НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи №2
з дисципліни «Системи нейронних мереж»
на тему
«Розробка програмного забезпечення для реалізації ймовірнісної нейронної мережі PNN »

Виконав:
студент групи КМ-02
Пилипченко Б. О.

Керівник:
Терейковський І. А.

Київ—2023

Мета роботи:

розробити програмне забезпечення для реалізації мережі PNN, що призначена для апроксимації функції y=x1+x2. Передбачити режими навчання та розпізнавання

Теоретичні відомості:

Роберт Каллан Основні концепції нейронних мереж ст. 152—164

Теоретичні відомості

PNN (Probabilistic Neural Network)

В мережі PNN зразки класифікуються за допомогою оцінки їхньої близькості до сусідніх зразків.

Одним із методів, на основі якого можна класифікувати зразок, є обчислення центроїда кожного класу (тобто середньої характеристики розміщення усіх зразків класу). Такий метод чудово підійде для розподілів, подібних до рис. 1, але може призводити до помилок при розподілах, подібних до рис. 2.
Інший популярний метод заснований на використанні "найближчого сусіда". Такий метод теж не завжди працюватиме (рис. 3)

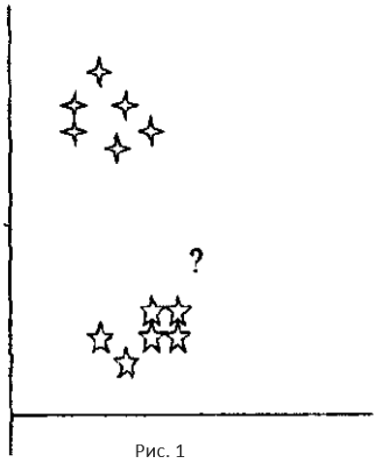


Рис. 1

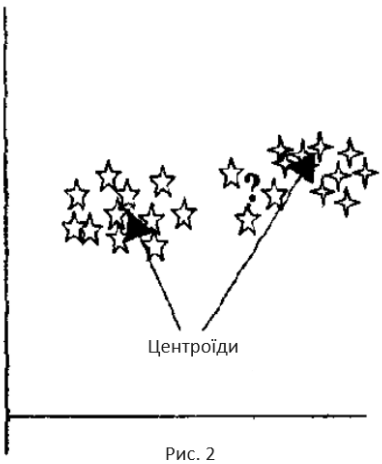


Рис. 2

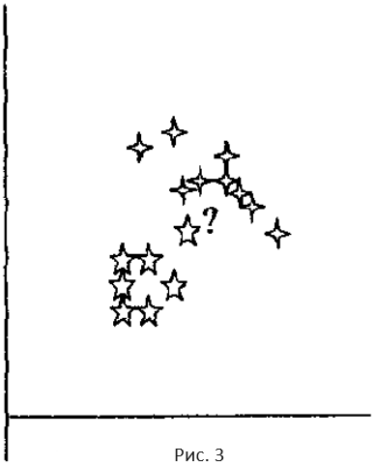


Рис. 3

PNN мережа проводитиме класифікацію ефективно в усіх трьох випадках - на відміну від представлених методів, PNN враховує густину розподілу елементів одного класу.

Функція Гауса:

$$g(\mathbf{x}) = \sum_{i=1}^{n} \exp\left(\frac{-\| \mathbf{x} - \mathbf{x}_i \|^2}{\sigma^2} \right)$$

x - вектор характеристик досліджуваного зразка
xi - вектор характеристик i-того класу
σ - параметр розтягнення функції.

Архітектура PNN мережі

PNN мережа складається з трьох шарів:
1. Вхідний шар:
   Вхідний шар розподіляє характеристики вхідного зразка на шар зразків. Кожен нейрон вхідного шару пов'язаний із кожним нейроном шару зв'язків.
2. Шар зразків:
   Шар зразків має по одному нейрону для кожного зразка. Вагові коефіцієнти вхідних сигналів нейрону дорівнюють значенням характеристик відповідного зразка.
   Функція активації нейрону шару зразків:

$$O_j = \exp\left(\frac{-\sum (w_{ij} - x_i)^2}{\sigma^2} \right)$$

Oj—характеристика близькості вхідного зразка до j-того зразка класу;
xi—i-та характеристика вхідного зразка;
wij—i-та характеристика j– того зразка класу;
σ—параметр мережі.

3. Шар класів:
   Шар класів має по одному нейрону для кожного класу. Кожен нейрон шару класів має зв'язки лише з нейронами зв'язків, які належать до класу. Всі вагові коефіцієнти між шаром зразків та шаром класів рівні 1.
   Функція активації нейрону шару класів є сумою вхідних сигналів нейрону.

