

Sieć rekurencyjna w problemie kWTA

1. Cel projektu

Celem projektu jest wykonanie programu realizującego wybór k największych spośród N zadanych wartości n nieujemnych przy zastosowaniu rekurencyjnej sieci neuronowej. Problem wyboru wyrażono jako dynamiczny proces, w którym wyjścia neuronów o mniejszej wartości są hamowane i tłumione aż do osiągnięcia wartości zerowej. Dodatkowo zmodyfikowano algorytm w taki sposób aby w przypadku niepowodzenia mógł dynamicznie powtarzać proces zbiegania do wyniku ze zmienioną wartością parametrów.

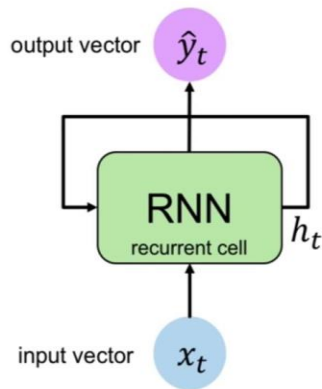
2. Wstęp

Ogólnie o sieciach neuronowych

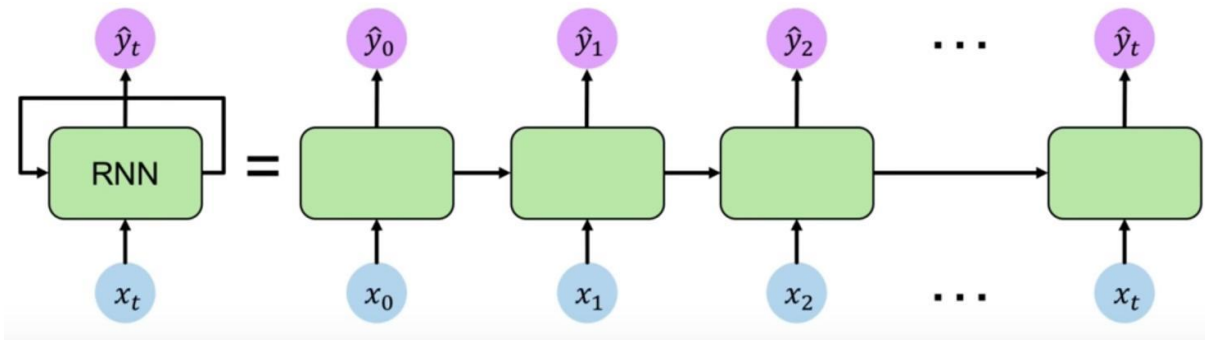
W ciągu ostatnich dekad obserwuje się burzliwy rozwój badań nad sztucznymi sieciami neuronowymi. Badania te są prowadzone przez fizyków i inżynierów, ale ich podstawą są odkrycia w dziedzinie fizjologii układu nerwowego organizmów żywych, a zwłaszcza funkcjonowania mózgu ludzkiego [1]. Sieci neuronowe tworzą modele neuronów połączone w odpowiednie struktury, np. jednokierunkowe wielowarstwowe, rekurencyjne ze sprzężeniami zwrotnymi bądź komórkowe. Działanie sieci neuronowych jest wypadkową działania poszczególnych neuronów oraz zachodzących między nimi interakcji.

Rekurencyjne sieci neuronowe

W sieciach rekurencyjnych istnieją sprzężenia zwrotne między wejściem, a wyjściem (rys. 1, rys. 2). Zależności dynamiczne jakie panują w sieci są widoczne na każdym etapie działania. Zmiana stanu jednego neuronu przenosi się przez masowe sprzężenie zwrotne na całą sieć, wywołując stan przejściowy, kończący się określonym stanem ustalonym, na ogół innym niż stan poprzedni [2]. Połączenie wyjścia z wejściem to tak zwane połączenie hamujące. Połączenie to realizuje sprzężenie zwrotne. Aktywacje w sieciach rekurencyjnych ustalają się w wyniku relaksacji (proces dynamicznych zmian). Rekurencyjne sieci nadają się do rozpoznawania obrazów, rozpoznawania mowy, budowania systemów pomagających w analizie tekstu, a także do rozwiązywania problemów minimalizacji [3].



