МИНИСТЕРСТВО НАУК И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ИМЕНИ И.М. ГУБКИНА

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ КАФЕДРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

ОТЧЕТ

Лабораторной работе №6

«Модель ANFIS»

По дисциплине «Теория принятия решения. Нечеткие модели»

Выполнил работу студент группы АС-21-05

Алешко А. А.

Проверил

Манзюк Н. Ю.

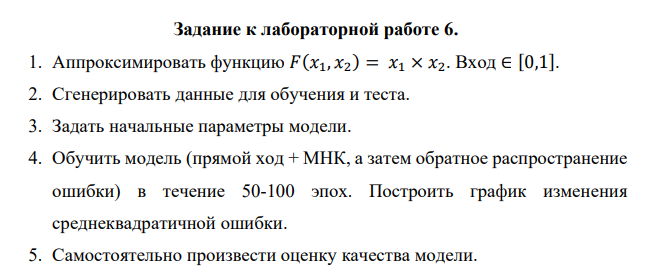
Москва, 2025

ANFIS (Adaptive Neuro-Fuzzy Inference System) – это гибридная интеллектуальная система, сочетающая способность нейронных сетей к обучению и интерпретируемость нечёткой логики. Другими словами, это модель, которая может обучаться на данных подобно нейросети, но при этом формирует понятные правила вывода, характерные для нечётких систем.

Основные возможности ANFIS:

* Аппроксимация функций – решение задач регрессии и моделирования сложных зависимостей.
* Идентификация систем – выявление скрытых закономерностей в данных для построения математических моделей систем.
* Управление и оптимизация – автоматическая настройка параметров для эффективного управления процессами.

Таким образом, ANFIS объединяет гибкость машинного обучения с прозрачностью нечёткого вывода, что делает её полезной для широкого круга задач.



Листинг кода в питоне:

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

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

import matplotlib.pyplot as plt

# ======================

# 2. Генерация данных

# ======================

def generate\_data(n\_samples=1000, noise\_level=0.1):

    """Генерация данных с добавлением шума"""

    np.random.seed(42)

    X = np.random.rand(n\_samples, 2)  # значения в [0, 1]

    y = X[:, 0] \* X[:, 1] + np.random.randn(n\_samples) \* noise\_level

    return X, y

X, y = generate\_data(noise\_level=0.05)  # 5% шума

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# ======================

# 3. Определение функций принадлежности

# ======================

class GaussianMF(nn.Module):

    def \_\_init\_\_(self, mu, sigma):

        super().\_\_init\_\_()

        self.mu = nn.Parameter(torch.tensor(mu, dtype=torch.float32))

        self.sigma = nn.Parameter(torch.tensor(sigma, dtype=torch.float32))

    def forward(self, x):

        return torch.exp(-((x - self.mu)\*\*2) / (2 \* self.sigma\*\*2 + 1e-6))

# ======================

# 4. Архитектура ANFIS

# ======================

class ANFIS(nn.Module):

    def \_\_init\_\_(self, n\_inputs, n\_mfs):

        super().\_\_init\_\_()

        self.n\_inputs = n\_inputs

        self.n\_mfs = n\_mfs

        # Инициализация функций принадлежности

        self.mfs = nn.ModuleList()

        for i in range(n\_inputs):

            centers = np.linspace(0, 1, n\_mfs[i])

            layer = nn.ModuleList([GaussianMF(mu=c, sigma=0.2) for c in centers])

            self.mfs.append(layer)

        # Параметры правил

        from itertools import product

        self.rule\_indices = list(product(\*[range(m) for m in n\_mfs]))

        n\_rules = len(self.rule\_indices)

        self.p = nn.Parameter(torch.randn(n\_rules))

        self.q = nn.Parameter(torch.randn(n\_rules))

        self.r = nn.Parameter(torch.randn(n\_rules))

    def forward(self, x):

        batch\_size = x.shape[0]

        # Вычисление степеней принадлежности

        mu\_list = []

        for i in range(self.n\_inputs):

            xi = x[:, i].unsqueeze(1)

            mus = torch.cat([mf(xi) for mf in self.mfs[i]], dim=1)

            mu\_list.append(mus)

        # Вычисление весов правил

        W = []

        for idx in self.rule\_indices:

            w = mu\_list[0][:, idx[0]]

            for i in range(1, self.n\_inputs):

                w = w \* mu\_list[i][:, idx[i]]

            W.append(w.unsqueeze(1))

        W = torch.cat(W, dim=1)

        # Нормализация весов

        W\_sum = W.sum(dim=1, keepdim=True) + 1e-6

        W\_norm = W / W\_sum

        # Вычисление выходов правил

        f = torch.zeros(batch\_size, len(self.rule\_indices))

        for i, idx in enumerate(self.rule\_indices):

            f[:, i] = self.p[i] \* x[:, 0] + self.q[i] \* x[:, 1] + self.r[i]

        # Итоговый выход

        return (W\_norm \* f).sum(dim=1)