PNN мережа не потребує навчання, подібного до навчання мереж із оберненим поширенням сигналу. Всі параметри мережі PNN, крім значення сігми, визначаються безпосередньо навчальними даними:
Кількість нейронів вхідного шару = кількість характеристик зразків
Кількість нейронів шару зразків = кількість зразків
Кількість нейронів шару класів = кількість класів
Навчання мережі полягає у підборі оптимального значення сігми.
В роботі сігма підбиралась експериментально.

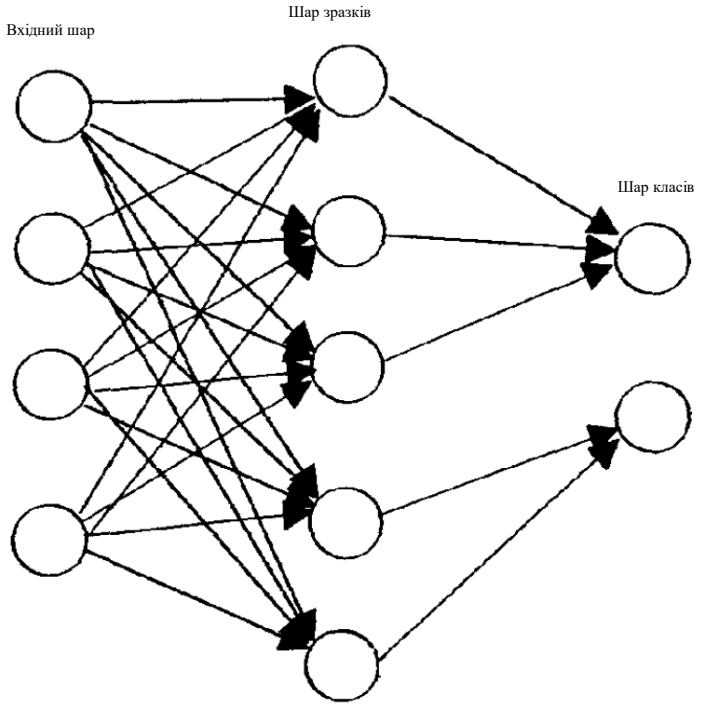


Рис. 4 Приклад PNN мережі (4 характеристики, 5 зразків, 2 класи)

