26.06.2018 r.

Małgorzata Sierbin
Robert Greliński
TI1-Z3

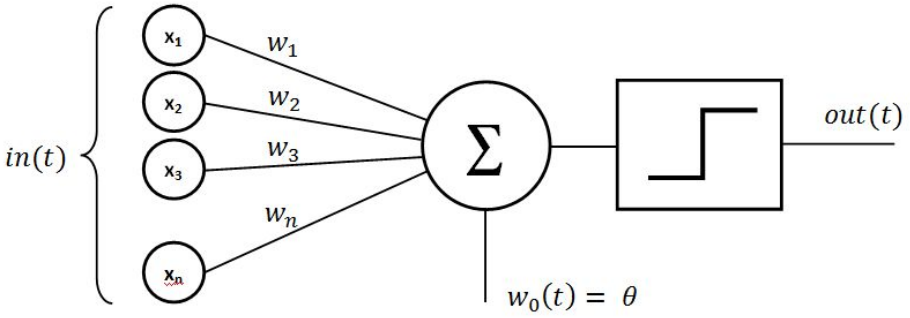
Sprawozdanie z laboratorium sztucznej inteligencji

Informatyka, sem. VI

I. Opis zadania

Wykonać symulator sieci typu *spiking neural network*, czyli pulsujących sieci neuronowych.
Założenia i wymagania dotyczące programu:

- dowolna liczba wejść,
- losowanie wartości wag,
- ręczny wybór wartości wag,
- działanie w czasie rzeczywistym,
- losowanie wartości na wejściu,
- podgląd uzyskanej wartości na wyjściu,
- podgląd uzyskanej wartości jako potencjał na membranie,
- obliczanie wielowątkowe,
- możliwość ciągłego działania programu,
- możliwość budowy symulacji całej sieci neuronowej.



Rys. 1. Działanie neuronu

Rysunek przedstawia schemat działania przykładowego neuronu. Posiada on cztery wejścia o danych wagach, których iloczyn jest następnie sumowany. Na podstawie funkcji aktywacyjnej względem sumatora, ustalana jest wartość na wyjściu neuronu.

II. Założenia realizacyjne

Założenia dodatkowe:

- reprezentacja wyjścia na wykresie,
- reprezentacja potencjału na membranie na wykresie,
- reprezentacja wejść na wykresie,
- wybór częstości impulsów,
- możliwość zatrzymania obliczeń,
- możliwość wznowienia obliczeń,
- początkowa wartość potencjału na membranie, dzięki której wartość nie spadnie poniżej 0,
- wagi losowane są w przedziale od 0 do wartości potencjału na membranie powiększonej o 10%, dzięki czemu nie było konieczne zastosowanie górnego ograniczenia oraz możliwe jest uzyskanie 1 na wyjściu, nawet w przypadku bardzo dużej wartości potencjału na membranie.

Liczba danych wejściowych jest dowolna. Ich wartości są losowe zgodnie z wybraną częstością występowania wartości 1: rzadko 25%, umiarkowanie 50%, często 75%. Oznacza to prawdopodobieństwo wystąpienia 1 na wejściu. Wagi dla wejść są losowe lub wpisywane ręcznie. Wartości na wejściu oraz ich wagi są przez siebie mnożone. Wagi oznaczają ważność danego sygnału. Następnie funkcja aktywacji sprawdza wartość sumy iloczynów względem potencjału na membranie, który początkowo wynosi 0.75f. Jeśli suma jest mniejsza niż potencjał, potencjał rośnie. Zwiększamy czas od występowania ostatniej 1 o 0.1f i na wyjściu otrzymujemy 0. Jeśli suma jest większa od potencjału, potencjał jest ustawiany na 0.75f, czas od występowania ostatniej 1 zerujemy, a na wyjściu otrzymujemy wartość 1.

Metody, strategie oraz algorytmy:

Dane wejściowe:
Na wejściu znajdują się losowe ciągi. Użytkownik ustala ich długość. Składają się one z wartości 0 i 1. Dla każdej wartości *in* ciągu generowane są wagi w_i w przedziale od 0 do 110% wartości potencjału U na membranie. Użytkownik może sam wprowadzić wartości wag w_i. Następnie oblicza się iloczyn wejść x_i i wag w_i oraz sumę $\Sigma(x_i*w_i)$ uzyskanych wartości .