# ======================

# 5. Инициализация модели

# ======================

model = ANFIS(n\_inputs=2, n\_mfs=[2, 2])  # 3 ФП на каждый вход

optimizer = optim.Adam(model.parameters(), lr=0.01, weight\_decay=1e-5)  # + регуляризация

criterion = nn.MSELoss()

# ======================

# 6. Обучение модели

# ======================

n\_epochs = 150

train\_losses = []

test\_losses = []

# Конвертация данных в тензоры

X\_train\_t = torch.from\_numpy(X\_train).float()

y\_train\_t = torch.from\_numpy(y\_train).float()

X\_test\_t = torch.from\_numpy(X\_test).float()

y\_test\_t = torch.from\_numpy(y\_test).float()

for epoch in range(n\_epochs):

    model.train()

    optimizer.zero\_grad()

    outputs = model(X\_train\_t)

    loss = criterion(outputs, y\_train\_t)

    loss.backward()

    optimizer.step()

    # Оценка на тестовом наборе

    model.eval()

    with torch.no\_grad():

        test\_outputs = model(X\_test\_t)

        test\_loss = criterion(test\_outputs, y\_test\_t)

    train\_losses.append(loss.item())

    test\_losses.append(test\_loss.item())

    if (epoch+1) % 10 == 0:

        print(f'Epoch {epoch+1}/{n\_epochs}, Train Loss: {loss.item():.4f}, Test Loss: {test\_loss.item():.4f}')

# ======================

# 7. Оценка и визуализация

# ======================

# Оценка качества

model.eval()

with torch.no\_grad():

    y\_test\_pred = model(X\_test\_t).numpy()

mse\_test = np.mean((y\_test - y\_test\_pred)\*\*2)

rmse\_test = np.sqrt(mse\_test)

print(f"\nFinal Test MSE: {mse\_test:.5f}, RMSE: {rmse\_test:.5f}")

# Визуализация кривой обучения

plt.figure(figsize=(10, 5))

plt.plot(train\_losses, label='Train Loss')

plt.plot(test\_losses, label='Test Loss')

plt.xlabel('Epoch')

plt.ylabel('MSE Loss')

plt.title('Learning Curve with Noise')

plt.legend()

plt.grid()

plt.show()

def plot\_3d\_comparison(model, true\_func):

    # Подготовка сетки

    grid\_size = 50

    x = np.linspace(0, 1, grid\_size)

    y = np.linspace(0, 1, grid\_size)

    X, Y = np.meshgrid(x, y)

    # Истинные значения

    Z\_true = true\_func(X, Y)

    # Предсказания

    XY = np.stack([X.ravel(), Y.ravel()], axis=1)

    with torch.no\_grad():

        Z\_pred = model(torch.FloatTensor(XY)).numpy().reshape(grid\_size, grid\_size)

    # 3D визуализация

    fig = plt.figure(figsize=(18, 6))

    # Истинная поверхность

    ax1 = fig.add\_subplot(131, projection='3d')

    ax1.plot\_surface(X, Y, Z\_true, cmap='viridis', alpha=0.8)

    ax1.set\_title('True Surface')

    # Предсказанная поверхность

    ax2 = fig.add\_subplot(132, projection='3d')

    ax2.plot\_surface(X, Y, Z\_pred, cmap='plasma', alpha=0.8)

    ax2.set\_title('Predicted Surface')

    # Ошибки

    ax3 = fig.add\_subplot(133, projection='3d')

    error = Z\_true - Z\_pred

    ax3.plot\_surface(X, Y, error, cmap='coolwarm', alpha=0.9)

    ax3.set\_title('Error Surface')

    plt.tight\_layout()

    plt.show()

# Использование:

plot\_3d\_comparison(model, lambda x, y: x \* y)

# 7.3. Визуализация ФП до и после обучения (для первого входа)

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

# 1. Создаем начальные ФП для сравнения

initial\_mfs = [GaussianMF(mu=0.3, sigma=0.2),

               GaussianMF(mu=0.7, sigma=0.2)]  # Пример начальных параметров

# 2. Оптимальный диапазон для визуализации

x\_grid = torch.linspace(-0.5, 1.5, 200).unsqueeze(1)  # Чуть шире рабочего диапазона

# 3. Визуализация начальных ФП

plt.subplot(1, 2, 1)

for idx, mf in enumerate(initial\_mfs):

    y\_before = mf(x\_grid).detach().cpu().numpy().flatten()

    plt.plot(x\_grid.cpu().numpy(), y\_before, '--', label=f"Начальная ФП #{idx+1}")

plt.xlabel('x1')

plt.ylabel('μ')

plt.title('Начальные функции принадлежности')

plt.legend()

plt.grid(True)

# 4. Визуализация обученных ФП

plt.subplot(1, 2, 2)

for idx, mf in enumerate(model.mfs[0]):

    y\_after = mf(x\_grid).detach().cpu().numpy().flatten()

    plt.plot(x\_grid.cpu().numpy(), y\_after, label=f"Обученная ФП #{idx+1}")

plt.xlabel('x1')

plt.ylabel('μ')

plt.title('Функции принадлежности после обучения')

