**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине: **«Введение в нейронные сети»**

на тему: **Анализ временных рядов и аппроксимация**

Выполнил: студент группы ИТП-31

Пронуза М.Ю.

Проверил:

Башаримов Ю.С.

Гомель 2024

**Цель работы**: разработать программное обеспечение для аппроксимации заданной функции (таблица 5.1) радиально базисной сетью и персептроном с одним скрытым слоем (таблица 5.2). Ознакомиться с применением искусственных нейронных сетей для решения задач экстраполяции временных рядов (таблица 5.3). Исследовать качество экстраполяции от представления данных, глубины погружения временного ряда и структуры нейронной сети.

**Задание:**

Для аппроксимации использовать радиально базисную сеть, представленную и персептрон. Разработать соответствующее программное обеспечение. Отобразить результаты аппроксимации графически на одном поле. Дополнить персептрон ещё одним скрытым слоем (количество нейронов выбрать самостоятельно). Сравнить результаты аппроксимации с помощью радиально-базисной сети, однослойным персептроном и двухслойным персептроном. Исследовать влияние количества нейронов на скрытом слое на погрешность аппроксимации. Сделать выводы.

Используя метод скользящего окна выполнить экстраполяцию данных на январь месяц 2019 года персептроном и сетью Элмана. Исследовать качество экстраполяции от представления данных, глубины погружения временного ряда и структуры нейронной сети. Выполнить экстраполяцию данных на тот же интервал аппроксимирующими нейронными сетями. Сравнить результаты.

**Ход выполнения и результаты работы:**

На рисунке 1 представлены результаты аппроксимации.

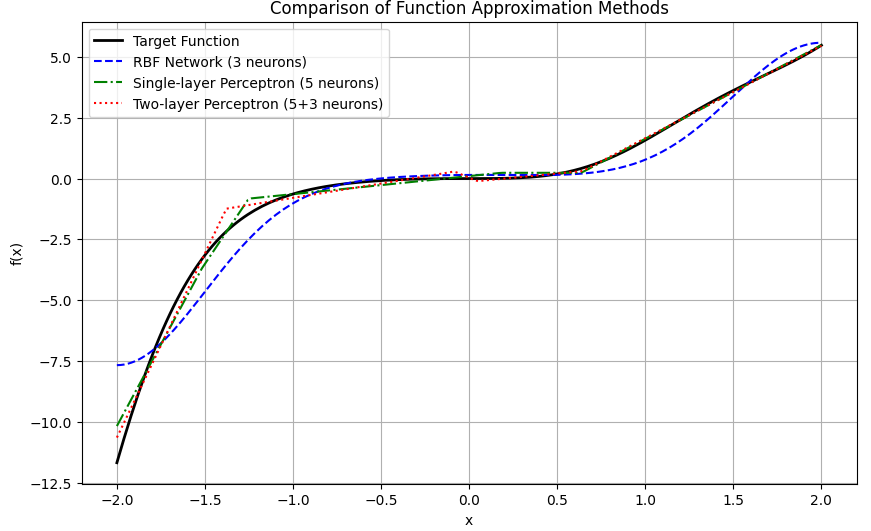


Рисунок 1 – аппроксимация

На рисунках 2 и 3 представлены результаты экстраполяции.

