1 элемент. Проходимся в цикле по переменным y, в нашем случае только одна итерация, так как в каждом входном наборе только один y. Выписываем в массив номера правил в которых условия под номером обрабатываемого столбца равны номеру рассматриваемой функции принадлежности, то есть выписываем номера правил, которые зависят от функции принадлежности, а значит от параметра предпосылки для которого вычисляется поправка. Из списка с номерами всех столбцов удаляем номер рассматриваемого. Затем вычисляем значение частной производной функции принадлежности по рассматриваемому параметру в точке x. Вычисления производятся по следующей формуле: $\Delta a = -a\frac{\partial E}{\partial a} = -a\sum(\frac{\partial E}{\partial e} * \sum(\frac{\partial e}{\partial y} * \sum(\frac{\partial y}{\partial(\overline{\mu}_i f_i)} * \sum(\frac{\partial(\overline{\mu}_i f_i)}{\partial \overline{\mu}_i} * \sum(\frac{\partial \overline{\mu}_i}{\partial \mu_i} * \sum(\frac{\partial \mu_i}{\partial \mu_{Ai}} * \frac{\partial \mu_{Ai}}{\partial a}))))))$ вычисляя справа налево. Значение частной производной уже получено, $\frac{\partial \mu_i}{\partial \mu_{Ai}} = \mu_{Bi}$, то есть берем противоположные условия правил содержащих рассматриваемый параметр и вычисляем функции принадлежности условий в точке x. Затем по формуле умножаем значения функций на производную, найденную ранее и суммируем получившиеся

значения по количеству узлов зависящих от параметра. Перейдем к рассмотрению второй части кода функции, отображенной на рисунке 3.8.

Создаётся массив, размер которого равен количеству правил. Затем запускается цикл в диапазоне значений от нуля до количества правил. Производная $\frac{\partial(\overline{\mu}_i f_i)}{\partial \overline{\mu}_i}$ равна значению параметров заключения умноженных на входные переменных x: $(c_{io} + c_{i1} * x_1 + c_{i2} * x_2)$. Данная производная в точке (x1, x2) представляет из себя число, поэтому можно внести это значение под следующий знак суммы. Частная производная $\frac{\partial y}{\partial(\overline{\mu}_i f_i)}$ равна 1. Поэтому в следующей строке вычисляем необходимое значение для очередного правила. Производная $\frac{\partial(\overline{\mu}_i f_i)}{\partial \overline{\mu}_i} = \frac{W_{sum} - w_i}{(W_{sum})^2}$. Так как на предыдущем слое от параметра зависели только 4 правила, то проверяем есть ли номер рассматриваемого правила в списке с правилами, зависящими от параметра функции принадлежности, если есть, то умножаем соответствующее из массива из предыдущего шага на W_{sum} – сумму степеней истинности правил. Затем производим вычисления по формуле, а также умножение на вычисленный множитель. Такие действия проводим по всем правилам, и в конце суммируем полученные значения. Затем происходи умножение суммы на коэффициент погрешности с выходного слоя: $-2 * (Y - O)$, где это значение из выборки, а O – значение, выданное сетью при прямом проходе. После прохождения цикла по всем входным наборам, полученные значения суммируются, эта сумма записывается в массив и происходит обработка следующего параметра предпосылки для этой функции принадлежности. Так находятся суммы для всех параметров предпосылки всех функций принадлежности указанного столбца. Функция возвращает массив со всеми поправками параметров.

```
            bucket1 = np.empty(len(ANFISObj.rules[:,0]))
            for consequent in range(len(ANFISObj.rules[:,0])):
                fConsequent = np.dot(np.append(ANFISObj.X[rowX,:],1.),ANFISObj.consequents[((A
                                                    (((ANFISObj.X.shape[1] + 1) * consequent)
                acum = 0
                if consequent in rulesWithAlpha:
                    acum = dW_dAplha[np.where(rulesWithAlpha==consequent)] * theWSum[rowX]

                acum = acum - theW[consequent,rowX] * np.sum(dW_dAplha)
                acum = acum / theWSum[rowX]**2
                bucket1[consequent] = fConsequent * acum

            sum1 = np.sum(bucket1)

            if ANFISObj.Y.ndim == 1:
                bucket2[colY] = sum1 * (ANFISObj.Y[rowX]-theLayerFive[rowX,colY])*(-2)
            else:
                bucket2[colY] = sum1 * (ANFISObj.Y[rowX,colY]-theLayerFive[rowX,colY])*(-2)

        sum2 = np.sum(bucket2)
        bucket3[rowX] = sum2

    sum3 = np.sum(bucket3)
    parameters[timesThru] = sum3
    timesThru = timesThru + 1

paramGrp[MF] = parameters

urn paramGrp
```

