Чернівецький національний університет імені Юрія Федьковича

Факультет математики та інформатики

Кафедра математичного моделювання

**Лабораторна робота №1**

з навчальної дисципліни «**Системи штучного інтелекту**»

на тему:

«Штучні нейронні мережі. Моделювання формальних логічних функцій. Прогнозування часових рядів»

Варіант №4

Виконав: студент 3 курсу

311 групи

спеціальності «Комп’ютерні науки»

Вайнагій Д.В.

Перевірив: доцент Юрченко І. В.

Чернівці – 2024

Завдання:

1) Написати програму для реалізації штучних нейронів та нейронних мереж

для:

− моделювання логічної функції І,

− моделювання логічної функції АБО,

− моделювання логічної функції НІ,

− моделювання логічної функції Виключне АБО,

− прогнозування часового ряду (див. № варіанту в таблиці).



2) Написати програму для реалізації штучної нейронної мережі, що моделює логічну функцію, таблиця істинності якої наведена нижче.



Хід Роботи.

Моделювання логічних функцій міститься у файлі task1.1.py

import tkinter as tk

from tkinter import ttk

def calculate\_result():

    selected\_operation = operation\_var.get()

    x2\_entry.configure(state='normal')

    if selected\_operation == "AND":

        calculate\_logic\_function(and\_neuron)

    elif selected\_operation == "OR":

        calculate\_logic\_function(or\_neuron)

    elif selected\_operation == "NOT":

        x2\_entry.configure(state='disabled')

        calculate\_logic\_function(not\_neuron)

    elif selected\_operation == "XOR":

        calculate\_logic\_function(xor\_neuron)

def calculate\_logic\_function(logic\_function):

    x1 = float(x1\_var.get())

    x1 = round\_value(x1)

    if operation\_var.get() != "NOT":

        x2 = float(x2\_var.get())

        x2 = round\_value(x2)

        result = logic\_function(x1, x2)

    else:

        result = logic\_function(x1, '')

    result\_var.set(result)

def round\_value(value):

    return 1 if value != 0 else 0

def and\_neuron(x1, x2):

    w1, w2 = 1, 1

    threshold = 1.5

    S = x1 \* w1 + x2 \* w2

    return 1 if S >= threshold else 0

def or\_neuron(x1, x2):

    w1, w2 = 1, 1

    threshold = 0.5

    S = x1 \* w1 + x2 \* w2

    return 1 if S >= threshold else 0

def not\_neuron(x, \_):

    w = -1.5

    threshold = -1

    S = x \* w

    return 1 if S >= threshold else 0

def xor\_neuron(x1, x2):

    w1, w2 = 1, -1

    threshold\_1 = 0.5

    S1 = x1 \* w1 + x2 \* w2

    output\_1 = 1 if S1 >= threshold\_1 else 0

    w3, w4 = -1, 1

    threshold\_2 = 0.5

    S2 = x1 \* w3 + x2 \* w4

    output\_2 = 1 if S2 >= threshold\_2 else 0

    w5, w6 = 1, 1

    threshold\_3 = 0.5

    S3 = output\_1 \* w5 + output\_2 \* w6

    return 1 if S3 >= threshold\_3 else 0

root = tk.Tk()

root.title("Логічні функції")

x1\_var = tk.StringVar()

x2\_var = tk.StringVar()

x1\_label = ttk.Label(root, text="x1:")

x1\_entry = ttk.Entry(root, textvariable=x1\_var)

x2\_label = ttk.Label(root, text="x2:")

x2\_entry = ttk.Entry(root, textvariable=x2\_var)

operation\_var = tk.StringVar()

operations = ["choice operation", "AND", "OR", "NOT", "XOR"]

operation\_label = ttk.Label(root, text="Operation:")

operation\_menu = ttk.OptionMenu(root, operation\_var, \*operations)

operation\_var.set(operations[0])

calculate\_button = ttk.Button(root, text="Calculate", command=calculate\_result)

result\_var = tk.StringVar()

result\_label = ttk.Label(root, text="Result:")

result\_entry = ttk.Entry(root, textvariable=result\_var, state="readonly")

x1\_label.grid(row=0, column=0, padx=5, pady=5)

x2\_label.grid(row=0, column=1, padx=5, pady=5)

x1\_entry.grid(row=1, column=0, padx=5, pady=5)

x2\_entry.grid(row=1, column=1, padx=5, pady=5)

operation\_label.grid(row=2, column=0, padx=5, pady=5)

operation\_menu.grid(row=2, column=1, padx=5, pady=5)

calculate\_button.grid(row=3, column=1, columnspan=2, pady=10)

result\_label.grid(row=4, column=0, padx=5, pady=5)

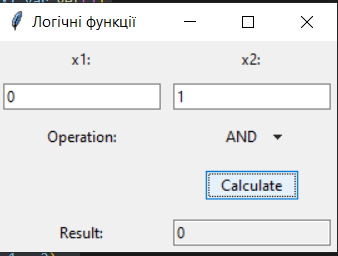
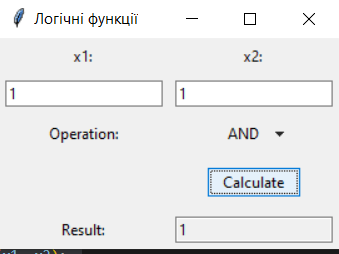
result\_entry.grid(row=4, column=1, padx=5, pady=5)