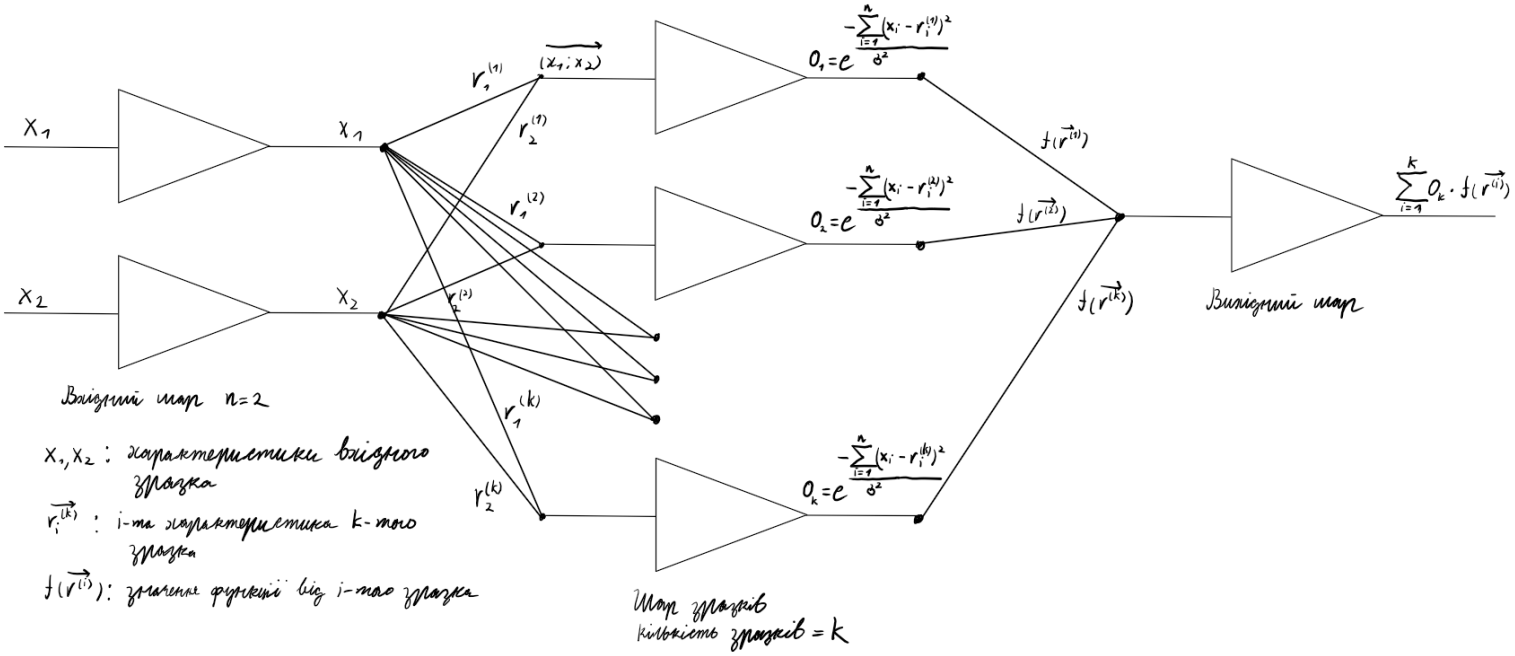
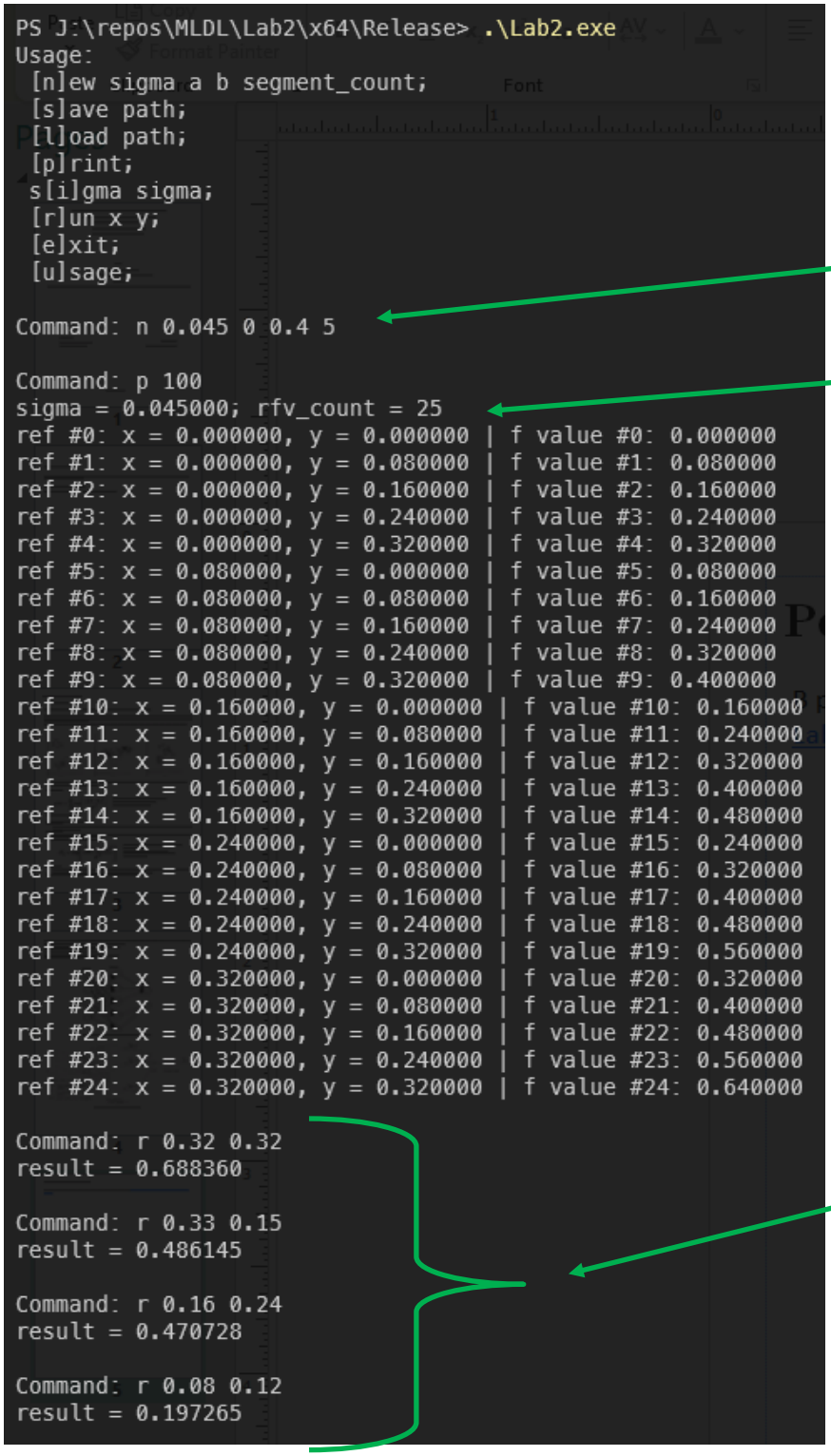