Рисунок 3.8 – Вторая часть кода функции, реализующей обратное распространение ошибки в сети

Вернемся к функции обучения и рассмотрим часть, где производится обратное распространение ошибки с помощью функции, рассмотренной ранее. Код, реализующий эти действия представлен на рисунке 3.9. Проверяем параметр сходимости, который используется для остановки обучения. Если обучение продолжается тогда создаем список номеров столбцов из обучающей выборки. После чего запускаем функцию обратного распространения для каждого столбца и получаем список из всех поправок параметров предпосылки. Далее идут участки кода, изменяющие размер шага для градиентного спуска. Далее с помощью нескольких циклов выписываем все поправки параметров в один список. И рассчитываем показатель скорости обучения по формуле: делим размер шага k на модуль суммы всех поправок параметров. Если в результате деления получена бесконечность, то

скорость устанавливается равной размеру шага. Далее находятся реальные поправки параметров, путем умножения на полученную скорость обучения, в зависимости от того были ли равные количества функций принадлежности для входных переменных вычисления производятся по-разному.

```
# back propagation
if convergence is not True:
    cols = range(len(self.X[0,:]))
    dE_dAlpha = list(backprop(self, colX, cols, wSum, w, layerFive) for colX in ran


if len(self.errors) >= 4:
    if (self.errors[-4] > self.errors[-3] > self.errors[-2] > self.errors[-1]):
        k = k * 1.1

if len(self.errors) >= 5:
    if (self.errors[-1] < self.errors[-2]) and (self.errors[-3] < self.errors[-2])
        k = k * 0.9

t = []
for x in range(len(dE_dAlpha)):
    for y in range(len(dE_dAlpha[x])):
        for z in range(len(dE_dAlpha[x][y])):
            t.append(dE_dAlpha[x][y][z])

eta = k / np.abs(np.sum(t))

if(np.isinf(eta)):
    eta = k

## handling of variables with a different number of MFs
dAlpha = copy.deepcopy(dE_dAlpha)
if not(self.memFuncsHomo):
    for x in range(len(dE_dAlpha)):
        for y in range(len(dE_dAlpha[x])):
            for z in range(len(dE_dAlpha[x][y])):
                dAlpha[x][y][z] = -eta * dE_dAlpha[x][y][z]
else:
    dAlpha = -eta * np.array(dE_dAlpha)
```

Рисунок 3.9 – Вторая часть функции обучения сети ANFIS

Перейдем к заключительной части функции обучения, отображенной на рисунке 3.10. Здесь производится добавления к параметрам функций принадлежности поправок. Для этого цикл проходит по всем функциям принадлежности и по всем параметрам и добавляет к ним соответствующие им вычисленные поправки. После этого происходит инкрементация счетчика

итераций. После обучения сети рассчитываются выходы обученной сети и итоговые ошибки.

```
        for varsWithMemFuncs in range(len(self.memFuncs)):
            for MFs in range(len(self.memFuncsByVariable[varsWithMemFuncs])):
                paramList = sorted(self.memFuncs[varsWithMemFuncs][MFs][1])
                for param in range(len(paramList)):
                    self.memFuncs[varsWithMemFuncs][MFs][1][paramList[param]] = self
        epoch = epoch + 1


    self.fittedValues = predict(self,self.X) # Значения, выдаваемые моделью
    self.residuals = self.Y - self.fittedValues[:,0]
```