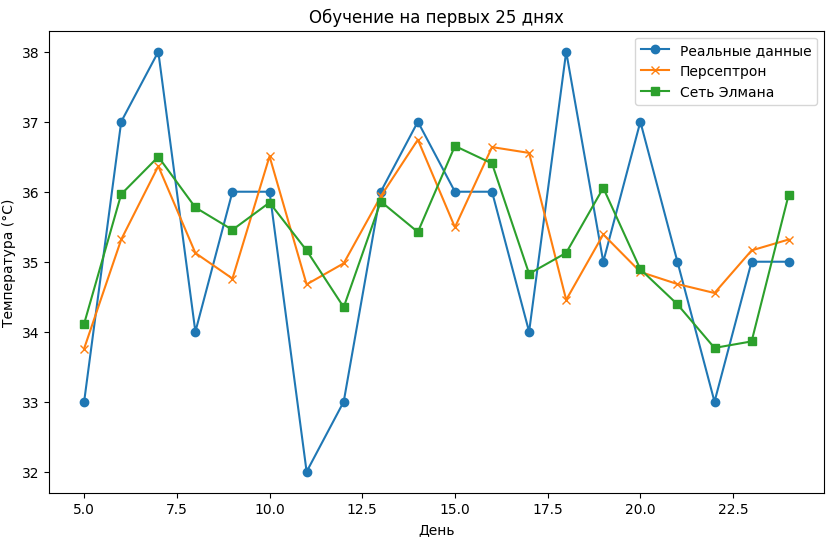


Рисунок 2 – обучение на первых 25 днях

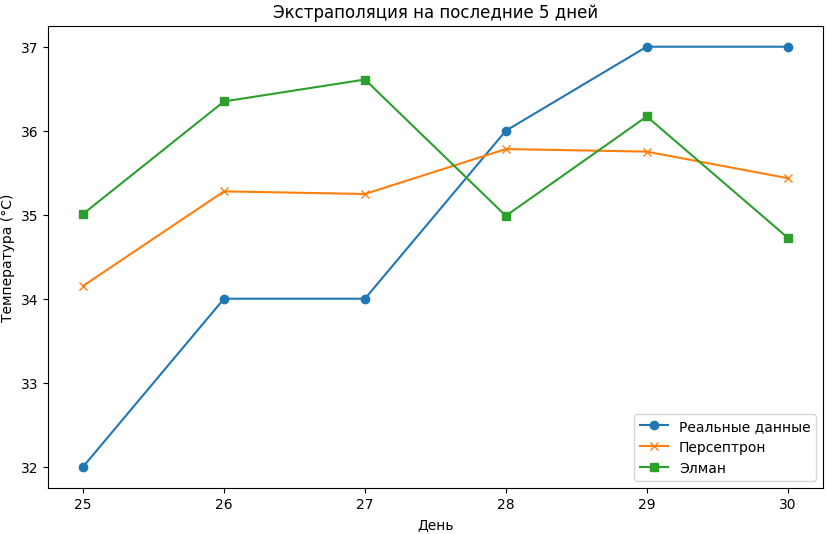


Рисунок 3 – предсказание на следующие 5 дней

Текст аппроксимации данных представлен в приложении А.

Текст экстраполяции данных представлен в приложении Б.

**Вывод:** в ходе выполнения лабораторной работы было выявлены различия качества аппроксимации и экстраполяции данных для разных видов нейронных сетей. Однослойных персепрон позволяет добиться лучшего качества апроксимации в сравнении с RBF сетью. В свою очередь двухслойный показывает большую точность в сравнении с однослойным. В силу заложенных архитектурных ограничений, персептрон не способен находить скрытых паттернов и закономерностей, вследствии чего хуже экстраполирует данные, чем сеть Элмана, которая может учитывать значения на предыдущем шаге обучения.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст аппроксимации данных**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error

import pandas as pd

# --- Function Definition ---

def target\_function(x):

return x\*\*3 \* np.exp(np.sin(x) \* np.cos(x))

# --- Radial Basis Function Network ---

# RBF activation function

def rbf(x, center, sigma):

return np.exp(-((x - center)\*\*2) / (2 \* sigma\*\*2))

# Build the RBF network output

def rbf\_network(x, centers, sigma, weights):

rbf\_outputs = np.array([rbf(x, c, sigma) for c in centers]).T

return rbf\_outputs @ weights

# Train RBF network (solve weights using least squares)

def train\_rbf(x, y, centers, sigma):

# Calculate RBF outputs for all centers

rbf\_outputs = np.array([rbf(x, c, sigma) for c in centers]).T

# Solve weights using least squares

weights = np.linalg.pinv(rbf\_outputs) @ y

return weights

# --- Single-layer Perceptron ---

# Activation function (ReLU)

def relu(x):

return np.maximum(0, x)