Схема PNN мережі, реалізованої в роботі

Реалізація/демонстрація

В роботі було реалізовано PNN мережу мовою C. Посилання на репозиторій: https://github.com/Bohdan628318ylypchenko/MLDL-Lab2.git

Демонстрація роботи PNN мережі:



Створення нової мережі. Sigma = 0.045, a = 0, b = 0.4, segment_count = 5

Новостворена мережа. Мережа апроксимує функцію в межах квадрату (a, a), (a, b), (b, a), (b, b)

За рахунок експериментально підібраного значення сігми
маємо мережу, що обчислює "адекватні" значення як для відомих прикладів,
так і для довільних прикладів.

Selected files

3 printable files

pnn_io.c
pnn.c
pnn.h

pnn_io.c

```c
#include "pnn.h"

#include <stdio.h>
#include <stdlib.h>

void pnn_fread(pnn * nn, FILE * stream)
{
    fread(&(nn->sigma), sizeof(double), 1, stream);

    int rfv_count;
    fread(&rfv_count, sizeof(int), 1, stream);
    nn->rfv_count = rfv_count;

    nn->refs = (v2 *)malloc(sizeof(v2) * rfv_count);
    nn->f_vals = (double *)malloc(sizeof(double) * rfv_count);
    fread(nn->refs, sizeof(v2), rfv_count, stream);
    fread(nn->f_vals, sizeof(double), rfv_count, stream);
}

void pnn_fwrite(pnn * nn, FILE * stream)
{
    fwrite(&(nn->sigma), sizeof(double), 1, stream);
    fwrite(&(nn->rfv_count), sizeof(int), 1, stream);

    fwrite(nn->refs, sizeof(v2), nn->rfv_count, stream);
    fwrite(nn->f_vals, sizeof(double), nn->rfv_count, stream);
}

void pnn_fprint(pnn * nn, FILE * stream, int head_count)
{
    fprintf(stream, "sigma = %lf; rfv_count = %d\n", nn->sigma, nn->rfv_count);

    for (int i = 0; (i < nn->rfv_count) && (i < head_count); i++)
        fprintf(stream, "ref #%d: x = %lf, y = %lf | f value #%d: %lf\n",
                i, nn->refs[i].x, nn->refs[i].y, i, nn->f_vals[i]);
}

```

pnn.c

```c
#include "pnn.h"

#include <stdlib.h>
```

```c
#include <math.h>

static double act(v2 * x, v2 * ref, double sigma);
static double eucl2(v2 * a, v2 * b);

void pnn_new(double sigma,
             double a, double b, int segment_count,
             pnn * nn,
             double(*f)(double, double))
{
    int rfv_count = segment_count * segment_count;

    nn->sigma = sigma;
    nn->rfv_count = rfv_count;
    nn->refs = (v2 *)malloc(sizeof(v2) * rfv_count);
    nn->f_vals = (double *)malloc(sizeof(double) * rfv_count);

    double delta = (b - a) / (double)segment_count;
    double x, y;
    int k;
    for (int i = 0; i < segment_count; i++)
    {
        x = a + (double)i * delta;
        for (int j = 0; j < segment_count; j++)
        {
            y = a + (double)j * delta;

            k = i * segment_count + j;

            nn->refs[k].x = x;
            nn->refs[k].y = y;
            nn->f_vals[k] = f(x, y);
        }
    }
}

double pnn_run(pnn * nn, v2 * x)
{
    double result = 0;

    for (int i = 0; i < nn->rfv_count; i++)
        result += act(x, &(nn->refs[i]), nn->sigma) * nn->f_vals[i];

    return result;
}

static double act(v2 * x, v2 * ref, double sigma)
{
    return exp((-1.0 * (eucl2(x, ref)) / (sigma * sigma)));
}

static double eucl2(v2 * a, v2 * b)
{
    double x = b->x - a->x;
    double y = b->y - a->y;
    return x * x + y * y;
```

```
}

```

pnn.h

```c
#pragma once

#include <stdio.h>

typedef struct
{
    double x;
    double y;
} v2;

typedef struct
{
    double sigma;
    int rfv_count;
    v2 * refs;
    double * f_vals;
} pnn;

void pnn_new(double sigma,
             double a, double b, int segment_count,
             pnn * nn,
             double(*f)(double, double));

double pnn_run(pnn * nn, v2 * x);

void pnn_fread(pnn * nn, FILE * stream);

void pnn_fwrite(pnn * nn, FILE * stream);

void pnn_fprint(pnn * nn, FILE * stream, int head_count);

```