Dane wyjściowe:
Funkcja aktywacji polega na obliczaniu potencjału U na membranie w zależności od wag w_i i czasu T od wystąpienia ostatniej 1 na wyjściu *out*. Jeśli suma iloczynów wag i wejść $\Sigma(x_i*w_i)$ jest większa od potencjału początkowego U wynoszącego 0.75f, potencjał jest ustawiany na wartość początkową tak jak czas T od wystąpienia ostatniej 1 na wyjściu *out*, czyli 1. W przeciwnym wypadku potencjał U jest zwiększany o iloraz sumy $\Sigma(x_i*w_i)$ i czasu T od wystąpienia ostatniej 1 na wyjściu, a sam czas zwiększamy o 0.1f. Jeśli wartości zmieniły się na początkowe, wyjście *out* przyjmuje 1, w przeciwnym wypadku 0.

Dane wyjściowe neuronu w sieci neuronowej są wejściami dla neuronu kolejnej warstwy.

Algorytm:

1. Wybierz częstość impulsów *f*.
2. Wybierz liczbę wejść *n*.
3. Losuj wejścia x_i.
4. Losuj wagi w_i lub wpisz je ręcznie.
5. Wygeneruj wykres wejść *in(t)*, gdzie *in* to $x_i * w_i$, a *t* oznacza czas.
6. Oblicz sumę iloczynów wejść i wag $\Sigma(x_i * w_i)$.
7. Oblicz zmianę potencjału ΔU na membranie na podstawie wartości sumy iloczynu wejść i wag $\Sigma(x_i * w_i)$ i czasu *T* od uzyskania ostatniej 1 na wyjściu *out*:
    a. jeśli suma $\Sigma(x_i * w_i)$ jest większa od potencjału początkowego *U*, potencjał zmienia się na wartość początkową, czyli 0.75f,
    b. w przeciwnym wypadku zwiększ potencjał *U* o iloraz $\Sigma(x_i * w_i)$ i czasu *T* od ostatniego pojawienia się 1 na wyjściu *out*.
8. Ustawienie czasu *T* od ostatniego pojawienia się 1 na wyjściu *out*:
    a. jeśli potencjał *U* wzrósł, to czas *T* wzrasta o 0.1f,
    b. w przeciwnym wypadku czas *T* zmienia się na wartość początkową, czyli 1.
9. Oblicz wartość na wyjściu *out*:
    a. jeśli potencjał *U* i czas *T* wzrosły, to na wyjściu *out* jest 0,
    b. w przeciwnym wypadku jest 1.
10. Wygeneruj wykres wyjść *out(t)*.
11. Wygeneruj wykres potencjału na membranie *U(t)*.

Język programowania: *C#.*
Środowisko programistyczne: *Visual Studio 2017.*

III. Podział prac

Autor	**Podzadanie**
Małgorzata Sierbin	Analiza działania pulsującej sieci neuronowej Interfejs graficzny Stworzenie symulacji sieci neuronowej Wizualizacja zmiennych na wykresach Dokumentacja
Robert Greliński	Analiza działania pulsującej sieci neuronowej Opracowanie algorytmu Stworzenie klasy symulującej perceptron wraz z metodami Stworzenie symulacji sieci neuronowej Testowanie działania programu Dokumentacja

IV. Opis implementacji

1. Struktury danych:
`class SpikingPerceptron` – klasa symulująca działanie perceptronu
`private bool[] input` – tablica wejść
`private float[] weight` – tablica wag
`private bool output` – wyjście
`private float potentialOnTheMembrane` – potencjał na membranie
`private float timeOfLastTrue` – czas od wystąpienia ostatniej 1 na wyjściu
`private Random rand` – zmienna pomocnicza do losowania wartości wag
`public bool[] Input {}` – właściwość tablicy wartości na wejściu
`public float[] Weight {}` – właściwość tablicy wag dla wejść
`public bool Output {}` – właściwość wartości na wyjściu
`public float PotentialOnTheMembrane {}` – właściwość wartości potencjału na membranie
`public float TimeOfLastTrue {}` – właściwość czasu od wystąpienia ostatniej1 na wyjściu