# Derivative of ReLU

def relu\_derivative(x):

return np.where(x > 0, 1, 0)

# Single-layer Perceptron model

class SingleLayerPerceptron:

def \_\_init\_\_(self, input\_size, hidden\_neurons, learning\_rate=0.01, epochs=10000):

self.input\_size = input\_size

self.hidden\_neurons = hidden\_neurons

self.learning\_rate = learning\_rate

self.epochs = epochs

self.weights\_hidden = np.random.randn(input\_size, hidden\_neurons)

self.bias\_hidden = np.random.randn(hidden\_neurons)

self.weights\_output = np.random.randn(hidden\_neurons, 1)

self.bias\_output = np.random.randn(1)

def forward(self, x):

self.hidden\_input = np.dot(x, self.weights\_hidden) + self.bias\_hidden

self.hidden\_output = relu(self.hidden\_input)

self.output = np.dot(self.hidden\_output, self.weights\_output) + self.bias\_output

return self.output

def backward(self, x, y\_true, y\_pred):

output\_error = y\_pred - y\_true

grad\_weights\_output = np.dot(self.hidden\_output.T, output\_error) / x.shape[0]

grad\_bias\_output = np.sum(output\_error, axis=0) / x.shape[0]

hidden\_error = np.dot(output\_error, self.weights\_output.T) \* relu\_derivative(self.hidden\_input)

grad\_weights\_hidden = np.dot(x.T, hidden\_error) / x.shape[0]

grad\_bias\_hidden = np.sum(hidden\_error, axis=0) / x.shape[0]

self.weights\_output -= self.learning\_rate \* grad\_weights\_output

self.bias\_output -= self.learning\_rate \* grad\_bias\_output

self.weights\_hidden -= self.learning\_rate \* grad\_weights\_hidden

self.bias\_hidden -= self.learning\_rate \* grad\_bias\_hidden

def train(self, x, y):

for epoch in range(self.epochs):

y\_pred = self.forward(x)

self.backward(x, y, y\_pred)

# --- Two-layer Perceptron ---

class TwoLayerPerceptron:

def \_\_init\_\_(self, input\_size, hidden\_neurons\_1, hidden\_neurons\_2, learning\_rate=0.01, epochs=10000):

self.input\_size = input\_size

self.hidden\_neurons\_1 = hidden\_neurons\_1

self.hidden\_neurons\_2 = hidden\_neurons\_2

self.learning\_rate = learning\_rate

self.epochs = epochs

self.weights\_hidden\_1 = np.random.randn(input\_size, hidden\_neurons\_1)

self.bias\_hidden\_1 = np.random.randn(hidden\_neurons\_1)

self.weights\_hidden\_2 = np.random.randn(hidden\_neurons\_1, hidden\_neurons\_2)

self.bias\_hidden\_2 = np.random.randn(hidden\_neurons\_2)

self.weights\_output = np.random.randn(hidden\_neurons\_2, 1)

self.bias\_output = np.random.randn(1)

def forward(self, x):

self.hidden\_input\_1 = np.dot(x, self.weights\_hidden\_1) + self.bias\_hidden\_1

self.hidden\_output\_1 = relu(self.hidden\_input\_1)

self.hidden\_input\_2 = np.dot(self.hidden\_output\_1, self.weights\_hidden\_2) + self.bias\_hidden\_2

self.hidden\_output\_2 = relu(self.hidden\_input\_2)

self.output = np.dot(self.hidden\_output\_2, self.weights\_output) + self.bias\_output

return self.output

def backward(self, x, y\_true, y\_pred):

output\_error = y\_pred - y\_true

grad\_weights\_output = np.dot(self.hidden\_output\_2.T, output\_error) / x.shape[0]

grad\_bias\_output = np.sum(output\_error, axis=0) / x.shape[0]

hidden\_error\_2 = np.dot(output\_error, self.weights\_output.T) \* relu\_derivative(self.hidden\_input\_2)