Рисунок 3.10 – Третья часть функции обучения сети ANFIS

Рассмотрим функцию, использующуюся для нахождения частных производных функций принадлежности в точках, представленную на рисунке 3.11. Аргументами функции являются: точка, в которой необходимо вычислить производную, функция принадлежности, параметр функции по которому берется производная. Записываются параметры функции и вычисляется значение производной в точке.

```
def partial_dMF(x, mf_definition, partial_parameter):
    """Calculates the partial derivative of a membership function at a point x.
       Вычисляет частную производную функции принадлежности в точке x.
    """
    mf_name = mf_definition[0]

    if mf_name == 'gaussmf':

        sigma = mf_definition[1]['sigma']
        mean = mf_definition[1]['mean']

        if partial_parameter == 'sigma':
            result = (2./sigma**3) * np.exp(-(((x-mean)**2)/(sigma)**2))*(x-mean)**2
        elif partial_parameter == 'mean':
            result = (2./sigma**2) * np.exp(-(((x-mean)**2)/(sigma)**2))*(x-mean)
```

Рисунок 3.11 – Функция нахождения частной производной

На рисунке 3.12 представлены функции построения графиков. Первая функция строит график ошибок, которые были высчитаны после каждой итерации обучения. Он строится для того, чтобы можно было отследить снижение ошибки после каждой итерации обучающей функции.

```
def plotErrors(self):
    if self.trainingType == 'Not trained yet':
        print(self.trainingType)
    else:
        plt.plot(range(len(self.errors)),self.errors,'ro', label='errors')
        plt.ylabel('error')
        plt.xlabel('epoch')
        plt.show()

def plotMF(self, x, inputVar):
    for mf in range(len(self.memFuncs[inputVar])):
        if self.memFuncs[inputVar][mf][0] == 'gaussmf':
            y = gaussmf(x,**self.memClass.MFList[inputVar][mf][1])
        elif self.memFuncs[inputVar][mf][0] == 'gbellmf':
            y = gbellmf(x,**self.memClass.MFList[inputVar][mf][1])
        elif self.memFuncs[inputVar][mf][0] == 'sigmf':
            y = sigmf(x,**self.memClass.MFList[inputVar][mf][1])

        plt.plot(x,y,'r')

    plt.show()

def plotResults(self):
    if self.trainingType == 'Not trained yet':
        print(self.trainingType)
    else:
        plt.plot(range(len(self.fittedValues)),self.fittedValues,'r', label='trained')
        plt.plot(range(len(self.Y)),self.Y,'b', label='original')
        plt.legend(loc='upper left')
        plt.show()
```

Рисунок 3.12 – Функции построения графиков

Вторая функция используется для построения графика функции принадлежности. Третья функция строит результирующий график, на котором отображены исходная функция и построенная по данным обученной сети. Он строится для того, чтобы проанализировать насколько успешно обучена сеть.

3.2 Пример работы

Пример использования разработанного класса представлен на рисунке 3.13. С помощью библиотеки «numpy» загружаем обучающую выборку из файла «trainingSet.txt» и распределяем её на массивы с x и y.

Задаем список функций принадлежности. Создаем объект класса «MemFuncs», создаем объект класса «ANFIS». Запускаем функцию обучения и указываем количество итераций равное 20. Выводим результирующие графики.

```
ts = np.loadtxt("trainingSet.txt", usecols=[1,2,3])#numpy.loadtxt('c:\\Python_fidc
X = ts[:,0:2]
Y = ts[:,2]

mf = [
      [
         ['gaussmf',{'mean':0.,'sigma':1.}],
         ['gaussmf',{'mean':-1.,'sigma':2.}],
         ['gaussmf',{'mean':-4.,'sigma':10.}],
         ['gaussmf',{'mean':-7.,'sigma':7.}]
      ],
      [
         ['gaussmf',{'mean':1.,'sigma':2.}],
         ['gaussmf',{'mean':2.,'sigma':3.}],
         ['gaussmf',{'mean':-2.,'sigma':10.}],
         ['gaussmf',{'mean':-10.5,'sigma':5.}]
      ]
]

mfc = MemFuncs(mf)
anf = ANFIS(X, Y, mfc)
anf.trainHybridJangOffLine(epochs=20)
print(round(anf.consequents[-1][0],6))
print(round(anf.consequents[-2][0],6))
print(round(anf.fittedValues[9][0],6))
if round(anf.consequents[-1][0],6) == -5.275538 and round(anf.consequents[-2][0],6
    print('test is good')
anf.plotErrors()
anf.plotResults()
```

