Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационные системы и анализ данных

Центр программной инженерии

**ОТЧЁТ**

к лабораторной работе №3 по дисциплине:

|  |
| --- |
| «Нейросетевые технологии» |
| [Аппроксимация функций](https://el.istu.edu/mod/glossary/showentry.php?eid=207397&displayformat=dictionary) с помощью [нейрон](https://el.istu.edu/mod/glossary/showentry.php?eid=207468&displayformat=dictionary)ных сетей |
| наименование темы |

Выполнилстудент ИСТб-21-1 Д.И. Морозов

номер группы подпись И. О. Фамилия

дата

Проверил Доцент Е.А. Осипова

Должность подпись И. О. Фамилия

дата

Иркутск – 2023 г.

# 1 Постановка задачи

# Цель работы: изучение и программная реализация простейшей искусственной нейронной сети для решения задачи аппроксимации, а также её решение через ANFIS и RBF.

# ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Сгенерировать случайные числа по указанному закону распределения. Из них создается матрица из M строк (наблюдений) 1 или 2 столбцов (аргументов). Эти числа играют роль аргументов некоторой функциональной зависимости от одного или двух аргументов
2. Разделить данные варианта на обучающую и тестирующую выборки
3. Осуществить аппроксимацию функций f1 и f2 тремя способами:
   1. С помощью сети прямого распространения сигнала;
   2. С помощью RBF-сети;
   3. С помощью нейро-нечеткой сети ANFIS.
4. Построить графики ошибок и сравнения значений функции для обучающей и тестирующей выборки и выходных значений, прогнозированных нейронными сетями. Оценить количественно точность аппроксимации функций.
5. Для построения нейро-нечеткой сети ANFIS необходимо выполнить следующие шаги:
   1. По обучающей выборке сгенерировать систему нечеткого вывода и настроить эту систему
   2. Показать структуру построенной системы нечеткого вывода (входные и выходные переменные, функции принадлежности, правила)
   3. Просмотреть поверхность системы нечеткого вывода до и после настройки системы нечеткого вывода с помощью ANFIS
   4. Вычислить выходные значения по системе нечеткого вывода для обучающей и тестирующей выборок.
   5. Построить графики ошибок и сравнения значений функции для обучающей и контролирующей выборки и выходных значений по системе нечеткого вывода до настройки системы и после настройки.

# Вариант:

# Вид закона распределения “нормальный”.

# Вид функциональной зависимости от 1 аргумента “логарифмическая функция”.

# Вид функциональной зависимости от 2 аргументов “функция Бирда”.

# 2.1 Подготовка и проверка данных

Импорт библиотек для дальнейшей работы.

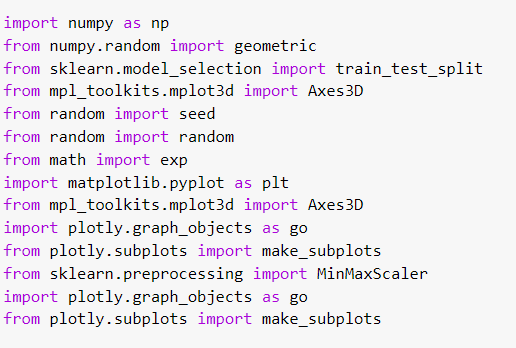


Рисунок 1 – Импорт библиотек

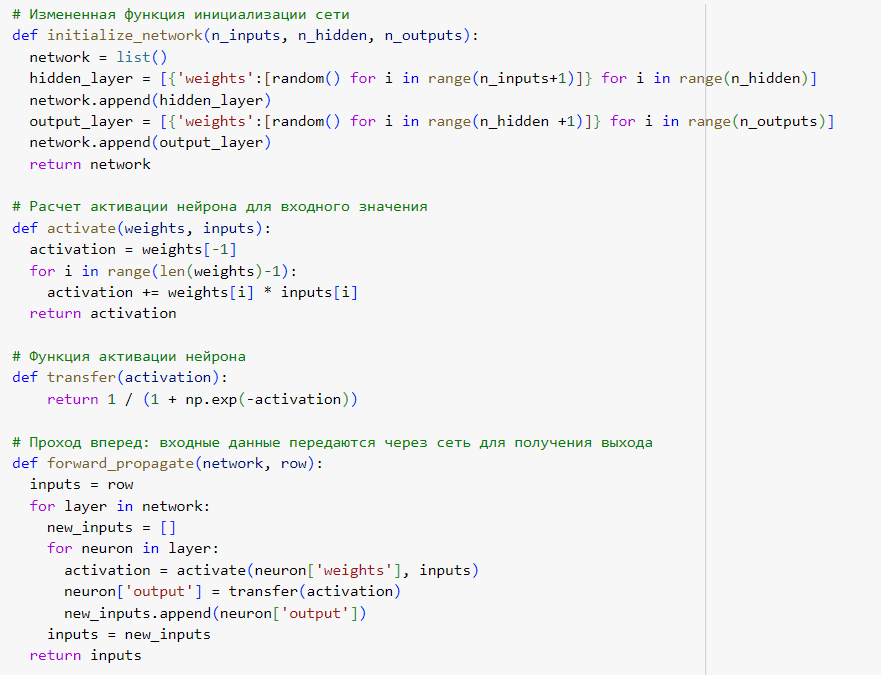
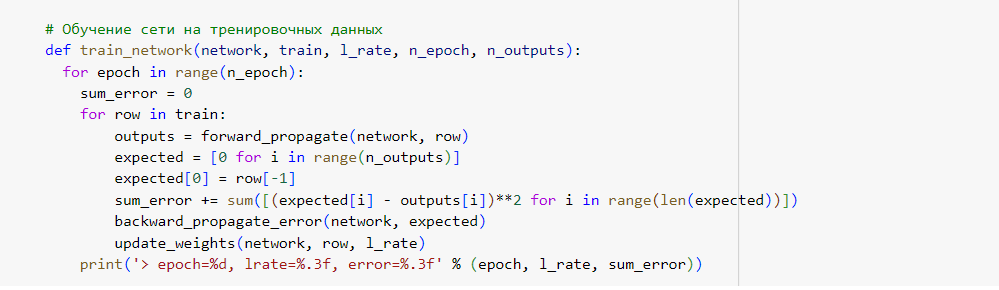
  

Рисунок 2 – Инициализация однослойной нейронной сети

На рисунке 2 описана нейронная сеть с сигмоидальной функцией активации, эта функция подходит для решения задачи

Для правильной работы нейронной сети, необходимо подобрать правильную функцию активации из перечня на рисунке 3, функция активации должна подходить под данные из задания.



Рисунок 3 – Функции активации

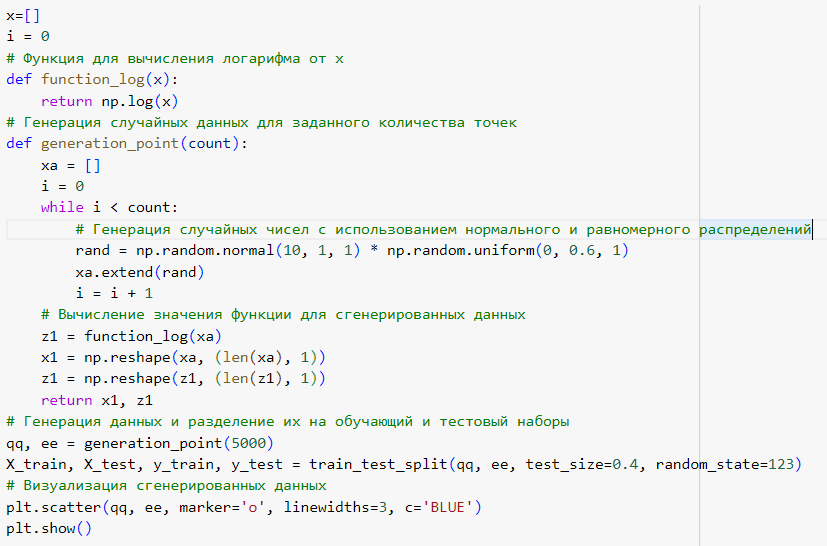


Рисунок 4 – Определение функции и генерация случайных данных

Используется нормальное распределение случайных значений и умножается на равномерное распределение для того, чтобы числа не были отрицательными.

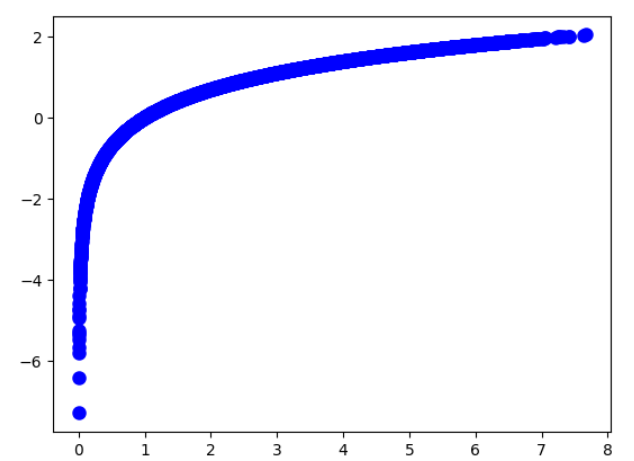
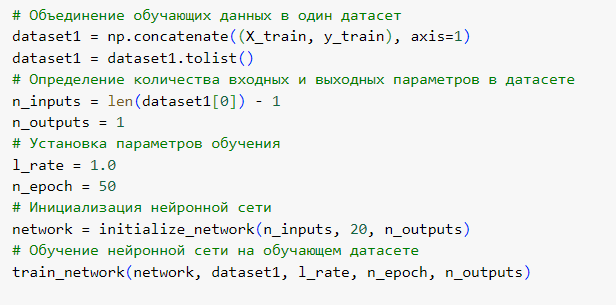


Рисунок 5 – График логарифмической функции

На данном графике видно, как ведет себя логарифмическая функция, необходимо обучить нейронную сеть, чтобы данные на выходе были похожи на логарифмическую функцию.



Поставлены параметры для обучения нейронной сети, увеличен l\_rate для корректной работы нейронной сети с логарифмической функцией.

Рисунок 6 – Инициализация сети и подбор параметров для её работы

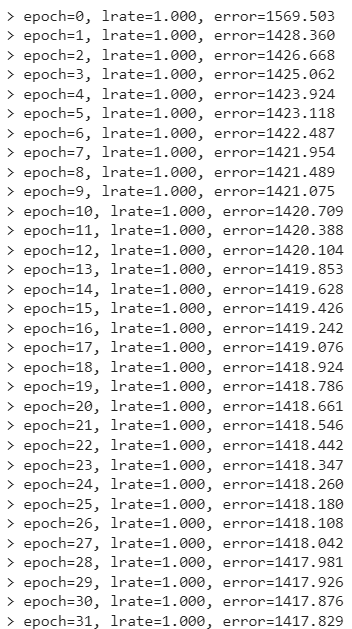


Рисунок 7 – Процесс обучения нейронной сети

По подобранным параметрам видно, что происходит обучение нейронной сети и количество ошибок уменьшается.