grad\_weights\_hidden\_2 = np.dot(self.hidden\_output\_1.T, hidden\_error\_2) / x.shape[0]

grad\_bias\_hidden\_2 = np.sum(hidden\_error\_2, axis=0) / x.shape[0]

hidden\_error\_1 = np.dot(hidden\_error\_2, self.weights\_hidden\_2.T) \* relu\_derivative(self.hidden\_input\_1)

grad\_weights\_hidden\_1 = np.dot(x.T, hidden\_error\_1) / x.shape[0]

grad\_bias\_hidden\_1 = np.sum(hidden\_error\_1, axis=0) / x.shape[0]

self.weights\_output -= self.learning\_rate \* grad\_weights\_output

self.bias\_output -= self.learning\_rate \* grad\_bias\_output

self.weights\_hidden\_2 -= self.learning\_rate \* grad\_weights\_hidden\_2

self.bias\_hidden\_2 -= self.learning\_rate \* grad\_bias\_hidden\_2

self.weights\_hidden\_1 -= self.learning\_rate \* grad\_weights\_hidden\_1

self.bias\_hidden\_1 -= self.learning\_rate \* grad\_bias\_hidden\_1

def train(self, x, y):

for epoch in range(self.epochs):

y\_pred = self.forward(x)

self.backward(x, y, y\_pred)

# --- Data Preparation ---

x = np.linspace(-2, 2, 200)

y = target\_function(x)

x\_reshaped = x.reshape(-1, 1)

y\_reshaped = y.reshape(-1, 1)

# --- RBF Network Training ---

centers = np.linspace(-2, 2, 3)

sigma = 0.5

weights\_rbf = train\_rbf(x, y, centers, sigma)

y\_rbf = rbf\_network(x, centers, sigma, weights\_rbf)

# --- Single-layer Perceptron Training ---

slp = SingleLayerPerceptron(input\_size=1, hidden\_neurons=5, learning\_rate=0.01, epochs=5000)

slp.train(x\_reshaped, y\_reshaped)

y\_mlp\_single = slp.forward(x\_reshaped).flatten()

# --- Two-layer Perceptron Training ---

tlp = TwoLayerPerceptron(input\_size=1, hidden\_neurons\_1=5, hidden\_neurons\_2=3, learning\_rate=0.01, epochs=5000)

tlp.train(x\_reshaped, y\_reshaped)

y\_mlp\_double = tlp.forward(x\_reshaped).flatten()

# --- Visualization ---

plt.figure(figsize=(10, 6))

plt.plot(x, y, label="Target Function", color="black", linewidth=2)

plt.plot(x, y\_rbf, label="RBF Network (3 neurons)", linestyle="--", color="blue")

plt.plot(x, y\_mlp\_single, label="Single-layer Perceptron (5 neurons)", linestyle="-.", color="green")

plt.plot(x, y\_mlp\_double, label="Two-layer Perceptron (5+3 neurons)", linestyle=":", color="red")

plt.xlabel("x")

plt.ylabel("f(x)")

plt.title("Comparison of Function Approximation Methods")

plt.legend()

plt.grid(True)

plt.show()

# --- Error Calculation ---

errors = {

"Method": ["RBF Network", "Single-layer Perceptron", "Two-layer Perceptron"],

"Mean Squared Error": [

mean\_squared\_error(y, y\_rbf),

mean\_squared\_error(y, y\_mlp\_single),

mean\_squared\_error(y, y\_mlp\_double)

]

}

error\_df = pd.DataFrame(errors)

# Display errors

print(error\_df)