Рисунок 3.13 – Функции построения графиков

Проведем запуск программы и посмотрим на результаты работы. После каждой итерации обучения программа выводит текущую ошибку работы ошибку на обучающей выборке (рисунок 3.14)

```
current error: 0.32197133130238204
current error: 0.32064969677854194
current error: 0.31935983596568546
current error: 0.3180964819660493
current error: 0.31673125060860485
current error: 0.3152546970544982
current error: 0.3136564810260567
current error: 0.3119254351800986
current error: 0.3100497693048592
current error: 0.30801749864651917
current error: 0.3058172226732122
current error: 0.3034394205128306
current error: 0.3008784504795888
current error: 0.29813538447505605
current error: 0.29522149867151354
current error: 0.2921610523482191
current error: 0.2889868451587407
current error: 0.2856912569174573
current error: 0.28184647973587823
```

Рисунок 3.14 – Результат работы программы

На рисунках 3.15 и 3.16 представлены графики, выводимые программой. На первом графике отображено изменения ошибки нейро-нечеткой сети после каждой итерации (эпохи) обучения. На втором графике представлены функция, построенная по обучающей выборки и функция, построенная по выходным данным обученной сети.

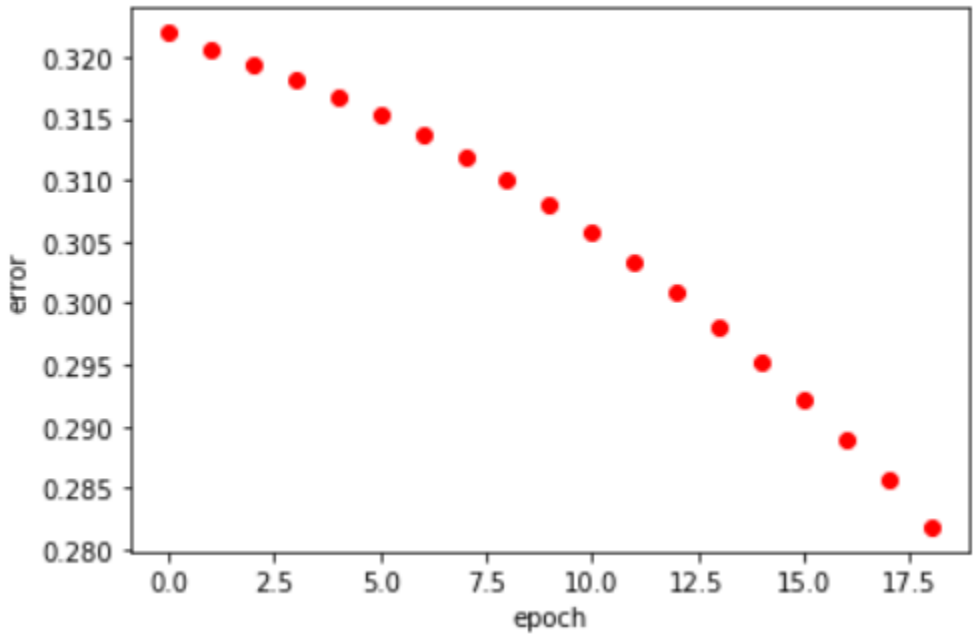

Рисунок 3.15 – График изменения ошибки (error) работы сети ANFIS в зависимости от номера итерации обучения (epoch)