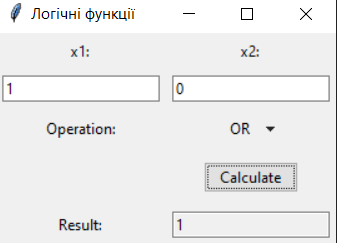
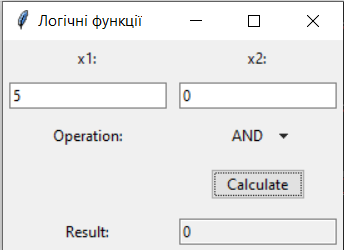
root.mainloop()

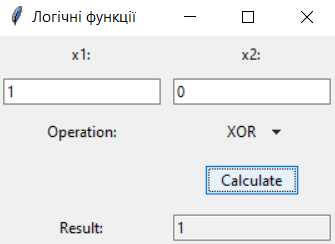
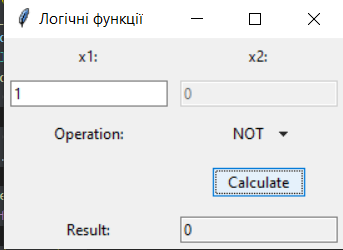
Код містить саме моделювання функцій та логіку реалізації інтерфейсу.

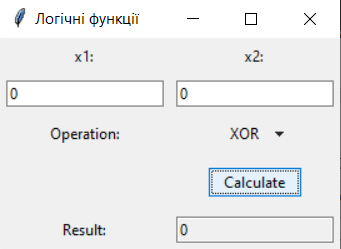
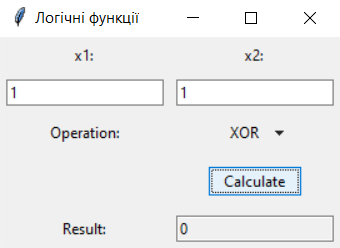
Також всі значення відмінні від 0 вважатимуться 1

Ось приклади виконання коду





Моделювання числового ряду міститься в файлі task1.2.py

import numpy as np

data = np.array([0.1, 5.9, 0.5, 4.0, 0.3, 5.5, 0.1, 4.5, 0.6, 4.3, 1.5, 4.7, 0.5, 5.8, 0.0])

scale = np.max(data)

class Neural:

    def \_\_init\_\_(self):

        self.weights = np.random.rand(2)

    def sigmoid(self, x):

        return 1/(1+ np.exp(-x))

    def predict(self, inputs):

        weighted\_sum = np.dot(inputs, self.weights)

        return self.sigmoid(weighted\_sum)

    def train(self, inputs, expected\_output, epochs):

        learning\_rate = 0.1

        for \_ in range(epochs):

            for i in range(len(inputs)):

                prediction = self.predict(inputs[i])

                error = expected\_output[i] - prediction

                self.weights += learning\_rate \* error \* prediction \* (1 - prediction) \* inputs[i]

X\_time\_series = np.array([ [0.1, 5.9], [5.9, 0.5], [0.5, 4.0], [4.0, 0.3], [0.3, 5.5],

    [5.5, 0.1], [0.1, 4.5], [4.5, 0.6], [0.6, 4.3], [4.3, 1.5], [1.5, 4.7]])/scale

y\_time\_series = np.array([0.5, 4.0, 0.3, 5.5, 0.1, 4.5, 0.6, 4.3, 1.5, 4.7, 0.5])/scale

neural = Neural()

neural.train(X\_time\_series, y\_time\_series, epochs=1000)

#Prediction for the new input values

predicted\_values = [neural.predict([4.7,0.5])\*scale]

predicted\_values = [neural.predict([0.5,5.8])\*scale]

print("Predicted value for x\_14:", neural.predict([4.7,0.5])\*scale,"Expected value=5.8")

print("Predicted value for x\_15:", neural.predict([0.5,5.8])\*scale,"Expected value=0")

Нейронна мережа навчатиметься на наборі данних з числового ряду і прогнозуватиме 14 та 15 член ряду(доступу до яких вона раніше ніколи немала)



Як ми можем побачити точність прогнозування дуже висока(прим. Для 15 члена програма відображає результат у вигляді числа з плаваючою крапкою)

Код програми що моделює нейронну мережу таблиця істинності якої нам відома представлена у файлі task2.py

class Custom\_Neuron:

    def \_\_init\_\_(self):

        self.weights = [0, 0, 0]  # Ваги для x1, x2, x3

        self.bias = 0

        self.learning\_rate = 0.1

    def activate(self, weighted\_sum):

        return 1 if weighted\_sum >= 0 else 0

    def predict(self, inputs):

        weighted\_sum = sum(w \* x for w, x in zip(self.weights, inputs)) + self.bias

        return self.activate(weighted\_sum)

    def train(self, training\_inputs, labels, epochs):

        for \_ in range(epochs):

            for inputs, label in zip(training\_inputs, labels):

                prediction = self.predict(inputs)

                if prediction != label:

                    error = label - prediction

                    for i in range(len(self.weights)):

                        self.weights[i] += error \* inputs[i] \* self.learning\_rate

                    self.bias += error \* self.learning\_rate

# Таблиця істинності для навчання

training\_inputs = [(0, 0, 0), (0, 1, 0), (1, 0, 0), (1, 1, 1)]

labels = [1, 1, 0, 1]

neuron = Custom\_Neuron()

neuron.train(training\_inputs, labels, epochs=1000)

print("x1 x2 x3 y")

for inputs, label in zip(training\_inputs, labels):

    result = neuron.predict(inputs)

    print(f"{inputs[0]}  {inputs[1]}  {inputs[2]}  {result}")

Програма навчає нейронну мережу базуючись на таблиці істинності.

Після навчання отримуємо ось такі результати, що повністю відповідає таблиці