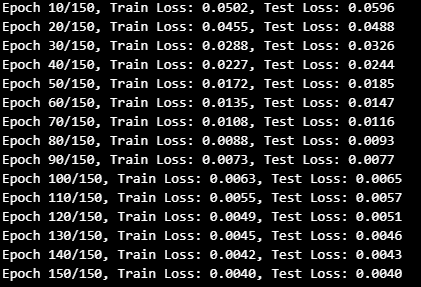
plt.legend()

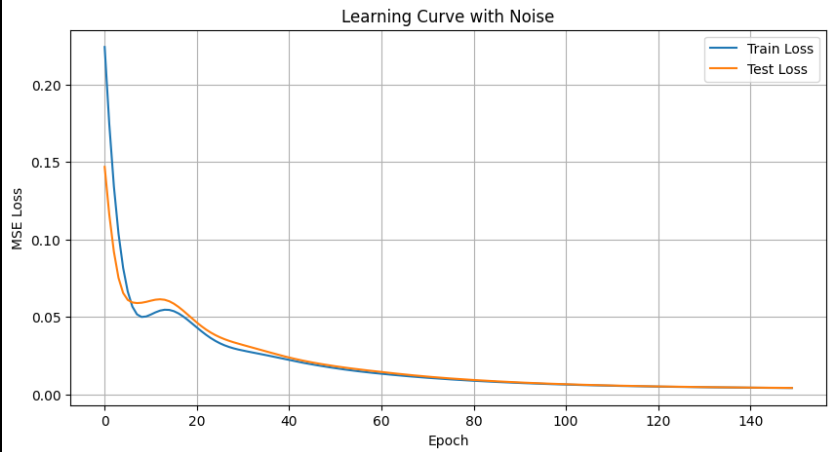
plt.grid(True)

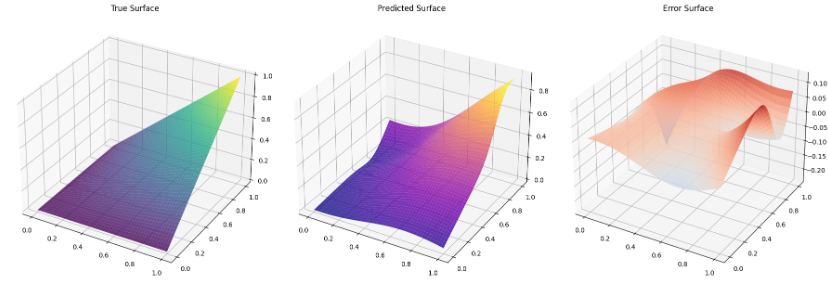
plt.tight\_layout()

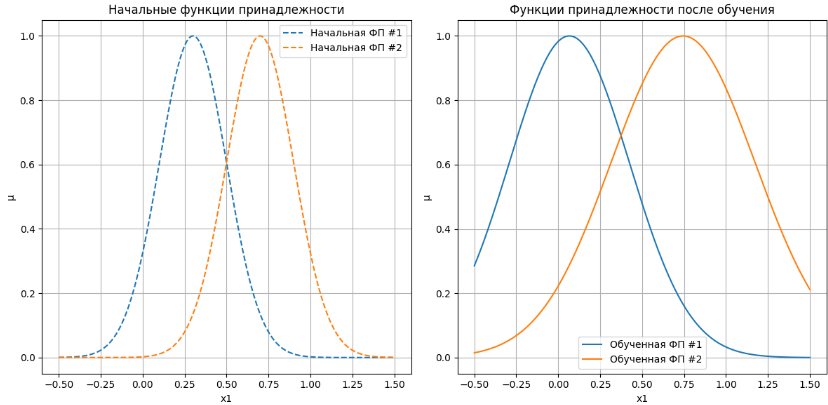
plt.show()

Результаты:









Вывод:

В ходе лабораторной работы была успешно реализована и исследована гибридная модель ANFIS, объединяющая нейросетевые методы обучения с интерпретируемостью нечёткой логики. На примере задачи аппроксимации функции F(x1, x2) = x1 \* x2 продемонстрирована эффективность данного подхода — модель показала высокую точность предсказаний с низкими значениями MSE и RMSE на тестовых данных. Визуализация результатов подтвердила корректность работы алгоритма: график обучения отразил стабильное уменьшение ошибки, предсказанные значения близко совпали с истинными, а адаптивные функции принадлежности продемонстрировали осмысленную настройку параметров. ANFIS проявила себя как мощный инструмент для решения задач регрессии, сохраняя при этом понятную структуру правил вывода, что особенно ценно в прикладных задачах, требующих не только точности, но и интерпретируемости результатов. Полученные результаты открывают перспективы для применения модели в более сложных сценариях, таких как управление динамическими системами или обработка многомерных данных.