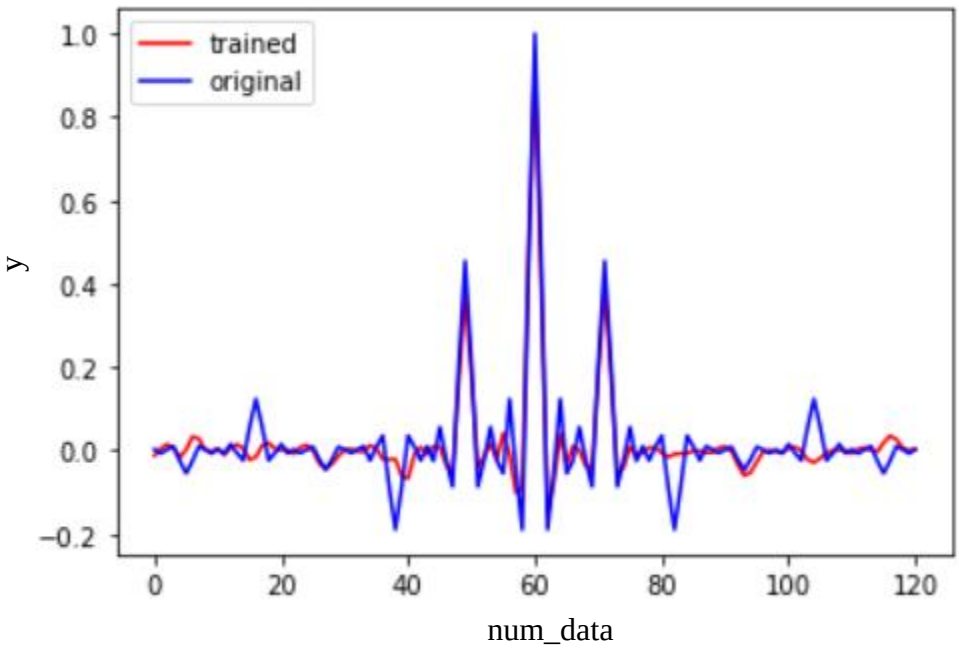

Рисунок 3.16 – График визуального сравнения описания данных обучающей выборки (синяя линии) обученной сетью ANFIS (красная линия): y – значение независимой переменной, num_data – номер записи в обучающей выборке

Как видно из рисунка 3.16 сеть ANFIS может успешно обучиться описывать данные представленные в обучающей выборке. Это значит, что созданный на языке Python программный класс для моделирования нейронной сети ANFIS работает правильно.

Таким образом, на языке Python был реализован класс, позволяющей моделировать работу, обучение и тестирование ANFIS сети. Данный класс можно подключать к другим проектам, написанным на языке Python или использовать его независимо при проведении научно-исследовательских работ.

ЗАКЛЮЧЕНИЕ

Исследования, проведенные в рамках данной бакалаврской работы, позволяют сделать следующие выводы:

1. Обзор отечественных и зарубежных источников литературы показывает, что одним из перспективных направлений развития искусственного интеллекта является комбинирование технологий с целью получения новых алгоритмов машинного обучения. Так комбинирование теории нечетких множеств и нейросетевых технологий привело к появлению нейро-нечетких сетей ANFIS.

2. Результаты сравнительного анализа библиотек Python по машинному обучению (Scikit-learn, Tensorflow, Keras, pyTorch), показал, что в них реализован функционал по работе с нейронными сетями прямого распространения, но отсутствует реализация функций для моделирования нейро-нечетких сетей ANFIS.

3. Был рассмотрен математический аппарата нейро-нечетких сетей ANFIS, описаны принципы их работы, а также показана процедура настройки сети.

4. На языке Python был реализован класс, позволяющей моделировать работу, обучение и тестирование ANFIS сети. Данный класс можно подключать к другим проектам, написанным на языке Python или использовать его независимо при проведении научно-исследовательских работ.

5. Проведенное тестирование на имеющейся выборке данных подтвердило работоспособность разработанного программного обеспечения. А по графикам представленным в третье граве можно определить насколько точно сеть ANFIS описывает данные обучающей выборки с двумя независимыми переменными.

Таким образом, цель данной бакалаврской работы была достигнута, а все поставленные задачи успешно решены.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Гатин, Р.Р. Функции и возможности конструктора ANFIS для проектирования нечетких нейронных сетей / Р.Р. Гатин, Р.Р. Бикмухаметов // Наука сегодня: Реальность и перспективы, Материалы международной научно-практической конференции. – Вологда : Издатель ООО "Маркер", 2019. С. 11-13. Текст : непосредственный.

2. Назаркин, О.А. Повышение эффективности параллельного обучения ансамблей аппроксиматоров на основе ненормализованного варианта моделей ANFIS / О.А. Назаркин, П.В. Сараев // Суперкомпьютерные технологии (СКТ-2016) Дивноморское, 19-24 сентября 2016 г. Материалы 4-й всероссийской научно-технической конференции. – Ростов-на-Дону : Издатель Южный федеральный университет, 2016. С. 184-188. Текст : непосредственный.