Rys. 1. Graficzne przedstawienie rekurencyjnej sieci neuronowej. Strzałka symbolizuje sprzężenie zwrotne.



Rys. 2. Graficzna interpretacja sprzężenia zwrotnego w postaci wektora oddzielnych neuronów. Każdy neuron przesyła pewien sygnał do kolejnego neuronu.

Sieć kWTA

Sieć kWTA jest rozwinięciem sieci WTA. Sieć WTA (winners take all) jest jednowarstwową siecią neuronową, która posiada nieliniową funkcję aktywacji ReLU (wzór 1.1) oraz odpowiednie wagi hamujące (wzór 1.2).

(1.1)

$$f(x) = \max(0, x)$$

(1.2)

$$h_{im} = \begin{cases} \varepsilon & \text{gdy } i \neq m \\ +1 & \text{gdy } i = m \end{cases}$$

Modyfikując sieć WTA w taki sposób aby szukała ona k zwycięzców zamiast jednego następuje poprzez dodanie dodatkowej warstwy [4]. Warstwa ta zlicza liczbę wyjść o zerowej wartości i przerywa działanie algorytmu gdy liczba zerowych wyjść osiągnie żądaną wartość.

3. Realizacja projektu

Projekt został zrealizowany w języku **C++** zaś interfejs graficzny został wykonany z wykorzystaniem wieloplatformowego frameworku graficznego **QT**.

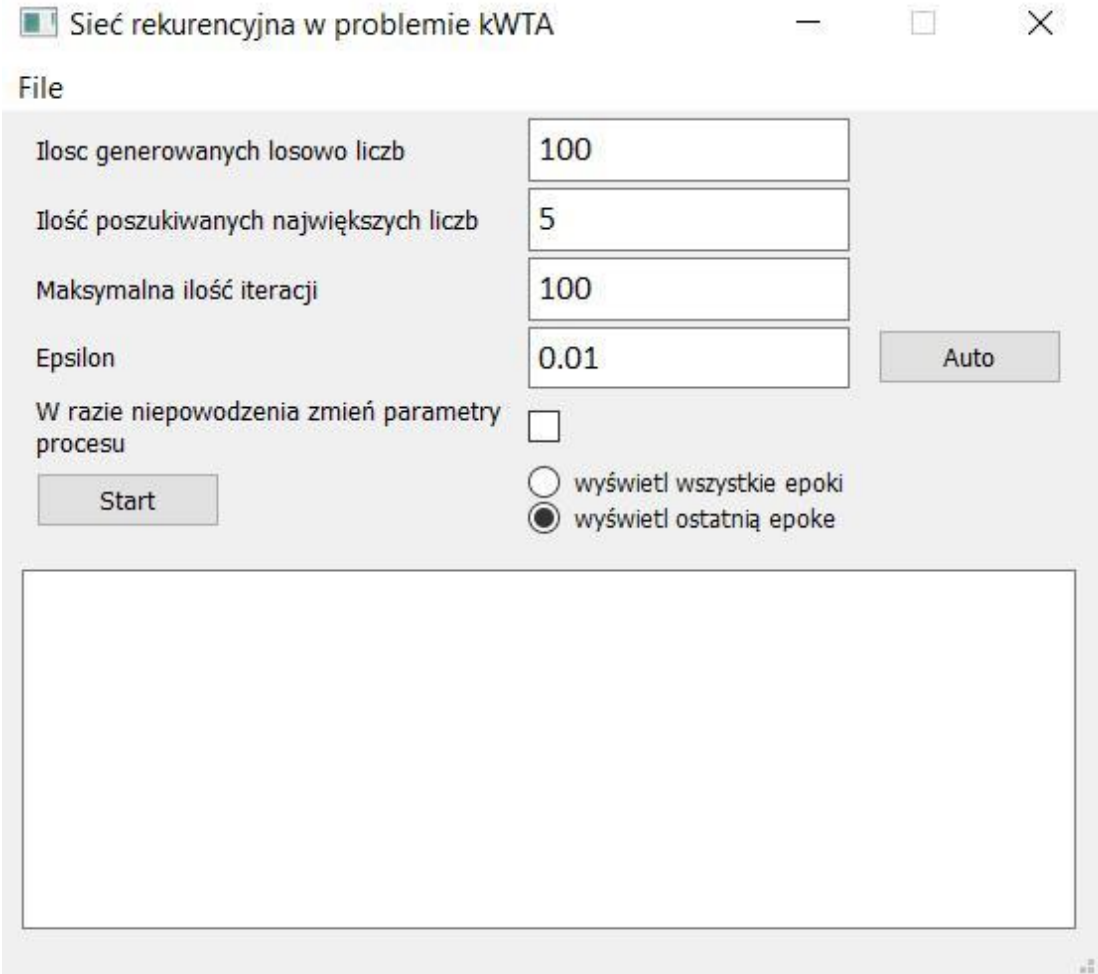
Program pozwala na automatyczne generowanie danych wejściowych lub wczytanie ich z pliku tekstowego. W przypadku automatycznego generowania danych użytkownik definiuje jedynie ile liczb ma zostać wygenerowanych. W przypadku importowania gotowych danych z pliku tekstowego program ładuje odpowiednie dane i oblicza rozmiar wektora wejściowego. Interfejs (rys. 3) pozwala na określenie ilu największych liczb poszukujemy, jaka ma być maksymalna liczba iteracji oraz jaką wartość ma mieć parametr epsilon będący wagą hamującą. Jeśli użytkownik nie wie jak dobrać odpowiedni parametr, w programie zaimplementowano funkcję "auto" obliczającą wynik parametru epsilon ze wzoru 2.1.