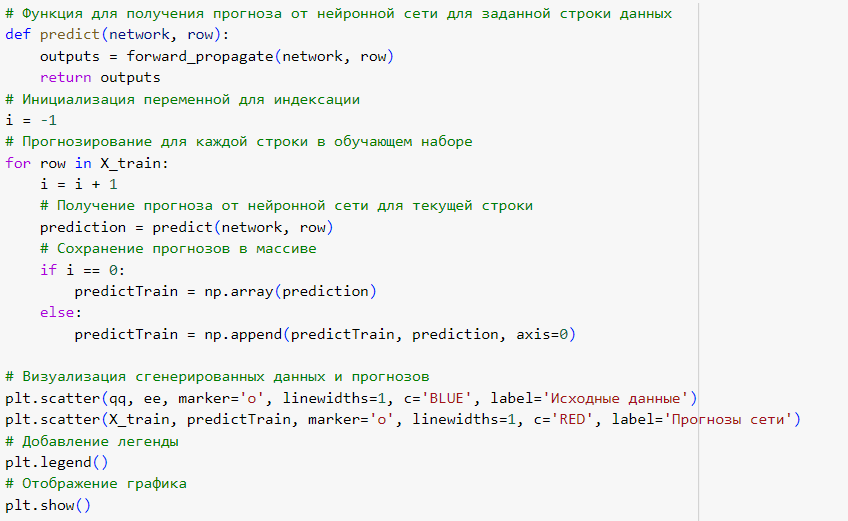


Рисунок 8 – Прогнозирование данных

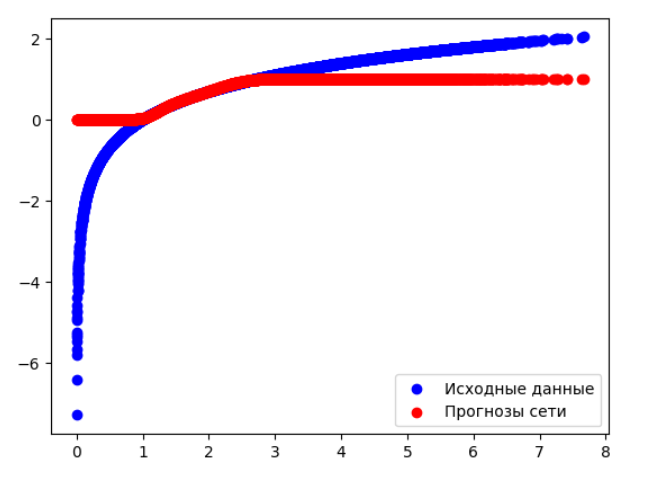


Рисунок 9 – График исходных данных и прогноз сети

На данном графике видно, что после определенного количества эпох обучение нейронной сети не происходит, необходимо изменять саму структуру сети либо добавлять новый функционал, либо увеличивать количество эпох.

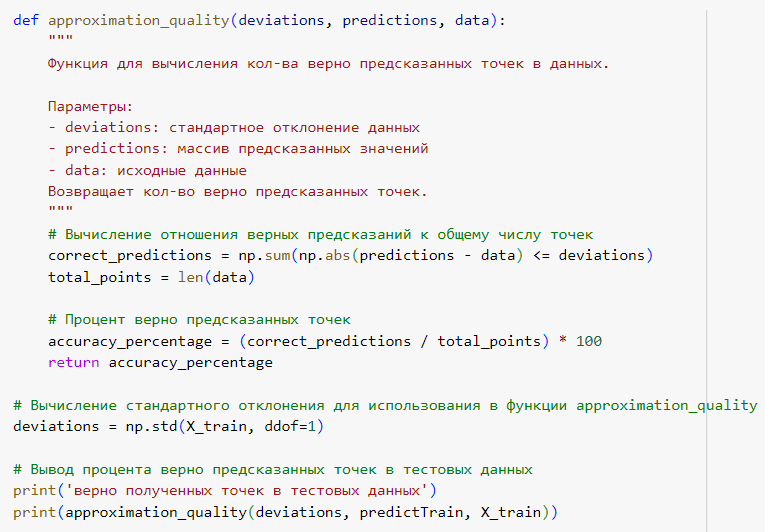


Рисунок 10 – Функция оценки аппроксимации



Для более точного результата необходимо менять разброс в функции аппроксимации.

Рисунок 11 – Результат функции оценки аппроксимации

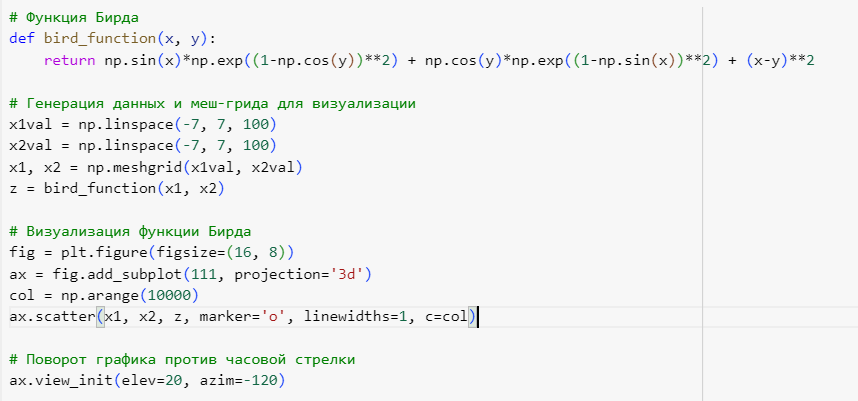


Рисунок 12 – Определение функции Бирда и код её визуализации

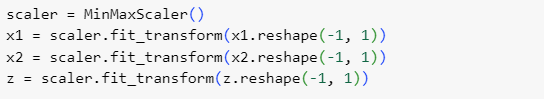


Рисунок 13 – Использование MinMaxScaler для подготовки данных подходящих под функцию

Для правильной работы нейронной сети с сигмоидальной функции активации, ей необходимы данные, находящиеся в интервале от 0 до 1, поэтому здесь используется MinMaxScaler для подготовки данных.

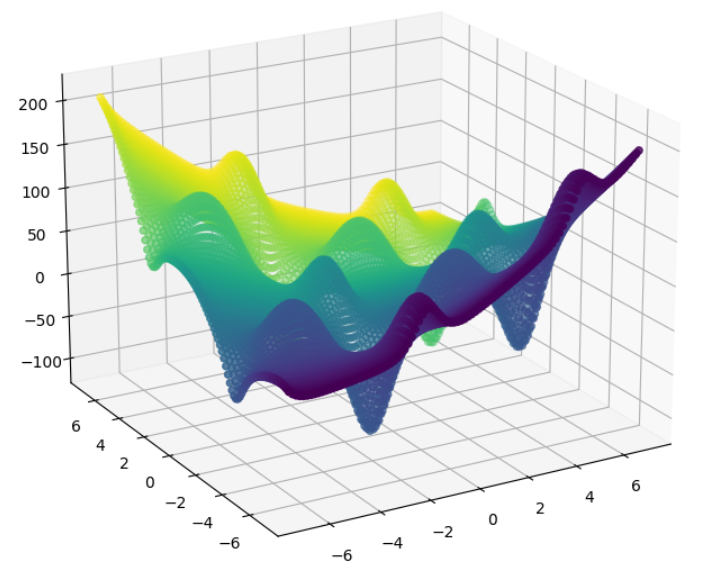


Рисунок 14 – Внешний вид функции Бирда

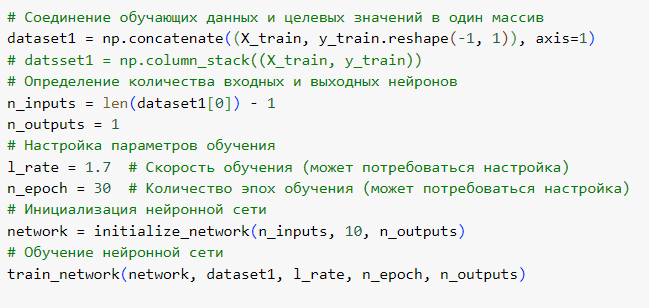


Рисунок 15 –Код для обучения нейронной сети для функции Бирда

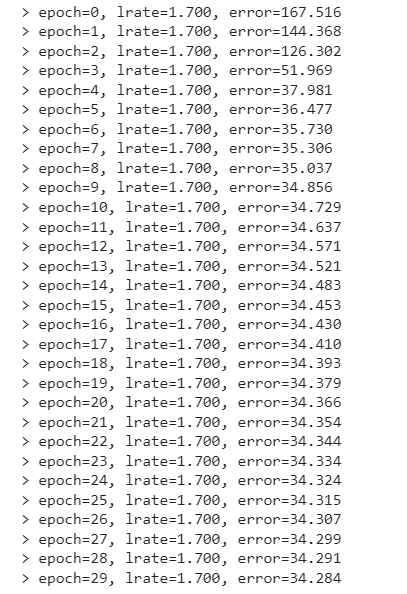


Рисунок 16 – Процесс обучения нейронной сети

На данном рисунке видно, как происходит обучение и количество ошибок уменьшается в разы.

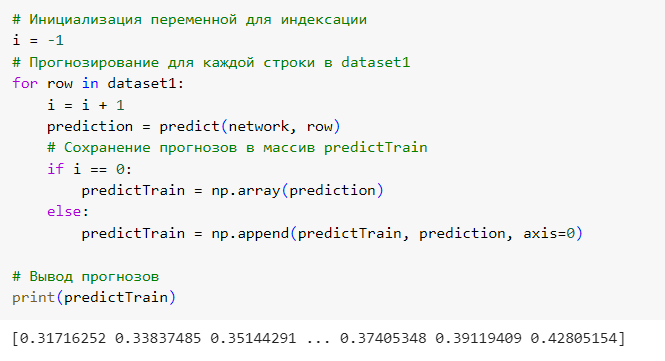


Рисунок 17 – Прогнозирование и вывод точек с прогнозом находящиеся в диапазоне от 0 до 1

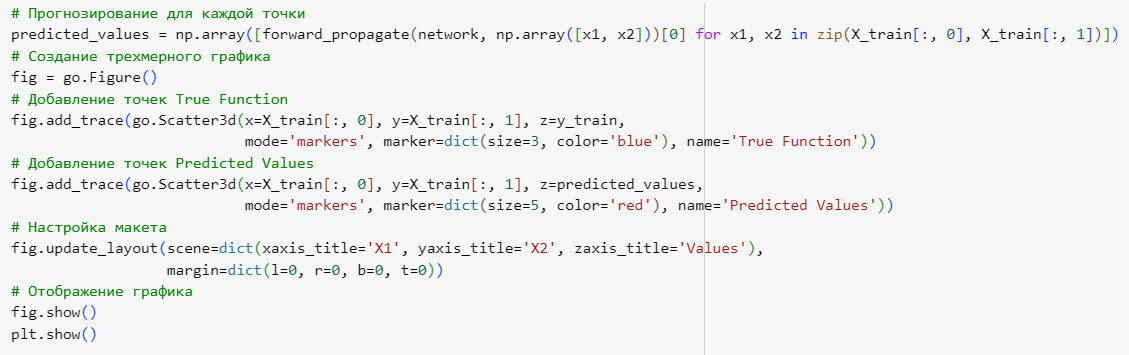


Рисунок 18 – Код визуализации предсказанных точек в 3d

Построение интерактивного 3д графика для более удобного визуального анализа данных.