`public partial class Form1 : Form` – klasa zawierająca program, wywołuje metody klasy *SpikingPerceptron*
`Random rand` – obiekt klasy *Random* służący do losowego wyznaczenia wartości wejścia na podstawie częstości impulsów oraz wartości wag
`SpikingPerceptron perceptron` – obiekt klasy *SpikingPerceptron*
`bool[] input` – tablica wejść
`float[] weight` – tablica wag
`int licznik_wag` – określa którą z kolejnych wag ustawia użytkownik, jeśli zdecydował się wpisywać ręcznie
`int licznik_cykli` – zlicza liczbę cykli
`Thread wizualizacja` – wątek w którym działa dokonywanie obliczeń
`bool czy_losowac` – 1 oznacza, że użytkownik chce losować wartości wag, w przeciwnym wypadku chce je wpisać ręcznie
`private bool start` – oznacza czy wątek działa, czy nie, czyli czy obliczanie jest wykonywane, czy wstrzymane
`bool stworz_series` – określa czy stworzono serię na wykresie
`List<int> tablica` – tablica reprezentująca budowę sieci neuronowej
`List<SpikingPerceptron[]> siec` – struktura sieci neuronowej
`TextBox[] textboxes` – tablica elementów typu *TextBox*, w sieci neuronowej. Służy do wyświetlania wyjścia poszczególnych neuronów.
`PictureBox[] boxes2` – tablica elementów typu *PictureBox*, w sieci neuronowej. Służy do wyświetlania grafiki poszczególnych neuronów.

2. Metody i funkcje:

`public SpikingPerceptron(){}` – ustawia rozmiar tablicy wejść na 2

`public SpikingPerceptron(int NOinput){}` – ustawia rozmiar tablic wejść i wag na *NOinput*

`public void Start(){}` – wywołuje funkcję *Activate*

`private float sum(){}` – oblicza sumę iloczynów wejść i wag i ją zwraca w typie *float*

`private bool Activate(){}` – ustala wartość na wyjściu na podstawie sumy i potencjału na membranie oraz ją zwraca w typie *bool*

`public void weightGen(){}` – generuje losowe wartości wag

`public Form1(){}` – wywołuje Initializecomponent, czyli zajmuje się scalaniem aplikacji z kodem

`private void wizualizuj(){}` – wypisuje informacje dotyczące wejść i wyjść, losuje wejścia, wywołuje funkcje tworzące wykresy, wywołuje losowanie wag jeśli to konieczne, wizualizuje sieć neuronową

`public void AppendrichTextBox1(string value){}` – służy do wypisywania wartości wag dla wejść, w argumencie przyjmuje daną wartość wagi typu *string*

`public void SetrichTextBox2(string value){}` – służy do wypisywania wartości będącej argumentem typu *string* na wejściu i wyjściu

`public void AddPointToChart_wyjscia(bool Y){}` – funkcja służy do wizualizacji wartości na wyjściu na wykresie, w argumencie przyjmuje daną wartość wyjścia typu *bool*

`public void AddPointToChart_membrana(float Y){}` – funkcja służy do wizualizacji potencjału na membranie na wykresie, w argumencie przyjmuje daną wartość membranie typu *float*

`public void AddPointToChart_wejscia(bool Y)` – funkcja służy do wizualizacji wartości na wejściu na wykresie, w argumencie przyjmuje daną wartość wejścia typu *bool*

`private void button4_Click(object sender, EventArgs e){}` – tworzy obiekt *SpikingPerceptron* o danych wejściach i wagach, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

`private void button1_Click_1(object sender, EventArgs e){}` – przycisk będący opcją start i stop wykonywania obliczeń, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

`private void button2_Click(object sender, EventArgs e){}` – wciśnięty przycisk oznacza, że wartości wag dla wejść mają być losowe, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

`private void losuj(){}` – funkcja służy do losowania wartości wag

`private void button3_Click(object sender, EventArgs e){}` – służy do ręcznego ustawiania wartości wag, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