(2.1)

$$\varepsilon = \frac{1}{n - k}$$

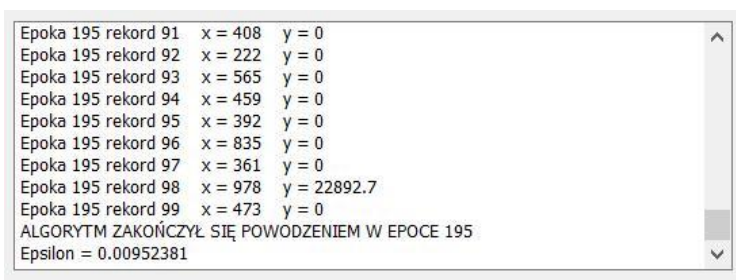
n – ilość liczb w wektorze wejściowym

k - poszukiwana ilość największych liczb

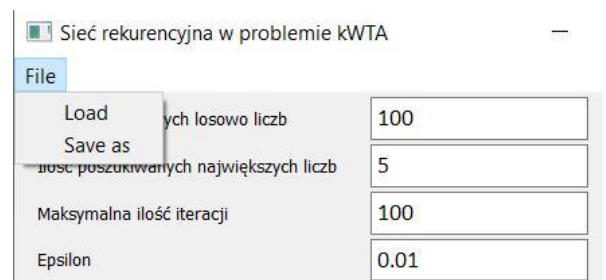


Rys. 3. Główne okno programu

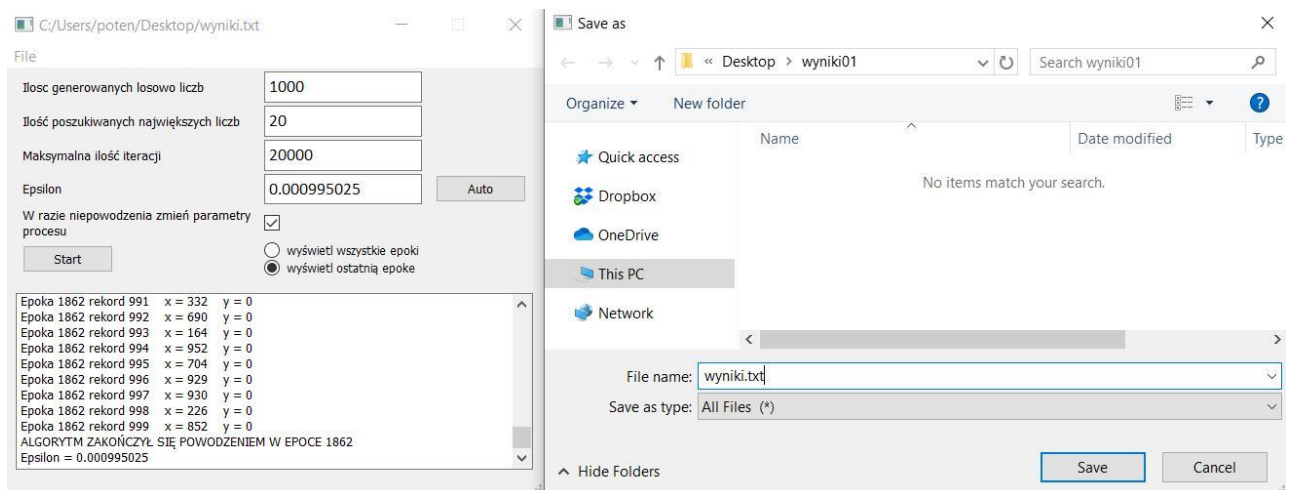
W trakcie trwania procesu wyniki działania programu są wyświetlane w dolnej części okna programu (rys. 4). Użytkownik może określić, czy chce uzyskać podgląd wartości wyjściowych wszystkich iteracji uczenia czy tylko ostatniej. W przypadku dużych zbiorów danych wyświetlanie wszystkich iteracji jest procesem bardzo czasochłonnym i wtedy lepiej jest skorzystać z okrojonej wersji raportu. Jeśli program przekroczy dozwoloną maksymalną