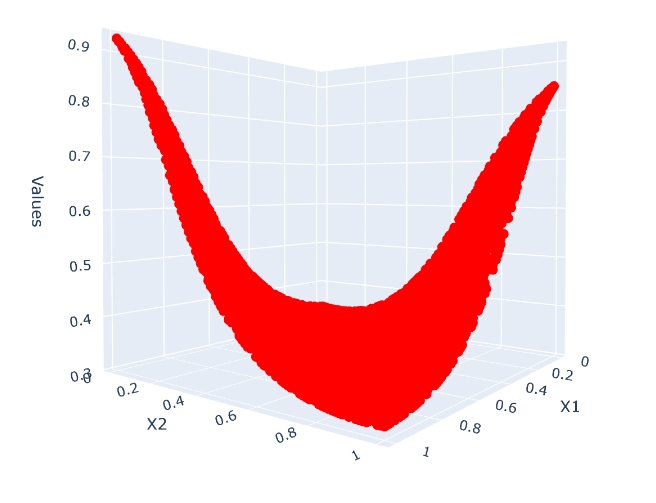


Рисунок 19 – Внешний вид графика обученных точек

На рисунке 19 видно, что данные обученные нейросетью повторяют форму построенную по функции Бирда, но они обучены не на 100%, так же необходимо увеличивать количество эпох, либо добавлять функционал в нейронную сеть для более корректного обучения.

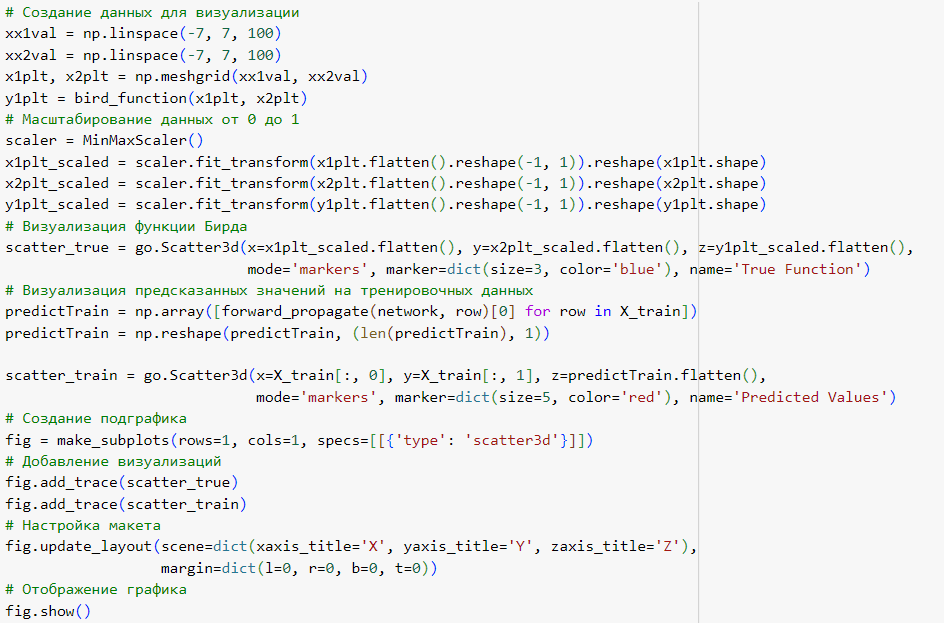


Рисунок 20 – Создание графика для сравнения исходных точек и предсказанных

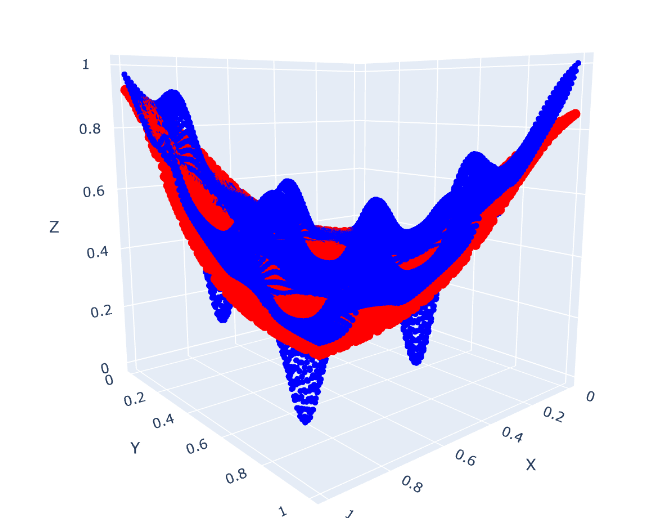


Рисунок 21 –График для сравнения исходных точек и предсказанных

На данном графике можно сравнить разные точки и их соответствие, а так же провести анализ данных.

2 ANFIS

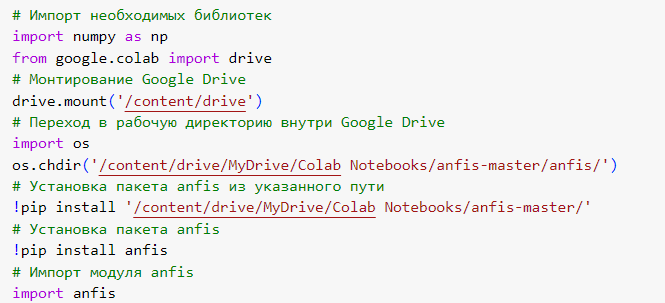


Рисунок 22 – Подключение необходимых библиотек для работы ANFIS

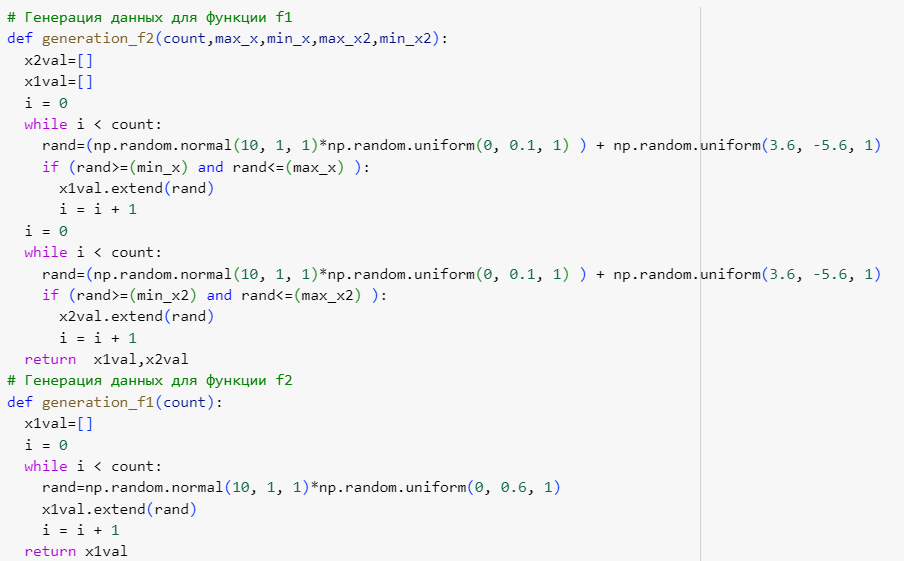


Рисунок 23 –Определение генераций данных для функций

Генерация данных по варианту для обучения нейронной сети ANFIS.

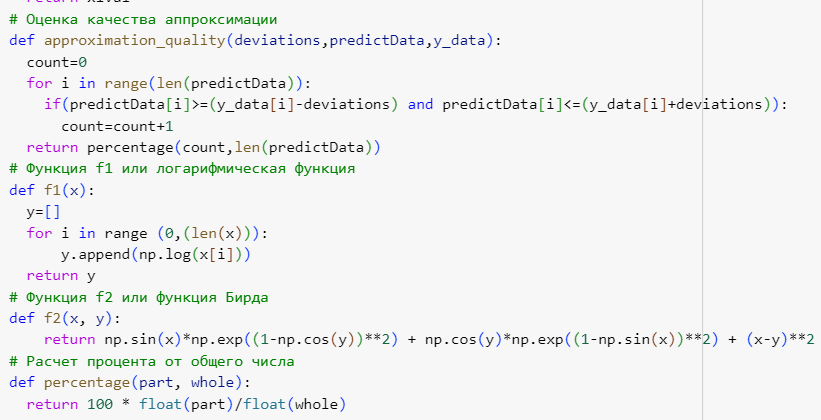


Рисунок 24 – Аппроксимация и определение функций по заданию



Рисунок 25 – Подготовка данных и сохранение их в текстовый файл

Сохранение в файл помогает в работе с нейронной сетью, для использования одних и тех же данных для обучения.

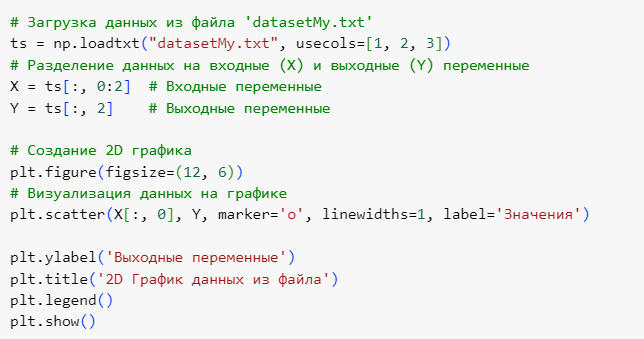


Рисунок 26 – Загрузка данных и её визуализация

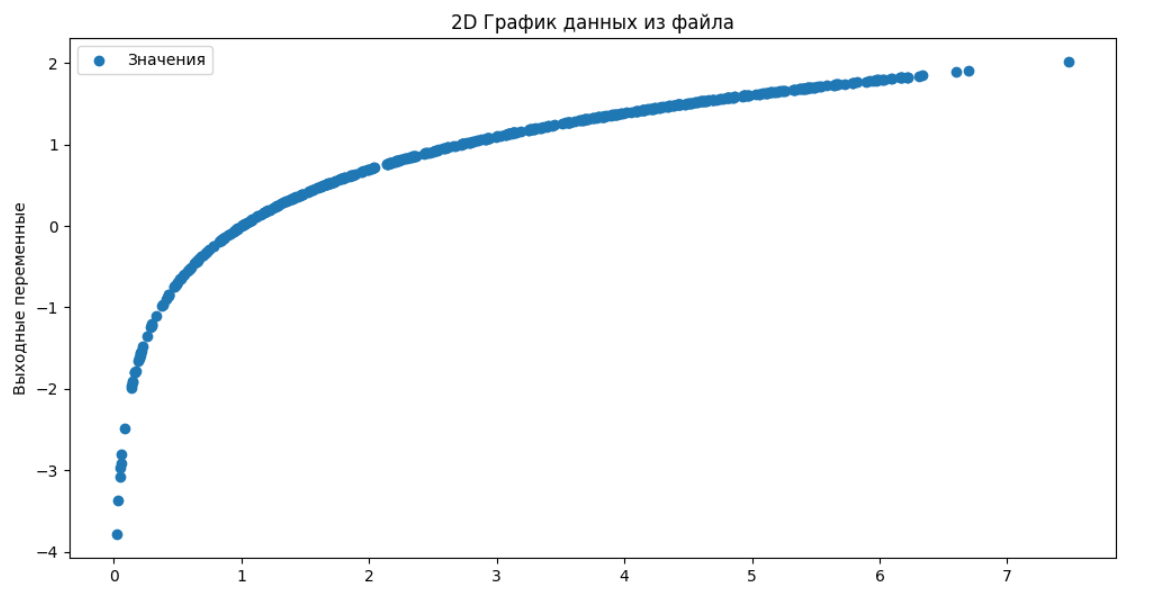


Рисунок 27 – График загруженных данных

На данном графике видна логарифмическая функция, воспроизведенная из текстового файла.