3. Аверкин, А.Н. Гибридный подход для прогнозирования временных рядов на основании нейросети ANFIS и нечетких когнитивных карт / А.Н. Аверкин, С.А. Ярушев // Международная конференция по мягким вычислениям и измерениям. – Санкт-Петербург : Издатель Санкт-Петербургский государственный электротехнический университет "ЛЭТИ" им. В.И. Ульянова (Ленина) (Санкт-Петербург), 2017. С. 467-470. Текст : непосредственный.

4. Грищенко, И.А. Исследование влияния формы входных функций принадлежности на результат работы системы нейро-нечеткого вывода ANFIS / И.А. Грищенко, В.И. Иванчура // Роль технических наук в развитии общества Сборник материалов Международной научно-практической конференции. Западно-Сибирский научный центр; Кузбасский государственный технический университет имени Т.Ф. Горбачева. 2015. – Кемерово : Издатель Кузбасский государственный технический университет имени Т.Ф. Горбачева, 2015. С. 25-29. Текст : непосредственный.

5. Солдатова, О.П. Решение задачи прогнозирования с использованием нейросети ANFIS / О.П. Солдатова, Д.И. Кривякин // Перспективные информационные технологии (ПИТ 2017) труды Международной научно-технической конференции. 2017. – Самара : Издатель Самарский научный центр РАН, 2017. С. 402-404. Текст : непосредственный.

6. Булыгина, О.В. Управление инновационными рисками в теплоэнерегетике с использованием гибридных ANFIS-сетей / О.В. Булыгина, С.М. Дли // Энергетика, информатика, инновации – 2015, Сборник трудов V Международной научно-технической конференции: в 2 томах. Национальный исследовательский университет «МЭИ» (Московский энергетический институт), филиал в г. Смоленске. 2015. – Смоленск : Издатель Универсум, 2015. С. 95-98. Текст : непосредственный.

7. Сараев, П.В. Автоматизация выбора структуры ANFIS на основе кластеризации входных переменных / П.В. Сараев, М.Г. Журавлева, П.А. Домашнев // Современные методы прикладной математики, теории управления и компьютерных технологий (ПМТУКТ-2016) Сборник трудов IX международной конференции. 2016. – Воронеж : Издатель ООО "Издательство "Научная книга", 2016. С. 302-303. Текст : непосредственный.

8. Елистратов, В.В. Адаптивная нейро-нечеткая система вывода (ANFIS) для классификации неисправностей в линиях электропередачи / В.В. Елистратов // Модернизация и инновационное развитие топливно-энергетического комплекса материалы международной научно-практической конференции. 2019. – Санкт-Петербург : Издатель Санкт-Петербургский филиал научно-исследовательского центра "МашиноСтроение", 2019. С. 76-82. Текст : непосредственный.

9. Чульдум, А.Ф. Пример создания C# .NET приложения для прогнозирования временного ряда с использованием адаптивной нейро-нечеткой системы вывода-ANFIS / А.Ф. Чульдум, У.А. Чульдум // Информатизация образования: история, проблемы и перспективы сборник

материалов Всероссийской научно-практической конференции, посвященной 70-летию со дня рождения первого ректора Тувинского государственного университета О.Б. Бузур-оола. 2016. – Кызыл : Издатель: Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Тувинский государственный университет", 2016. С. 38-41. Текст : непосредственный.

10. Поваров, В.П. Интеграция систем принятия решений в структуру верхнего уровня управления АЭС / В.П. Поваров, Д.В. Терехов, А.Д. Данилов // Альтернативная и интеллектуальная энергетика Материалы Международной научно-практической конференции. 2018. – Воронеж : Издатель Воронежский государственный технический университет, 2018. С. 54-55. Текст : непосредственный.

11. Akhilesh, K.M. Vehicle Classification Using Adaptive Neuro Fuzzy Inference System (ANFIS) / K.M Akhilesh, K.P Devesh // Proceedings of Fourth International Conference on Soft Computing for Problem Solving: Part of the Advances in Intelligent Systems and Computing book series (AISC, volume 336). – New Delhi : Springer India, 2015. – P. 137-152. – Text : direct.