ilość iteracji zdefiniowaną przez użytkownika algorytm zostanie przerwany a w okienku komunikatów pojawi się informacja o braku uzyskania rozwiązania. Użytkownik może zażądać, aby program w razie niepowodzenia powtarzał proces zbiegania do wyniku ze zmienioną wartością parametrów.



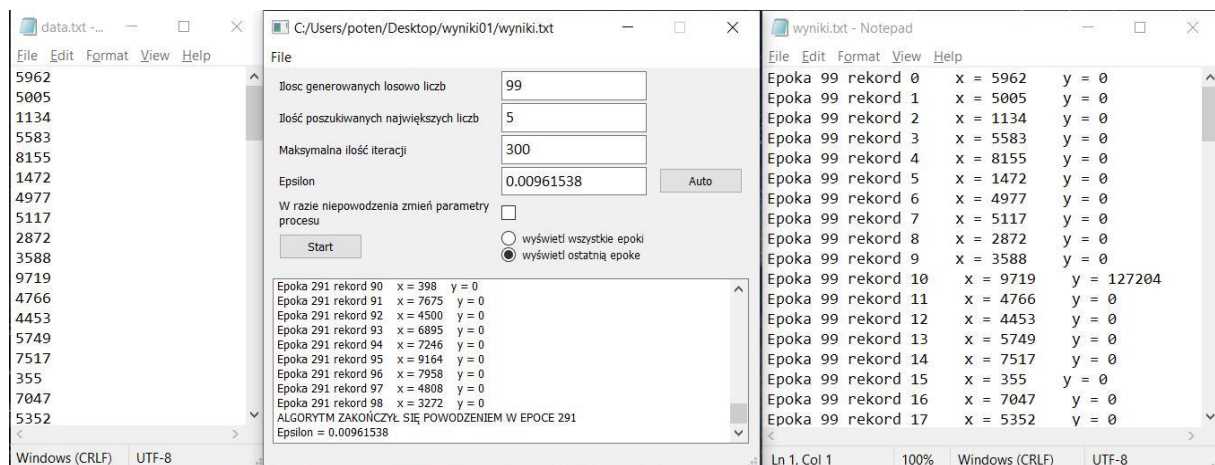
Rys. 4. Pole służące do wyświetlania danych monitorujących wykonywanie programu a także komunikacji z użytkownikiem.



Rys. 5. Górne menu do wczytywania danych lub zapisu wyników działania algorytmu.



Rys. 6. Zapis wyników do pliku monitorującego.



Rys. 7 Zestawienie pliku z danymi wejściowymi, wyników działania programu i pliku monitorującego wygenerowanego przez program.