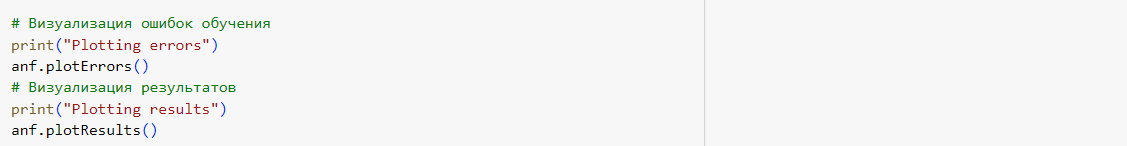


Рисунок 28 – Определение функций и создание объектов для работы сети

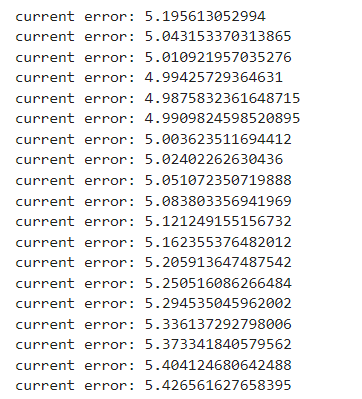


Рисунок 29 – Обучение сети

На данном рисунке видно, как нейронная сеть переобучается и кол-во ошибок растет после определенного кол-ва эпох.

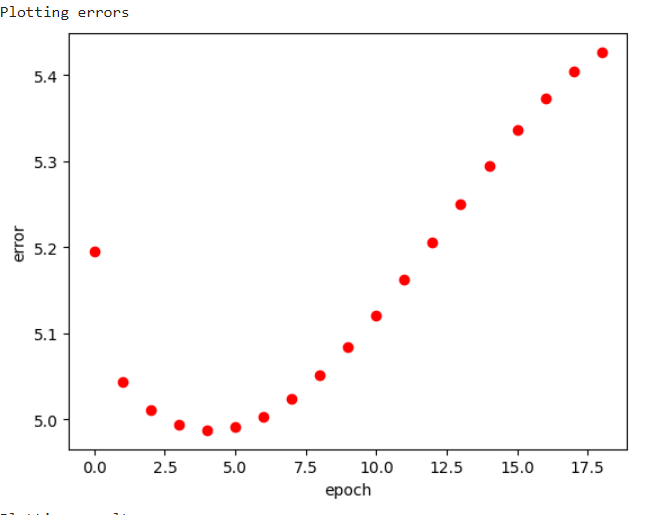


Рисунок 30 – График ошибок

Визуальное представление обучения нейронной сети ANFIS.

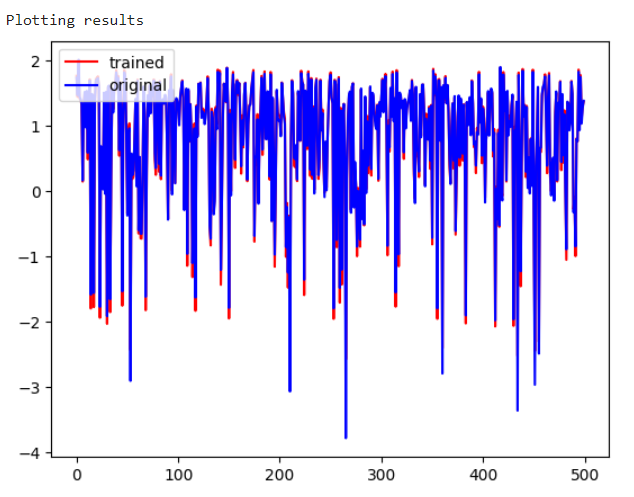


Рисунок 31 – График оригинальных и обученных точек

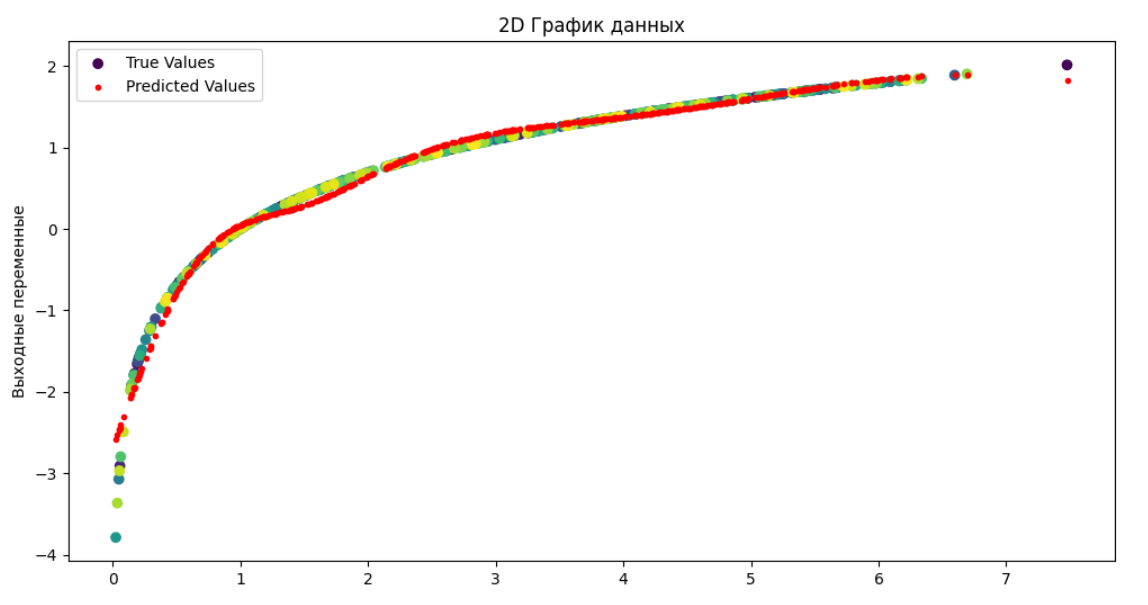


Рисунок 32 – Истинные и предсказанные метки

На данном рисунке видно, как соответствуют истинные данные и данные прошедшие обучение нейронной сетью.

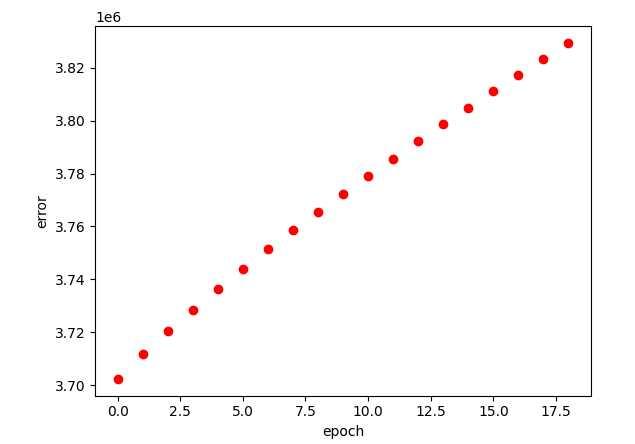


Рисунок 33 – График ошибок после повторного обучения

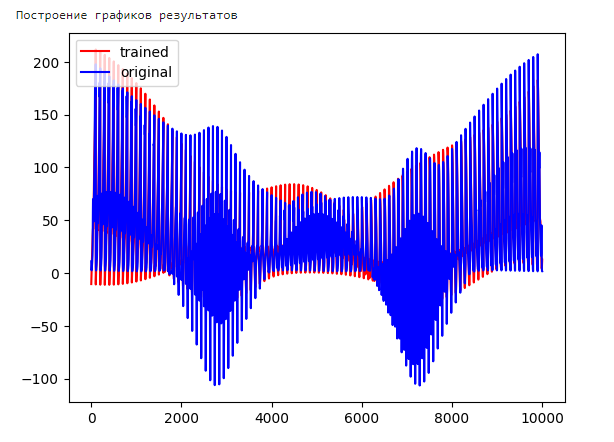


Рисунок 34 – График результатов

На данном графике можно сравнить соответствие данных и как они были обучены.

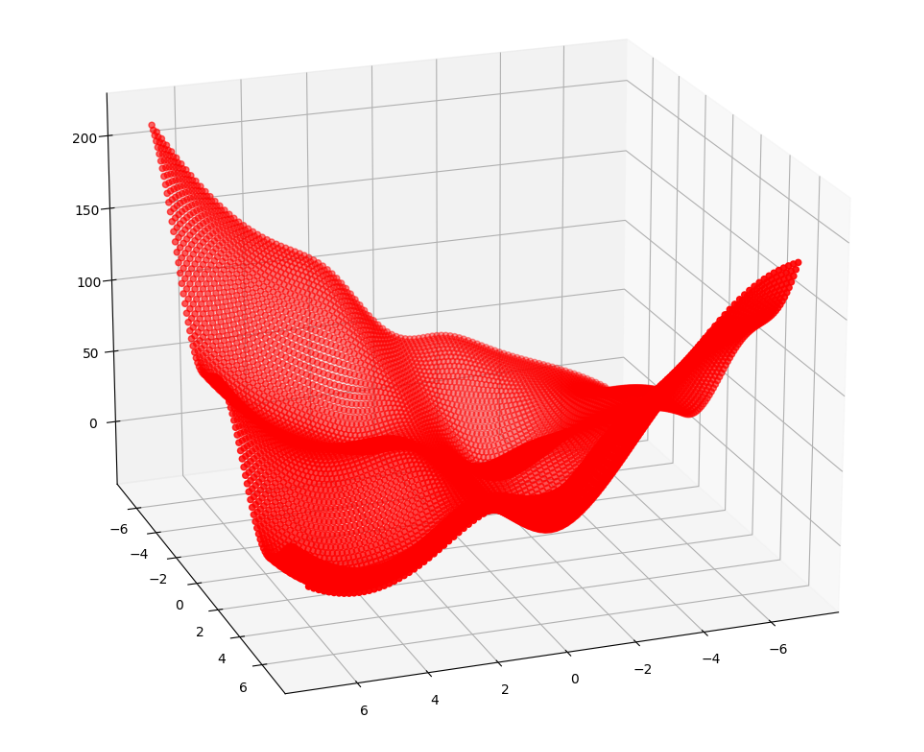


Рисунок 35 –Предсказанные метки после обучения

График, построенный по предсказанным точкам. По нему можно сказать, что обучение прошло не до конца и график только частично соответствует заявленной функции. Необходимо, либо лучше подготовить данные, либо изменить реализацию нейронной сети.