`private void Form1_FormClosing(object sender, FormClosingEventArgs e){}` – gdy użytkownik zamyka aplikację, pojawia się komunikat, który należy potwierdzić, aby wyłączyć, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

`private void button7_Click(object sender, EventArgs e)` – wywołuje przycisk *button1*, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

`public void AddPointToChart_Wyjście(bool Y)` – tworzy wykres wyjść sieci neuronowej,w argumencie przyjmuje daną wartość wyjścia typu *bool*

`private void wejściaToolStripMenuItem_Click(object sender, EventArgs e)` – służy do manewru między panelami aplikacji, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane
`private void neuronToolStripMenuItem_Click(object sender, EventArgs e)` – służy do manewru między panelami aplikacji, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane
`private void siećNeuronowaToolStripMenuItem_Click(object sender, EventArgs e)` – służy do manewru między panelami aplikacji, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane
`public void Appendtextbox(string value, int index)` – metoda służąca do synchronicznego wypisywania w *textboxes* wyjścia poszczególnych neuronów, *value* oznacza wartość wyjścia, a *index* to identyfikator neuronu
`private bool start` – flaga startu (czy program się już rozpoczął), zwraca 0 jeśli nie oraz 1 jeśli tak
`private void button5_Click(object sender, EventArgs e)` – ustawia liczbę warstw sieci neuronowej, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane
`private void button6_Click(object sender, EventArgs e)` - symulacja grafiki sieci neuronowej, argument *sender* przekazuje informację o tym jaki obiekt jest nadawcą funkcji, argument *e* przekazuje zdarzenie, które jest obsługiwane

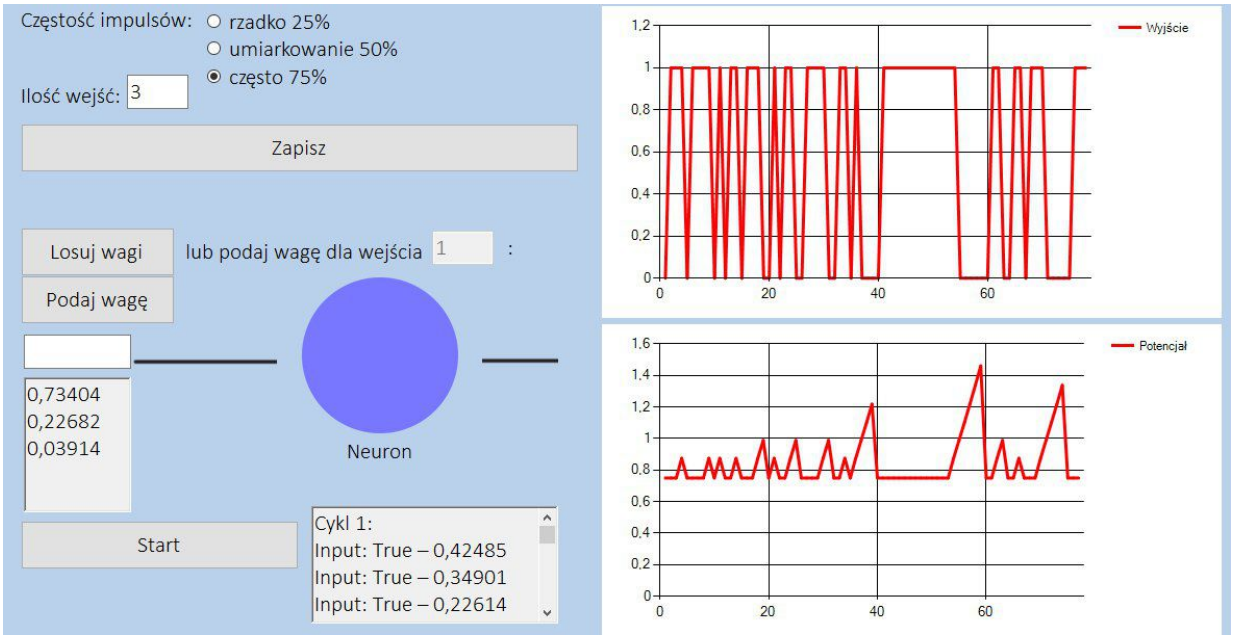
V. Użytkowanie i testowanie systemu

Po uruchomieniu programu otwiera się okno z 3 zakładkami: *Neuron*, *Wyjścia* oraz *Sieć neuronowa*.