W przypadku uzyskania pożądanych wyników, użytkownik może utworzyć plik monitorujący (rys. 6) zawierający pełne lub skrócone dane uwzględniające informacje o ilości epok, wartościach wejściowych, współczynniku epsilon oraz wartościach wyjściowych.

4. Implementacja projektu

Ogólne spojrzenie:

Projekt składa się z 3 klas: main, mainwindow, rnn.

Klasa **main** odpowiada jedynie za utworzenie i zamknięcie okna programu.

Klasa **mainwindow** odpowiada za zarządzanie interfejsem graficzny.

Klasa **rnn** odpowiada za właściwe działanie rekurencyjnej sieci neuronowej.

Szczegółowa analiza:

Klasa **main** składa się z jednej funkcji **int main(int argc, char *argv[])**, w której dokonywana jest inicjalizacja okna projektu, inicjalizacja generatora liczb pseudolosowych oraz po zakończeniu działania programu zamknięcie okna projektu.

Klasa **mainwindow** składa się z następujących funkcji:

- **void on_actionSave_as_triggered()**

Odpowiada za zapisanie wyników działania programu do pliku kontrolnego. Funkcja tworzy strumień danych a następnie przekierowuje go do pliku o nazwie podanej przez użytkownika. W przypadku wystąpienia błędu generuje odpowiedni komunikat.

- **void on_actionLoad_triggered();**

Odpowiada za wczytanie danych z pliku tekstowego znajdującego się w podanej przez użytkownika lokalizacji. Plik z danymi musi być w formacie txt oraz składać się z jednej kolumny danych. Na końcu pliku musi się znajdować napis END. Funkcja sama zlicza wielkość zbioru danych, stąd też nie jest konieczne określenie liczności zbioru wejściowego.

- **void on_start_QpushButton_clicked();**

Odpowiada za rozpoczęcie działania algorytmu realizującego wybór k największych wartości spośród N zadanych wartości n nieujemnych poprzez pobranie wartości z pól tekstowych okna użytkownika. Wartości te są rzutowane ze zmiennych typu QString na odpowiednie typy danych. Sprawdzane są także warunki związane z wyświetlaniem komunikatów pracy programu (czy wyświetlane mają być wyniki wszystkich iteracji, czy tylko ostatniej) oraz czy w razie niepowodzenia znalezienia wartości największych program powinien wykonać się ponownie ze zmienionymi parametrami procesu. Po sprawdzeniu wszystkich warunków wywoływana jest funkcja kWTA z obiektu klasy RNN z odpowiednimi parametrami.

- **void on_pushButton_clicked();**

Odpowiada za obliczenie odpowiedniej wartości parametru Epsilon w przypadku jeśli użytkownik nie będzie chciał wpisać tej wartości ręcznie.

Klasa RNN składa się z następujących funkcji:

- **bool kWTA(double epsilon, int n, int k, int maxiter, QTextEdit *terminal_QTextEdit, bool fullPrint, bool repeat);**

Funkcja zawierająca logikę sieci rekurencyjnej. Wywoływana jest w funkcji on_start_QpushButton_clicked() znajdującej się w klasie mainwindow. Jako parametry przyjmuje wartość Epsilon, wielkość zbioru wejściowego, ilość poszukiwanych liczb największych oraz flagi wyświetlania komunikatów i powtarzania procesu ze zmienionymi wartościami parametrów. Jej budowa jest bardzo podobna do uproszczonego algorytmu sieci rekurencyjnej w problemie kWTA zamieszczono w dodatku 1.

- **void generateNumber(int n);**

Odpowiada za generowanie liczb losowych do utworzenia danych wejściowych programu w przypadku gdy użytkownik nie załaduje ich z pliku. Jako argument wejściowy przyjmuje ilość liczb, jakie mają zostać wygenerowane. Funkcja losuje całkowite liczby nieujemne z przedziału (0;1000).

- **void loadNumber(int* tab, int n);**

Odpowiada za alokację pamięci dla wektora liczb pobranych z pliku. Nie pobiera ona danych z pliku, a jedynie tworzy odpowiedni wektor zmiennych.

~RNN();

Destruktor klasy RNN. Zwalnia pamięć która została dynamicznie zaalokowana w funkcji loadNumber(int* tab, int n).