3 RBF

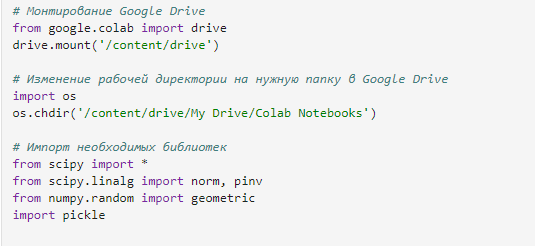


Рисунок 36 – Подключение библиотек



Рисунок 37 – Инициализация RBF сети

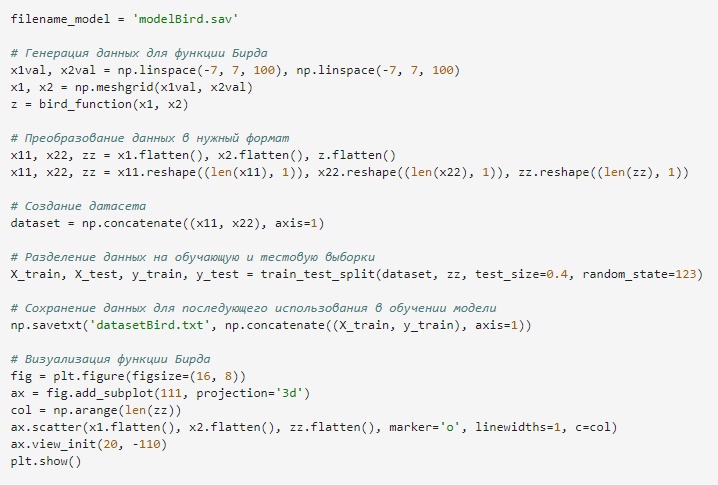


Рисунок 38 – Инициализация и вывод функции Бирда

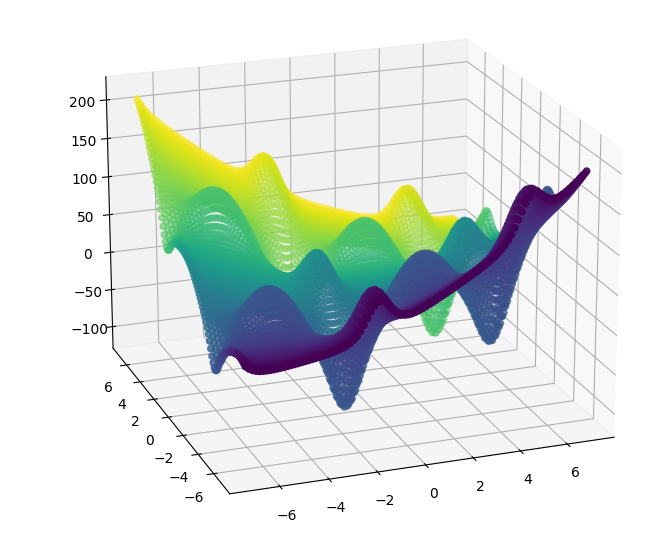


Рисунок 39 – График Бирда

Визуально представление функции Бирда на 2д графике.

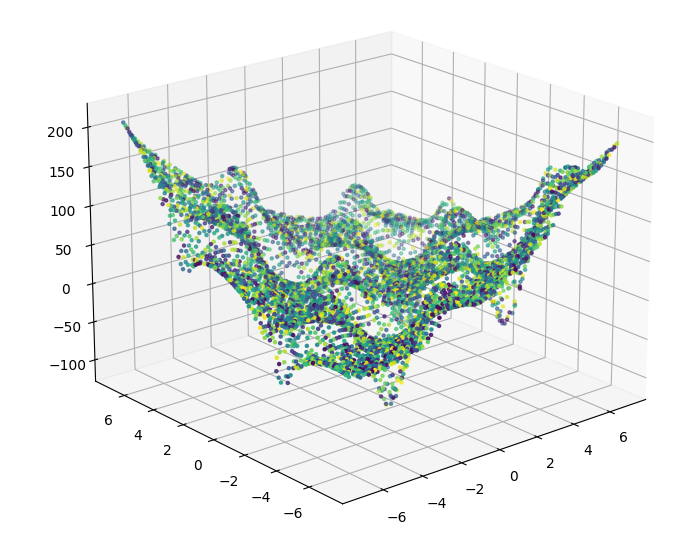


Рисунок 40 – Истинные метки

Внешний вид истинных меток, созданные для сравнения с предсказанными метками.

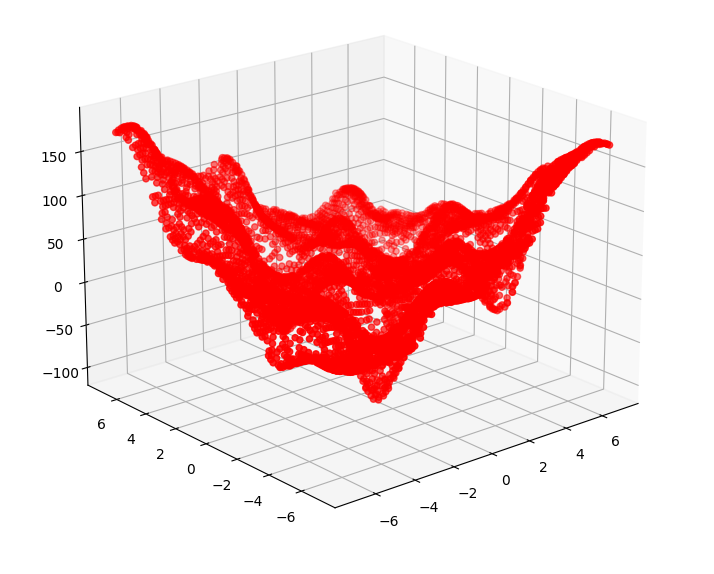


Рисунок 41 –Предсказанные метки

По предсказанными меткам можно сказать, что обучение прошло успешно и равно почти 100%.

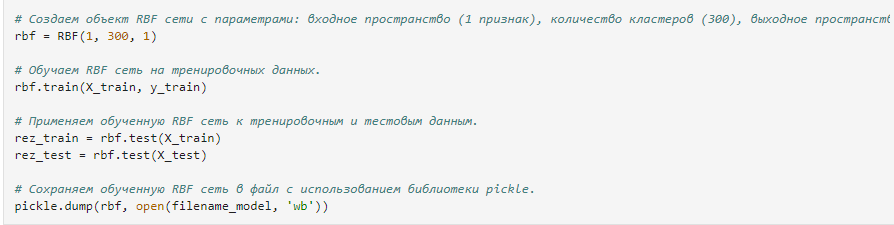


Рисунок 42 – Обучение с большим количеством эпох

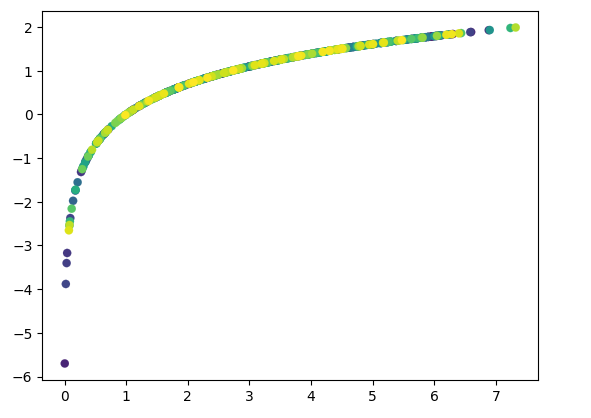


Рисунок 43 – Истинные метки в виде графика

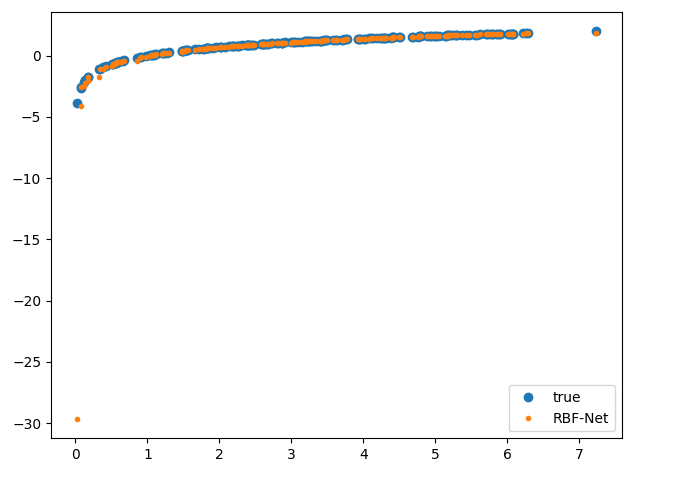


Рисунок 44 – Истинные и предсказанные метки после обучения

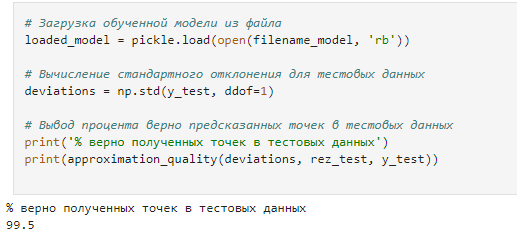


Рисунок 45 – Загрузка данных и подсчёт процентов

3 MLPRegressor

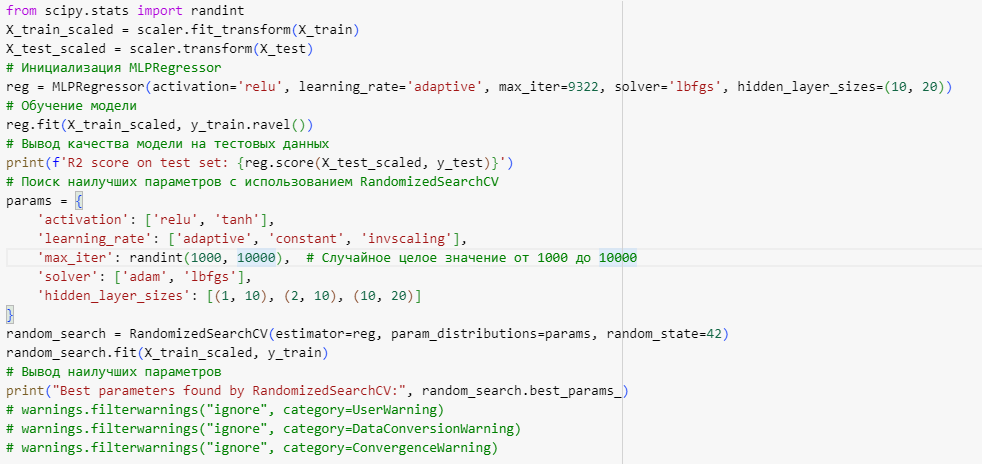


Рисунок 46 – Реализация MLPRegressor

Подборка параметров для лучшего выбора для работы сети.