12. Jaime, T. SERS and ANFIS: Fast Identification of the Presence of Retrovirus in CD4 Cells, Cause of AIDS / T. Jaime, L. Francisco, M. Julio, P. Alejandro, M. Miguel // Mexican International Conference on Artificial Intelligence: MICAI 2008: Advances in Artificial Intelligence,7th Mexican International Conference on Artificial Intelligence, Atizapán de Zaragoza, Mexico, October 27-31, 2008 Proceedings. – Wien : 1 Institut fur Informationssysteme (DBAI), TU Wien, Austria, 2008. – P. 936-947. – Text : direct.

13. Abhishek, K. M. Design of ANFIS Controller Based on Fusion Function for Linear Inverted Pendulum / K.M Akhilesh, R. Mitra // Proceedings of International Conference on Advances in Computing: Part of the Advances in Intelligent Systems and Computing book series (AISC, volume 174). – New Delhi : Springer India Indian Institute of Technology, Roorkee, Haridwar, 2013. – P. 379-386. – Text : direct.

14. Xiaoxu, L. Soil Moisture Retrieval Using UWB Echoes via ANFIS and ANN / L. Xiaoxu, Y. Xiaofeng, J. Ren, L. Jing // International Conference in Communications, Signal Processing, and Systems: Communications, Signal Processing, and Systems: Part of the Lecture Notes in Electrical Engineering book series (LNEE, volume 463). – Singapore : Springer Nature Singapore Pte Ltd, 2019. – P. 1261-1268. – Text : direct.

15. Dah-Jing, J. ANFIS Based Dynamic Model Compensator for Tracking and GPS Navigation Applications / Dah-Jing C. Zong-Ming // International Conference on Natural Computation, Advances in Natural Computation: Part of the Lecture Notes in Computer Science book series (LNCS, volume 3611). – Keelung : Department of Communications and Guidance Engineering, National Taiwan Ocean University, 20224 Keelung, Taiwan, 2005. – P. 425-431. – Text : direct.

16. Bedri, K. Hydraulic Head Interpolation in an Aquifer Unit Using ANFIS and Ordinary Kriging / K. Bedri, F. Nicolas, G. Patrick, V. Guillaume, J. Tournebize, T. Gaëlle // Computational Intelligence: Part of the Studies in Computational Intelligence book series (SCI, volume 343). – Berlin : Springer-Verlag Berlin Heidelberg, 2011. – P. 265-276. – Text : direct.

17. Rijun, Z. Application of the Wavelet-ANFIS Model / Z. Rijun, H. Caishui, L. Hui, Z. Meiyan, Z. Meixin, L. Zhongsheng, S. Liwu, L. Fengqin // Computer Engineering and Networking, Proceedings of the 2013 International Conference on Computer Engineering and Network (CENet2013): Part of the Lecture Notes in Electrical Engineering book series (LNEE, volume 277). – Cham : Springer International Publishing Switzerland, 2014. – P. 1373-1379. – Text : direct.

18. Khaled, A. A Hybrid Krill-ANFIS Model for Wind Speed Forecasting / A. Khaled, A. E. Ahmed, A. Mohamed, E.H. Aboul, G. Tarek, T. Pei-Wei, P. Jeng-Shyang // International Conference on Advanced Intelligent Systems and Informatics, Proceedings of the International Conference on Advanced Intelligent

Systems and Informatics 2016: Part of the Advances in Intelligent Systems and Computing book series (AISC, volume 533). – Cham : Springer International Publishing AG, 2017. – P. 365-372. – Text : direct.

19. Kyunghoon, J. Vision Guidance System for AGV Using ANFIS / J. Kyunghoon, L. Inseong, S. Hajun, K. Jungmin, K. Sungshin // International Conference on Intelligent Robotics and Applications, Intelligent Robotics and Applications: 5th International Conference, ICIRA 2012, Montreal, QC, Canada, October 3-5, 2012, Proceedings, Part I. – Berlin : Springer-Verlag Berlin Heidelberg, 2012 – 377-385. – Text : direct.

20. Engin, S. N. Modeling of a Coupled Industrial Tank System with ANFIS / S. N. Engin, J. Kuvulmaz, V. E. Ömürlü // Mexican International Conference on Artificial Intelligence, MICAI 2004: Advances in Artificial Intelligence: Third Mexican International Conference on Artificial Intelligence, Mexico City, Mexico, April 26-30, 2004. Proceedings. – Berlin : Springer-Verlag Berlin Heidelberg, 2004 – P. 804-812 – Text : direct.