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Текст экстраполяции данных**

temperature\_data\_jan\_2019 = [34, 34, 38, 38, 32,

33, 37, 38, 34, 36,

36, 32, 33, 36, 37,

36, 36, 34, 38, 35,

37, 35, 33, 35, 35,

32, 34, 34, 36, 37,

37]

temperature\_data\_norm = (temperature\_data\_jan\_2019 - np.mean(temperature\_data\_jan\_2019)) / np.std(temperature\_data\_jan\_2019)

window\_size = 5

x\_data, y\_data = [], []

for i in range(len(temperature\_data\_norm) - window\_size):

x\_data.append(temperature\_data\_norm[i:i + window\_size])

y\_data.append(temperature\_data\_norm[i + window\_size])

x\_data = np.array(x\_data)

y\_data = np.array(y\_data)

train\_data = temperature\_data\_norm[:25]

test\_data = temperature\_data\_norm[25:]

def mean\_squared\_error(y\_true, y\_pred):

return np.mean((y\_true - y\_pred) \*\* 2)

def mean\_absolute\_error(y\_true, y\_pred):

return np.mean(np.abs(y\_true - y\_pred))

class SingleLayerPerceptron:

def \_\_init\_\_(self, n\_hidden=7):

self.n\_hidden = n\_hidden

self.weights\_hidden = np.random.randn(window\_size, n\_hidden)

self.weights\_output = np.random.randn(n\_hidden)

def activation(self, x):

return np.tanh(x)

def predict(self, x):

hidden\_output = self.activation(np.dot(x, self.weights\_hidden))

return np.dot(hidden\_output, self.weights\_output)

def train(self, x, y, lr=0.01, epochs=100):

for epoch in range(epochs):

for i in range(len(x)):

y\_pred = self.predict(x[i])

error = y[i] - y\_pred

for j in range(self.n\_hidden):

self.weights\_hidden[:, j] += lr \* error \* x[i]

self.weights\_output[j] += lr \* error \* self.activation(np.dot(x[i], self.weights\_hidden[:, j]))

slp = SingleLayerPerceptron(n\_hidden=7)

slp.train(x\_data, y\_data)

predicted\_slp\_train = []

input\_window = train\_data[:window\_size]

for i in range(len(train\_data) - window\_size):

pred = slp.predict(input\_window)

predicted\_slp\_train.append(pred)

input\_window = np.append(input\_window[1:], train\_data[i + window\_size])

predicted\_slp\_train = np.array(predicted\_slp\_train) \* np.std(temperature\_data\_jan\_2019) + np.mean(temperature\_data\_jan\_2019)

train\_data\_original = train\_data[window\_size:] \* np.std(temperature\_data\_jan\_2019) + np.mean(temperature\_data\_jan\_2019)

mse\_slp\_train = mean\_squared\_error(train\_data\_original, predicted\_slp\_train)

mae\_slp\_train = mean\_absolute\_error(train\_data\_original, predicted\_slp\_train)

print(f"Персептрон (обучение) - MSE: {mse\_slp\_train:.4f}, MAE: {mae\_slp\_train:.4f}")

class ElmanNetwork:

def \_\_init\_\_(self, n\_hidden=7):

self.n\_hidden = n\_hidden

self.weights\_input\_hidden = np.random.randn(window\_size, n\_hidden)

self.weights\_hidden\_hidden = np.random.randn(n\_hidden, n\_hidden)

self.weights\_output = np.random.randn(n\_hidden)

self.hidden\_state = np.zeros(n\_hidden)

def activation(self, x):

return np.tanh(x)

def predict(self, x):

self.hidden\_state = self.activation(np.dot(x, self.weights\_input\_hidden) + np.dot(self.hidden\_state, self.weights\_hidden\_hidden))

return np.dot(self.hidden\_state, self.weights\_output)

def train(self, x, y, lr=0.01, epochs=100):

for epoch in range(epochs):

for i in range(len(x)):

y\_pred = self.predict(x[i])

error = y[i] - y\_pred

for j in range(self.n\_hidden):

self.weights\_input\_hidden[:, j] += lr \* error \* x[i]

self.weights\_hidden\_hidden[j, :] += lr \* error \* self.hidden\_state

self.weights\_output[j] += lr \* error \* self.hidden\_state[j]

elman\_net = ElmanNetwork(n\_hidden=7)

elman\_net.train(x\_data, y\_data)

predicted\_elman\_train = []

input\_window = train\_data[:window\_size]