Рисунок 47 – Лучшие параметры для данной сети

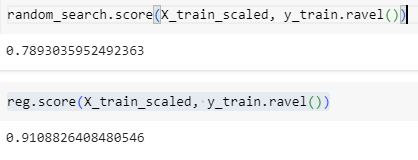


Рисунок 48 – Оценка эффективности на тренировочных данных

При случайном поиске гиперпараметров оценка эффективности составляет 78%, а оценка эффективности регрессионной составляет 91%.

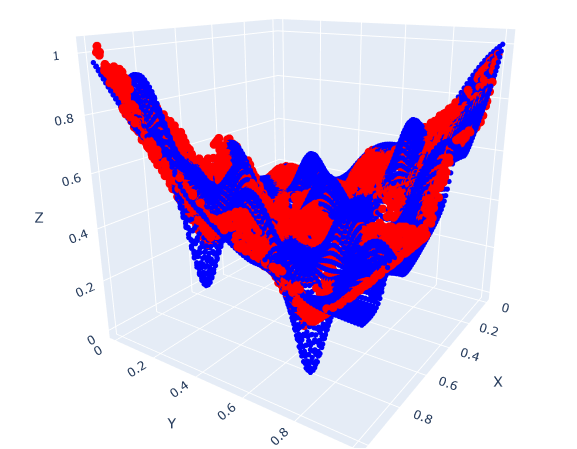


Рисунок 49 – График сравнения точек.

На данном графике можно увидеть, как предсказанные точки и истинные выглядят на графике, по нему можно оценить эффективность работы MLPregressor.

3 Результаты работы

В результате выполнения лабораторной работы была разработана и реализована матрица случайных чисел, созданная в соответствии с указанным законом распределения. Матрица состоит из M строк, представляющих собой наблюдения, и 1 или 2 столбцов, играющих роль аргументов некоторой функциональной зависимости от одного или двух аргументов.

Далее данные были разделены на обучающую и тестирующую выборки. С использованием этих данных была осуществлена аппроксимация функций f1 и f2 тремя способами: сетью прямого распространения сигнала, RBF-сетью и нейро-нечеткой сетью ANFIS.

Для построения нейро-нечеткой сети ANFIS были выполнены следующие шаги:

a. Сгенерирована система нечеткого вывода и настроена данная система по обучающей выборке.

b. Показана структура построенной системы нечеткого вывода, включая входные и выходные переменные, функции принадлежности и правила.

c. Проведен просмотр поверхности системы нечеткого вывода до и после настройки с использованием ANFIS.

d. Вычислены выходные значения системы нечеткого вывода для обучающей и тестирующей выборок.

e. Построены графики ошибок и сравнения значений функции для обучающей и контролирующей выборок, а также выходных значений системы нечеткого вывода до и после настройки.

f. Реализовать MLPRegressor и обучить на нем данные.

Анализ результатов показал эффективность использованных методов аппроксимации функций, что может быть использовано в дальнейших исследованиях и улучшениях в области анализа данных с применением нейронных сетей.

# Приложение

import numpy as np

from numpy.random import geometric

from sklearn.model\_selection import train\_test\_split

from mpl\_toolkits.mplot3d import Axes3D

from random import seed

from random import random

from math import exp

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import plotly.graph\_objects as go

from plotly.subplots import make\_subplots

from sklearn.preprocessing import MinMaxScaler

import plotly.graph\_objects as go

from plotly.subplots import make\_subplots

In [144]:

# Измененная функция инициализации сети

def initialize\_network(n\_inputs, n\_hidden, n\_outputs):

network = list()

hidden\_layer = [{'weights':[random() for i in range(n\_inputs+1)]} for i in range(n\_hidden)]

network.append(hidden\_layer)

output\_layer = [{'weights':[random() for i in range(n\_hidden +1)]} for i in range(n\_outputs)]

network.append(output\_layer)

return network

# Расчет активации нейрона для входного значения

def activate(weights, inputs):

activation = weights[-1]

for i in range(len(weights)-1):

activation += weights[i] \* inputs[i]

return activation

# Функция активации нейрона

def transfer(activation):

return 1 / (1 + np.exp(-activation))

# Проход вперед: входные данные передаются через сеть для получения выхода

def forward\_propagate(network, row):

inputs = row

for layer in network:

new\_inputs = []

for neuron in layer:

activation = activate(neuron['weights'], inputs)

neuron['output'] = transfer(activation)

new\_inputs.append(neuron['output'])

inputs = new\_inputs

return inputs

# Расчет производной вывода нейрона

def transfer\_derivative(output):

return output \* (1 - output)

# Обратное распространение ошибки и сохранение в нейронах

def backward\_propagate\_error(network, expected):

for i in reversed(range(len(network))):

layer = network[i]

errors = list()

if i != len(network)-1:

for j in range(len(layer)):

error = 0.0

for neuron in network[i + 1]:

error += (neuron['weights'][j] \* neuron['delta'])

errors.append(error)

else:

for j in range(len(layer)):

neuron = layer[j]

errors.append(expected[j] - neuron['output'])

for j in range(len(layer)):

neuron = layer[j]

neuron['delta'] = errors[j] \* transfer\_derivative(neuron['output'])

# Обновление весов сети в соответствии с ошибкой

def update\_weights(network, row, l\_rate):

for i in range(len(network)):

inputs = row[:-1]

if i != 0:

inputs = [neuron['output'] for neuron in network[i - 1]]

for neuron in network[i]:

for j in range(len(inputs)):

neuron['weights'][j] += l\_rate \* neuron['delta'] \* inputs[j]

neuron['weights'][-1] += l\_rate \* neuron['delta']

# Обучение сети на тренировочных данных

def train\_network(network, train, l\_rate, n\_epoch, n\_outputs):

for epoch in range(n\_epoch):

sum\_error = 0

for row in train:

outputs = forward\_propagate(network, row)

expected = [0 for i in range(n\_outputs)]

expected[0] = row[-1]

sum\_error += sum([(expected[i] - outputs[i])\*\*2 for i in range(len(expected))])

backward\_propagate\_error(network, expected)

update\_weights(network, row, l\_rate)

print('> epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l\_rate, sum\_error))

x=[]

i = 0

# Функция для вычисления логарифма от x

def function\_log(x):

return np.log(x)

# Генерация случайных данных для заданного количества точек

def generation\_point(count):

xa = []

i = 0

while i < count:

# Генерация случайных чисел с использованием нормального и равномерного распределений

rand = np.random.normal(10, 1, 1) \* np.random.uniform(0, 0.6, 1)

xa.extend(rand)

i = i + 1

# Вычисление значения функции для сгенерированных данных

z1 = function\_log(xa)

x1 = np.reshape(xa, (len(xa), 1))

z1 = np.reshape(z1, (len(z1), 1))

return x1, z1

# Генерация данных и разделение их на обучающий и тестовый наборы

qq, ee = generation\_point(5000)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(qq, ee, test\_size=0.4, random\_state=123)

# Визуализация сгенерированных данных

plt.scatter(qq, ee, marker='o', linewidths=3, c='BLUE')

plt.show()

# Объединение обучающих данных в один датасет

dataset1 = np.concatenate((X\_train, y\_train), axis=1)

dataset1 = dataset1.tolist()

# Определение количества входных и выходных параметров в датасете

n\_inputs = len(dataset1[0]) - 1

n\_outputs = 1

# Установка параметров обучения

l\_rate = 1.0

n\_epoch = 50

# Инициализация нейронной сети

network = initialize\_network(n\_inputs, 20, n\_outputs)

# Обучение нейронной сети на обучающем датасете

train\_network(network, dataset1, l\_rate, n\_epoch, n\_outputs)

# Функция для получения прогноза от нейронной сети для заданной строки данных

def predict(network, row):

outputs = forward\_propagate(network, row)

return outputs

# Инициализация переменной для индексации

i = -1

# Прогнозирование для каждой строки в обучающем наборе

for row in X\_train:

i = i + 1

# Получение прогноза от нейронной сети для текущей строки

prediction = predict(network, row)

# Сохранение прогнозов в массиве

if i == 0:

predictTrain = np.array(prediction)

else:

predictTrain = np.append(predictTrain, prediction, axis=0)

# Визуализация сгенерированных данных и прогнозов

plt.scatter(qq, ee, marker='o', linewidths=1, c='BLUE', label='Исходные данные')

plt.scatter(X\_train, predictTrain, marker='o', linewidths=1, c='RED', label='Прогнозы сети')

# Добавление легенды

plt.legend()

# Отображение графика

plt.show()

def approximation\_quality(deviations, predictions, data):

"""

Функция для вычисления кол-ва верно предсказанных точек в данных.

Параметры:

- deviations: стандартное отклонение данных

- predictions: массив предсказанных значений

- data: исходные данные

Возвращает кол-во верно предсказанных точек.

"""

# Вычисление отношения верных предсказаний к общему числу точек

correct\_predictions = np.sum(np.abs(predictions - data) <= deviations)

total\_points = len(data)

# Процент верно предсказанных точек

accuracy\_percentage = (correct\_predictions / total\_points) \* 100

return accuracy\_percentage

# Вычисление стандартного отклонения для использования в функции approximation\_quality

deviations = np.std(X\_train, ddof=1)

# Вывод процента верно предсказанных точек в тестовых данных

print('верно полученных точек в тестовых данных')

print(approximation\_quality(deviations, predictTrain, X\_train))

# Функция Бирда

def bird\_function(x, y):

return np.sin(x)\*np.exp((1-np.cos(y))\*\*2) + np.cos(y)\*np.exp((1-np.sin(x))\*\*2) + (x-y)\*\*2

# Генерация данных и меш-грида для визуализации

x1val = np.linspace(-7, 7, 100)

x2val = np.linspace(-7, 7, 100)

x1, x2 = np.meshgrid(x1val, x2val)

z = bird\_function(x1, x2)

# Визуализация функции Бирда

fig = plt.figure(figsize=(16, 8))

ax = fig.add\_subplot(111, projection='3d')

col = np.arange(10000)

ax.scatter(x1, x2, z, marker='o', linewidths=1, c=col)

# Поворот графика против часовой стрелки

ax.view\_init(elev=20, azim=-120)

# # # Подготовка данных для обучения и тестирования

# x1 = x1.flatten()

# x2 = x2.flatten()

# z = z.flatten()

scaler = MinMaxScaler()

x1 = scaler.fit\_transform(x1.reshape(-1, 1))

x2 = scaler.fit\_transform(x2.reshape(-1, 1))

z = scaler.fit\_transform(z.reshape(-1, 1))

dataset = np.column\_stack((x1, x2))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset, z, test\_size=0.4, random\_state=123)

plt.show()

# Соединение обучающих данных и целевых значений в один массив

dataset1 = np.concatenate((X\_train, y\_train.reshape(-1, 1)), axis=1)