Uproszczony algorytm sieci rekurencyjnej w problemie kWTA zamieszczono w dodatku 1. W celu lepszej czytelności kod ten zapewnia jedynie podstawowe działanie sieci (jego bardziej zaawansowana wersja została użyta w projekcie).

5. Omówienie wyników

Program działa poprawnie dla dowolnych zbiorów liczb, jednak jego skuteczność oraz czas działania zależą od wielkości zbioru wejściowego i odpowiedniego doboru parametrów sieci. Zależności między tymi wielkościami zestawiono w Tabeli 1.

Tabela 1. Wyniki działania programu dla różnych parametrów i zbiorów danych.

Wielkość zbioru	Ilość poszukiwanych wartości największych	Epsilon	Średnia liczba epok potrzebnych na zrealizowanie celu dla 3 prób.
100	50	0,1	BRAK ZBIERZNOŚCI
100	50	0,01	3
100	5	0,01	240
100	5	0,001	2452
1000	50	0,01	BRAK ZBIERZNOŚCI
1000	50	0,001	597
1000	5	0,001	4822
1000	5	0,0001	4891

- Im większy jest parametr epsilon, tym szybciej wyznaczany jest zbiór rozwiązań, jednak gdy jego wartość będzie zbyt duża algorytm nie uzyska zbieżności. Zbyt duża liczba wyjść zostanie zahamowana i stłumiona do osiągnięcia wartości 0 już po pierwszej epoce.

- Im więcej wartości największych poszukujemy, tym mniej epok potrzeba na ich znalezienie. Wynika to z prostego faktu – im mniej wyjść musimy stłumić do 0 tym szybciej cel zostanie osiągnięty.

- Im większy jest zbiór, tym czas wyznaczenia poszukiwanych wartości największych będzie większy.

6. Bibliografia

- [1] „Sztuczne sieci neuronowe” Robert A. Kosiński
- [2] Wykłady „Sztuczna inteligencja” Paweł Jarosz, Politechnika Krakowska
- [3] „Odkrywanie właściwości sieci neuronowych”, Ryszard Tadeusiewicz
- [4] Wykłady z przedmiotu „Metody Sztucznej inteligencji” Włodzimierz Kasprzak, Politechnika Warszawska

Dodatek 1. Uproszczony kod sieci rekurencyjnej w problemie kWTA.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    int n = 100;           // wymiar wektor liczb
    int k = 5;             // liczba zwyciezow
    int maxiter = 1000;    // maksymalna liczba iteracji
    double sumuvec = 0;
    double epsilon = 1.0 / (n + k);
    int zeronum = 0;
    int iternum = 0;

    //***** tworzenie wektora liczb i inicjalizacja wartosci poczatkowych *****
    srand(time(NULL));
    int* u = new int[n];    // wektor wartosci wejsciowych
    double* z = new double[n]; // wektor wartosci wyjsciowych
    double* newz = new double[n];

    for (int i = 0; i < n; i++)
    {
        u[i] = rand()%1000;
        // dla zerowej iteracji wartosci wyjsciowe poprzedniej iteracji sa sztucznie tworzone
        z[i] = u[i];
        newz[i] = u[i];
    }

    //***** rekurencyjna siec neuronowa *****
    for (int i = 0; i < maxiter; i++)
    {
        sumuvec = 0;        // suma wyjsc dla calego zbioru
        zeronum = 0;        // liczba zerowych wyjsc
        iternum = maxiter;

        for (int j = 0; j < n; j++)
        {
            sumuvec += z[j];
        }
        //wlascswa czesc uczenia sieci
        for (int j = 0; j < n; j++)
        {
            newz[j] = u[j] + z[j] - epsilon * (sumuvec - z[j]);
            if (newz[j] < 0)    // FUNKCJA AKTYWACJI ReLU
            {
                newz[j] = 0;
                zeronum += 1;
            }
        }
        //synchronizacja nowych wyjsc ze starymi
        for (int j = 0; j < n; j++)
        {
            z[j] = newz[j];
        }
        //warunek znalezienia odpowiedniej liczby największych liczb
        if (zeronum >= n - k)
        {
            iternum = i;
            break;
        }
    }
}
```