for i in range(len(train\_data) - window\_size):

pred = elman\_net.predict(input\_window)

predicted\_elman\_train.append(pred)

input\_window = np.append(input\_window[1:], train\_data[i + window\_size])

predicted\_elman\_train = np.array(predicted\_elman\_train) \* np.std(temperature\_data\_jan\_2019) + np.mean(temperature\_data\_jan\_2019)

mse\_elman\_train = mean\_squared\_error(train\_data\_original, predicted\_elman\_train)

mae\_elman\_train = mean\_absolute\_error(train\_data\_original, predicted\_elman\_train)

print(f"Сеть Элмана (обучение) - MSE: {mse\_elman\_train:.4f}, MAE: {mae\_elman\_train:.4f}")

plt.figure(figsize=(10, 6))

plt.plot(range(window\_size, 25), train\_data\_original, 'o-', label="Реальные данные")

plt.plot(range(window\_size, 25), predicted\_slp\_train, 'x-', label="Персептрон")

plt.plot(range(window\_size, 25), predicted\_elman\_train, 's-', label="Сеть Элмана")

plt.title("Обучение на первых 25 днях")

plt.xlabel("День")

plt.ylabel("Температура (°C)")

plt.legend()

plt.show()

predicted\_slp\_test, predicted\_elman\_test = [], []

input\_window = train\_data[-window\_size:]

for i in range(len(test\_data)):

pred = slp.predict(input\_window)

predicted\_slp\_test.append(pred)

input\_window = np.append(input\_window[1:], pred)

input\_window = train\_data[-window\_size:]

for i in range(len(test\_data)):

pred = elman\_net.predict(input\_window)

predicted\_elman\_test.append(pred)

input\_window = np.append(input\_window[1:], pred)

test\_data\_original = test\_data \* np.std(temperature\_data\_jan\_2019) + np.mean(temperature\_data\_jan\_2019)

predicted\_slp\_test = np.array(predicted\_slp\_test) \* np.std(temperature\_data\_jan\_2019) + np.mean(temperature\_data\_jan\_2019)

predicted\_elman\_test = np.array(predicted\_elman\_test) \* np.std(temperature\_data\_jan\_2019) + np.mean(temperature\_data\_jan\_2019)

mse\_slp\_test = mean\_squared\_error(test\_data\_original, predicted\_slp\_test)

mae\_slp\_test = mean\_absolute\_error(test\_data\_original, predicted\_slp\_test)

mse\_elman\_test = mean\_squared\_error(test\_data\_original, predicted\_elman\_test)

mae\_elman\_test = mean\_absolute\_error(test\_data\_original, predicted\_elman\_test)

print(f"Персептрон (тест) - MSE: {mse\_slp\_test:.4f}, MAE: {mae\_slp\_test:.4f}")

print(f"Сеть Элмана (тест) - MSE: {mse\_elman\_test:.4f}, MAE: {mae\_elman\_test:.4f}")

plt.figure(figsize=(10, 6))

plt.plot(range(25, 25 + len(test\_data\_original)), test\_data\_original, 'o-', label="Реальные данные")

plt.plot(range(25, 25 + len(predicted\_slp\_test)), predicted\_slp\_test, 'x-', label="Персептрон")

plt.plot(range(25, 25 + len(predicted\_elman\_test)), predicted\_elman\_test, 's-', label="Элман")

plt.title("Экстраполяция на последние 5 дней")

plt.xlabel("День")

plt.ylabel("Температура (°C)")

plt.legend()

plt.show()

**ПИЛОЖЕНИЕ В**

(обязательное)

**Текст графиков**

# Построение графиков

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

# График потерь

plt.subplot(1, 2, 1)

plt.plot(range(1, epochs + 1), losses, label="Loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.title("Loss per Epoch")

plt.legend()

plt.grid()

# График точности

plt.subplot(1, 2, 2)

plt.plot(range(1, epochs + 1), accuracies, label="Accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.title("Accuracy per Epoch")

plt.legend()

plt.grid()

plt.show()