# datsset1 = np.column\_stack((X\_train, y\_train))

# Определение количества входных и выходных нейронов

n\_inputs = len(dataset1[0]) - 1

n\_outputs = 1

# Настройка параметров обучения

l\_rate = 1.7 # Скорость обучения (может потребоваться настройка)

n\_epoch = 30 # Количество эпох обучения (может потребоваться настройка)

# Инициализация нейронной сети

network = initialize\_network(n\_inputs, 10, n\_outputs)

# Обучение нейронной сети

train\_network(network, dataset1, l\_rate, n\_epoch, n\_outputs)

# Инициализация переменной для индексации

i = -1

# Прогнозирование для каждой строки в dataset1

for row in dataset1:

i = i + 1

prediction = predict(network, row)

# Сохранение прогнозов в массив predictTrain

if i == 0:

predictTrain = np.array(prediction)

else:

predictTrain = np.append(predictTrain, prediction, axis=0)

# Вывод прогнозов

print(predictTrain)

# Прогнозирование для каждой точки

predicted\_values = np.array([forward\_propagate(network, np.array([x1, x2]))[0] for x1, x2 in zip(X\_train[:, 0], X\_train[:, 1])])

# Создание трехмерного графика

fig = go.Figure()

# Добавление точек True Function

fig.add\_trace(go.Scatter3d(x=X\_train[:, 0], y=X\_train[:, 1], z=y\_train,

mode='markers', marker=dict(size=3, color='blue'), name='True Function'))

# Добавление точек Predicted Values

fig.add\_trace(go.Scatter3d(x=X\_train[:, 0], y=X\_train[:, 1], z=predicted\_values,

mode='markers', marker=dict(size=5, color='red'), name='Predicted Values'))

# Настройка макета

fig.update\_layout(scene=dict(xaxis\_title='X1', yaxis\_title='X2', zaxis\_title='Values'),

margin=dict(l=0, r=0, b=0, t=0))

# Отображение графика

fig.show()

plt.show()

# Создание данных для визуализации

xx1val = np.linspace(-7, 7, 100)

xx2val = np.linspace(-7, 7, 100)

x1plt, x2plt = np.meshgrid(xx1val, xx2val)

y1plt = bird\_function(x1plt, x2plt)

# Визуализация функции Бирда

scatter\_true = go.Scatter3d(x=x1plt.flatten(), y=x2plt.flatten(), z=y1plt.flatten(),

mode='markers', marker=dict(size=3, color='blue'), name='True Function')

# Визуализация предсказанных значений на тренировочных данных

predictTrain = np.array([forward\_propagate(network, row) for row in X\_train])

predictTrain = np.reshape(predictTrain, (len(predictTrain), 1))

scatter\_train = go.Scatter3d(x=X\_train.flatten(), y=predictTrain.flatten(), z=y\_train.flatten(),

mode='markers', marker=dict(size=5, color='red'), name='Predicted Values')

# # scatter\_train = go.Scatter3d(x=X\_train[:, 0], y=X\_train[:, 1], z=predictTrain.flatten(),

# # mode='markers', marker=dict(size=5, color='red'), name='Predicted Values')

# Создание подграфика

fig = make\_subplots(rows=1, cols=1, specs=[[{'type': 'scatter3d'}]])

# Добавление визуализаций

fig.add\_trace(scatter\_true)

# fig.add\_trace(scatter\_train)

# Настройка макета

fig.update\_layout(scene=dict(xaxis\_title='X', yaxis\_title='Y', zaxis\_title='Z'),

margin=dict(l=0, r=0, b=0, t=0))

# Отображение графика

fig.show()

# Создание данных для визуализации

xx1val = np.linspace(-7, 7, 100)

xx2val = np.linspace(-7, 7, 100)

x1plt, x2plt = np.meshgrid(xx1val, xx2val)

y1plt = bird\_function(x1plt, x2plt)

# Масштабирование данных от 0 до 1

scaler = MinMaxScaler()

x1plt\_scaled = scaler.fit\_transform(x1plt.flatten().reshape(-1, 1)).reshape(x1plt.shape)

x2plt\_scaled = scaler.fit\_transform(x2plt.flatten().reshape(-1, 1)).reshape(x2plt.shape)

y1plt\_scaled = scaler.fit\_transform(y1plt.flatten().reshape(-1, 1)).reshape(y1plt.shape)

# Визуализация функции Бирда

scatter\_true = go.Scatter3d(x=x1plt\_scaled.flatten(), y=x2plt\_scaled.flatten(), z=y1plt\_scaled.flatten(),

mode='markers', marker=dict(size=3, color='blue'), name='True Function')

# Визуализация предсказанных значений на тренировочных данных

predictTrain = np.array([forward\_propagate(network, row)[0] for row in X\_train])

predictTrain = np.reshape(predictTrain, (len(predictTrain), 1))

scatter\_train = go.Scatter3d(x=X\_train[:, 0], y=X\_train[:, 1], z=predictTrain.flatten(),

mode='markers', marker=dict(size=5, color='red'), name='Predicted Values')

# Создание подграфика

fig = make\_subplots(rows=1, cols=1, specs=[[{'type': 'scatter3d'}]])

# Добавление визуализаций

fig.add\_trace(scatter\_true)

fig.add\_trace(scatter\_train)

# Настройка макета

fig.update\_layout(scene=dict(xaxis\_title='X', yaxis\_title='Y', zaxis\_title='Z'),

margin=dict(l=0, r=0, b=0, t=0))

# Отображение графика

fig.show()

# Импорт необходимых библиотек

import numpy as np

from google.colab import drive

# Монтирование Google Drive

drive.mount('/content/drive')

# Переход в рабочую директорию внутри Google Drive

import os

os.chdir('/content/drive/MyDrive/Colab Notebooks/anfis-master/anfis/')

# Установка пакета anfis из указанного пути

!pip install '/content/drive/MyDrive/Colab Notebooks/anfis-master/'

# Установка пакета anfis

!pip install anfis

# Импорт модуля anfis

import anfis

# Генерация данных для функции f1

def generation\_f2(count,max\_x,min\_x,max\_x2,min\_x2):

x2val=[]

x1val=[]

i = 0

while i < count:

rand=(np.random.normal(10, 1, 1)\*np.random.uniform(0, 0.1, 1) ) + np.random.uniform(3.6, -5.6, 1)

if (rand>=(min\_x) and rand<=(max\_x) ):

x1val.extend(rand)

i = i + 1

i = 0

while i < count:

rand=(np.random.normal(10, 1, 1)\*np.random.uniform(0, 0.1, 1) ) + np.random.uniform(3.6, -5.6, 1)

if (rand>=(min\_x2) and rand<=(max\_x2) ):

x2val.extend(rand)

i = i + 1

return x1val,x2val

# Генерация данных для функции f2

def generation\_f1(count):

x1val=[]

i = 0

while i < count:

rand=np.random.normal(10, 1, 1)\*np.random.uniform(0, 0.6, 1)

x1val.extend(rand)

i = i + 1

return x1val

# Оценка качества аппроксимации

def approximation\_quality(deviations,predictData,y\_data):

count=0

for i in range(len(predictData)):

if(predictData[i]>=(y\_data[i]-deviations) and predictData[i]<=(y\_data[i]+deviations)):

count=count+1

return percentage(count,len(predictData))

# Функция f1 или логарифмическая функция

def f1(x):

y=[]

for i in range (0,(len(x))):

y.append(np.log(x[i]))

return y

# Функция f2 или функция Бирда

def f2(x, y):

return np.sin(x)\*np.exp((1-np.cos(y))\*\*2) + np.cos(y)\*np.exp((1-np.sin(x))\*\*2) + (x-y)\*\*2

# Расчет процента от общего числа

def percentage(part, whole):

return 100 \* float(part)/float(whole)

# Подготовка данных для сохранения в файл

x1val = generation\_f1(500)

z = f1(x1val)

x11=[]

x22=np.full(2500,0)

f2zz=[]

for l in x1val:

x11.append(l)

for l in z:

f2zz.append(l)

x11=np.reshape(x11,(len(x11),1))

x22=np.reshape(x22,(len(x22),1))

f2zz=np.reshape(f2zz,(len(f2zz),1))

# Создание и запись данных в файл 'datasetMy.txt'

if(os.path.exists('datasetMy.txt')):

os.remove('datasetMy.txt')

file = open('datasetMy.txt', 'w')

for i in range(len(x11)):

file.write('"'+str(i)+'" '+str(x11[i][0])+' '+str(x22[i][0])+' '+str(f2zz[i][0])+'\n')

file.close()

# Загрузка данных из файла 'datasetMy.txt'

ts = np.loadtxt("datasetMy.txt", usecols=[1, 2, 3])

# Разделение данных на входные (X) и выходные (Y) переменные

X = ts[:, 0:2] # Входные переменные

Y = ts[:, 2] # Выходные переменные

# Создание 2D графика

plt.figure(figsize=(12, 6))

# Визуализация данных на графике

plt.scatter(X[:, 0], Y, marker='o', linewidths=1, label='Значения')

plt.ylabel('Выходные переменные')

plt.title('2D График данных из файла')

plt.legend()

plt.show()

# Импорт необходимых модулей из библиотеки ANFIS

from anfis import membershipfunction

# Определение функций принадлежности для входных переменных

# В данном случае используются функции гауссовского типа

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.}]]

]

# Создание объекта для хранения функций принадлежности

mfc = membershipfunction.MemFuncs(mf)

# Создание модели ANFIS с использованием определенных функций принадлежности

anf = anfis.anfis\_model.ANFIS(X, Y, mfc)

# Обучение модели с использованием гибридного алгоритма Jang Off-Line

anf.trainHybridJangOffLine(epochs=20)

# Вывод результатов

print("Consequent [-1] value:", round(anf.consequents[-1][0], 6))

print("Consequent [-2] value:", round(anf.consequents[-2][0], 6))

print("Fitted value at index 9:", round(anf.fittedValues[9][0], 6))

# Проверка результатов

if (

round(anf.consequents[-1][0], 6) == -5.275538 and

round(anf.consequents[-2][0], 6) == -1.990703 and

round(anf.fittedValues[9][0], 6) == 0.002249

):

print('Test is good')

# Визуализация ошибок обучения

print("Plotting errors")

anf.plotErrors()

# Визуализация результатов

print("Plotting results")

anf.plotResults()

# # Прогнозирование с использованием обученной модели ANFIS

predictTrain = anfis.anfis\_model.predict(anf, X)

# Создание 2D графика

plt.figure(figsize=(12, 6))

# Генерация цветовых значений для точек на графике

col = np.arange(len(Y))

# Визуализация исходных данных

plt.scatter(X[:, 0], Y, marker='o', linewidths=1, c=col, label='True Values')

# Визуализация предсказанных данных

plt.scatter(X[:, 0], predictTrain, marker='.', linewidths=1, c='red', label='Predicted Values')

# Настройка графика

plt.ylabel('Выходные переменные')

plt.title('2D График данных')

plt.legend()

# Отображение графика

plt.show()

# Вычисление стандартного отклонения и вывод процента верно предсказанных точек

deviations = np.std(Y, ddof=1)

print('% верно полученных точек в тестовых данных')

print(approximation\_quality(deviations, predictTrain, Y))

# Функция Бирда

def bird\_function(x, y):

return np.sin(x)\*np.exp((1-np.cos(y))\*\*2) + np.cos(y)\*np.exp((1-np.sin(x))\*\*2) + (x-y)\*\*2

# Генерация данных для функции Бирда

x1val, x2val = np.linspace(-7, 7, 100), np.linspace(-7, 7, 100)

x1f2, x2f2 = np.meshgrid(x1val, x2val)

f2z = bird\_function(x1f2, x2f2)

# Преобразование данных в нужный формат

x11f2, x22f2, f2zz = x1f2.flatten(), x2f2.flatten(), f2z.flatten()

x11f2, x22f2, f2zz = x11f2.reshape((len(x11f2), 1)), x22f2.reshape((len(x22f2), 1)), f2zz.reshape((len(f2zz), 1))

# Запись данных в файл

if os.path.exists('datasetMy.txt'):

os.remove('datasetMy.txt')

file = open('datasetMy.txt', 'w')

for i in range(len(x11f2)):

file.write(f'"{i}" {x11f2[i][0]} {x22f2[i][0]} {f2zz[i][0]}\n')

file.close()

# Загрузка данных из файла

ts = np.loadtxt("datasetMy.txt", usecols=[1, 2, 3])

XAF2 = ts[:, 0:2]

YAF2 = ts[:, 2]

# Визуализация функции Бирда

fig = plt.figure(figsize=(16, 8))

ax = fig.add\_subplot(111, projection='3d')

col = np.arange(len(YAF2))

ax.scatter(XAF2[:, 0], XAF2[:, 1], YAF2, marker='.', linewidths=1, c=col)

ax.view\_init(20,-110)

plt.show()

# Конфигурация функций принадлежности для модели ANFIS

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.}]]

]

# Создание функций принадлежности

mfc = membershipfunction.MemFuncs(mf)

# Создание и обучение модели ANFIS

anf = anfis.anfis\_model.ANFIS(XAF2, YAF2, mfc)

anf.trainHybridJangOffLine(epochs=20)

# Вывод ключевых свойств обученной модели

print("Значение последствия [-1]:", round(anf.consequents[-1][0], 6))

print("Значение последствия [-2]:", round(anf.consequents[-2][0], 6))

print("Предсказанное значение [9]:", round(anf.fittedValues[9][0], 6))

# Проверка, соответствуют ли определенные условия

if (

round(anf.consequents[-1][0], 6) == -5.275538 and

round(anf.consequents[-2][0], 6) == -1.990703 and

round(anf.fittedValues[9][0], 6) == 0.002249

):

print('Тест успешен (условия выполняются)')

# Построение графиков ошибок и результатов

print("Построение графиков ошибок")

anf.plotErrors()

print("Построение графиков результатов")

anf.plotResults()

# Предсказание значений с использованием обученной модели ANFIS

predictTrain = anfis.anfis\_model.predict(anf, XAF2)

# Построение трехмерного графика предсказанных значений

fig = plt.figure(figsize=(20, 12))

ax = fig.add\_subplot(111, projection='3d')

col = np.arange(len(YAF2))

# Визуализация предсказанных значений на тренировочных данных

ax.scatter(XAF2[:, 0], XAF2[:, 1], predictTrain, marker='o', linewidths=1, c='RED')

# Настройка угла обзора для лучшего отображения графика

ax.view\_init(20, 70)

# Вычисление стандартного отклонения тестовых данных

deviations = np.std(YAF2, ddof=1)

# Вывод процента верно предсказанных точек в тестовых данных

print('% верно полученных точек в тестовых данных')

print(approximation\_quality(deviations, predictTrain, YAF2))

# Монтирование Google Drive

from google.colab import drive

drive.mount('/content/drive')

# Изменение рабочей директории на нужную папку в Google Drive

import os

os.chdir('/content/drive/My Drive/Colab Notebooks')

# Импорт необходимых библиотек

from scipy import \*

from scipy.linalg import norm, pinv

from numpy.random import geometric

import pickle

class RBF:

def \_\_init\_\_(self, indim, numCenters, outdim):

self.indim = indim

self.outdim = outdim

self.numCenters = numCenters

# Здесь мы инициализируем центры, соответствующие скрытым нейронам RBF

self.centers = np.random.uniform(-1, 1, (numCenters, indim))

# Здесь два важных параметра, определяющих сеть RBF.

# Первый параметр представляет собой β, второй - вес соединения

self.betas = np.ones(numCenters) / 10

self.W = np.random.random((self.numCenters, self.outdim))

def \_basisfunc(self, c, d):

assert len(d) == self.indim

return np.log(1 + np.linalg.norm(d - self.centers[c]) \*\* 2)

def \_calcAct(self, X):

# рассчитываем активации RBF

G = np.zeros((X.shape[0], self.numCenters), float)

for ci, c in enumerate(self.centers):

for xi, x in enumerate(X):

G[xi, ci] = self.\_basisfunc(ci, x)

return G

def train(self, X, Y):

"""

X: матрица размерности n x indim

y: столбцовый вектор размерности n x 1

"""

# используем k-средние для вычисления центров

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=self.numCenters)

kmeans.fit(X)

self.centers = kmeans.cluster\_centers\_

# рассчитываем активации RBF

G = self.\_calcAct(X)

# рассчитываем веса выхода (псевдообратная матрица)

self.W = np.dot(np.linalg.pinv(G), Y)

def test(self, X):

"""

X: матрица размерности n x indim

"""

G = self.\_calcAct(X)

Y = np.dot(G, self.W)

return Y

# Функция percentage вычисляет процент относительно общего числа.

def percentage(part, whole):

return 100 \* float(part) / float(whole)

# Функция predict прогнозирует выходное значение сети для заданной строки данных.

def predict(loaded\_model, row):

outputs = forward\_propagate(loaded\_model, row)

return outputs

# Функция approximation\_quality оценивает качество аппроксимации данных.

def approximation\_quality(deviations, predictData, y\_data):

count = 0

for i in range(len(predictData)):

if (predictData[i] >= (y\_data[i] - deviations) and predictData[i] <= (y\_data[i] + deviations)):

count = count + 1

return percentage(count, len(predictData))

# Функция f2 представляет собой функцию для генерации данных.

def f2(x, y):

return np.sin(x) \* np.exp((1 - np.cos(y))\*\*2) + np.cos(y) \* np.exp((1 - np.sin(x))\*\*2) + (x - y)\*\*2

filename\_model = 'modelBird.sav'

# Генерация данных для функции Бирда

x1val, x2val = np.linspace(-7, 7, 100), np.linspace(-7, 7, 100)

x1, x2 = np.meshgrid(x1val, x2val)

z = bird\_function(x1, x2)

# Преобразование данных в нужный формат

x11, x22, zz = x1.flatten(), x2.flatten(), z.flatten()

x11, x22, zz = x11.reshape((len(x11), 1)), x22.reshape((len(x22), 1)), zz.reshape((len(zz), 1))

# Создание датасета

dataset = np.concatenate((x11, x22), axis=1)

# Разделение данных на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset, zz, test\_size=0.4, random\_state=123)

# Сохранение данных для последующего использования в обучении модели

np.savetxt('datasetBird.txt', np.concatenate((X\_train, y\_train), axis=1))

# Визуализация функции Бирда

fig = plt.figure(figsize=(16, 8))

ax = fig.add\_subplot(111, projection='3d')

col = np.arange(len(zz))

ax.scatter(x1.flatten(), x2.flatten(), zz.flatten(), marker='o', linewidths=1, c=col)

ax.view\_init(20, -110)

plt.show()

# Рисуем 3d график

fig = plt.figure(figsize=(16,8))

ax = fig.add\_subplot(111, projection='3d')

col = np.arange(len(y\_train))

ax.scatter(X\_train[:, 0], X\_train[:, 1], y\_train, marker = '.', linewidths=1, c=col)

ax.view\_init(20, -130)

# Создаем объект RBF сети с параметрами: входное пространство (2 признака), количество кластеров (150), выходное пространство (1).

rbf = RBF(2, 150, 1)

# Обучаем RBF сеть на тренировочных данных.

rbf.train(X\_train, y\_train)

# Применяем обученную RBF сеть к тренировочным и тестовым данным.

rez\_train = rbf.test(X\_train)

rez\_test = rbf.test(X\_test)

# Сохраняем обученную RBF сеть в файл с использованием библиотеки pickle.

pickle.dump(rbf, open(filename\_model, 'wb'))

# Рисуем 3d график

fig = plt.figure(figsize=(16,8))

ax = fig.add\_subplot(111, projection='3d')

col = np.arange(len(y\_train))

ax.scatter(X\_train[:, 0], X\_train[:, 1], rez\_train, marker = 'o', linewidths=1, c='red')

ax.view\_init(20, -130)

# Вывод процента верно предсказанных точек в тестовых данных

loaded\_model = pickle.load(open(filename\_model, 'rb'))

deviations=np.std(y\_test, ddof=1)

print('% верно полученных точек в тестовых данных')

print(approximation\_quality(deviations,rez\_test,y\_test))

# Генерация случайных чисел

def generation\_sa(count):

xa=[]

i = 0

while i < count:

rand=np.random.normal(10, 1, 1)\*np.random.uniform(0, 0.6, 1)

xa.extend(rand)

i = i + 1

z1 = f1(xa)

x1=np.reshape(xa,(len(xa),1))

z1=np.reshape(z1,(len(z1),1))

return x1, z1

# Задаем имя файла для сохранения модели

filename\_model = 'modelF1-СетьRBF-line.sav'

# Генерируем данные X, Y с использованием функции generation\_sa

X, Y = generation\_sa(500)

# Разделяем данные на тренировочную и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=123) # Создаем объект RBF сети с параметрами: входное пространство (1 признак), количество кластеров (300), выходное пространство (1).

rbf = RBF(1, 300, 1)

# Обучаем RBF сеть на тренировочных данных.

rbf.train(X\_train, y\_train)

# Применяем обученную RBF сеть к тренировочным и тестовым данным.

rez\_train = rbf.test(X\_train)

rez\_test = rbf.test(X\_test)

# Сохраняем обученную RBF сеть в файл с использованием библиотеки pickle.

pickle.dump(rbf, open(filename\_model, 'wb'))

# Визуализация данных

plt.scatter(X, Y , marker = 'o', linewidths=0.1, c=np.arange(len(Y)))

# Визуализация данных

rez\_test = rbf.test(X\_test)

plt.plot(np.sort(X\_test, axis=0), np.sort(y\_test, axis=0), 'o', label='true')

plt.plot(np.sort(X\_test, axis=0) , np.sort(rez\_test , axis=0), '.', label='RBF-Net')

plt.legend()

plt.tight\_layout()

plt.show()

# Загрузка обученной модели из файла

loaded\_model = pickle.load(open(filename\_model, 'rb'))

# Вычисление стандартного отклонения для тестовых данных

deviations = np.std(y\_test, ddof=1)

# Вывод процента верно предсказанных точек в тестовых данных

print('% верно полученных точек в тестовых данных')

print(approximation\_quality(deviations, rez\_test, y\_test))