W pierwszej zakładce istnieje możliwość symulacji działania pojedynczego neuronu. Użytkownik może wybrać częstość występowania impulsów spośród: 25% - rzadko, 50% - umiarkowanie oraz 75% - często. Oznacza to, jak często na wejściu występuje wartość 1. Ponadto można wybrać liczbę wejść do neuronu oraz zadecydować czy wagi dla wejść mają być wyznaczane losowo, czy ręcznie. Wejścia są postaci binarnej, a ich wagi w przedziale od 0 do 1. Następnie użytkownik może rozpocząć działanie neuronu oraz w dowolnym momencie go zatrzymać i ponownie wznowić. Dodatkowo dostępne są dwa wykresy przedstawiające wartości na wyjściu oraz potencjał na membranie.

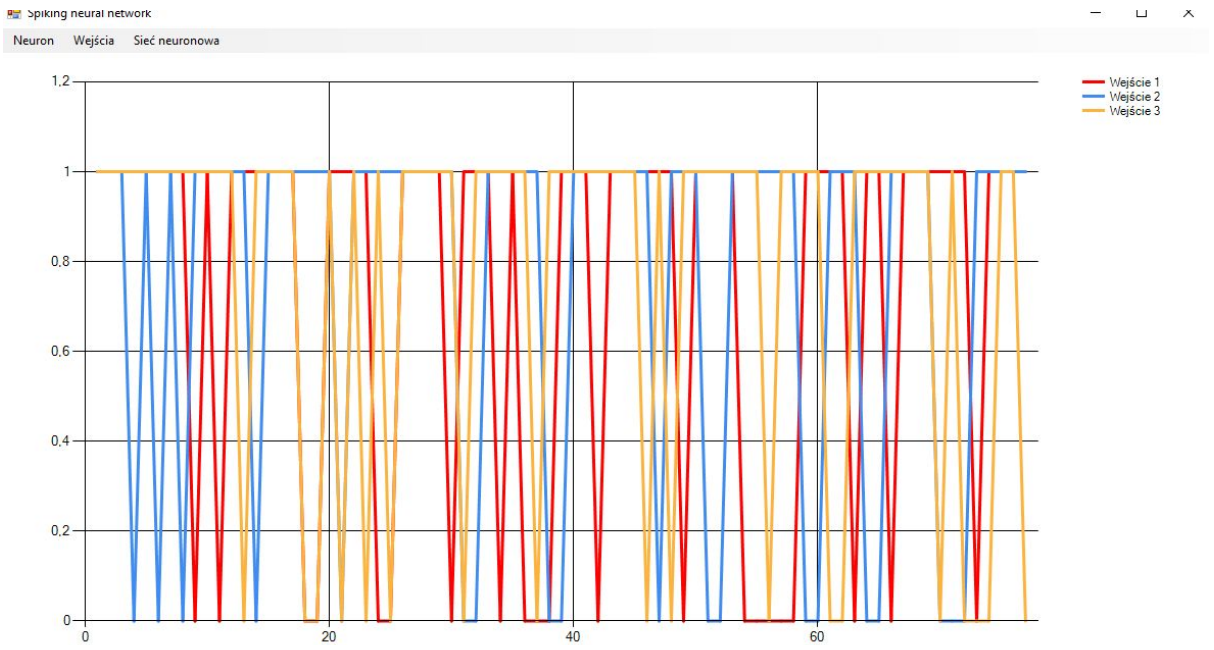
W drugiej zakładce znajduje się wykres przedstawiający wartości na wejściu.

W ostatniej zakładce możemy symulować działanie sieci neuronowej poprzez podanie jej parametrów, czyli liczba warstw oraz kolejno liczba neuronów w każdej z nich. Dostępny jest wykres wartości na wyjściu.



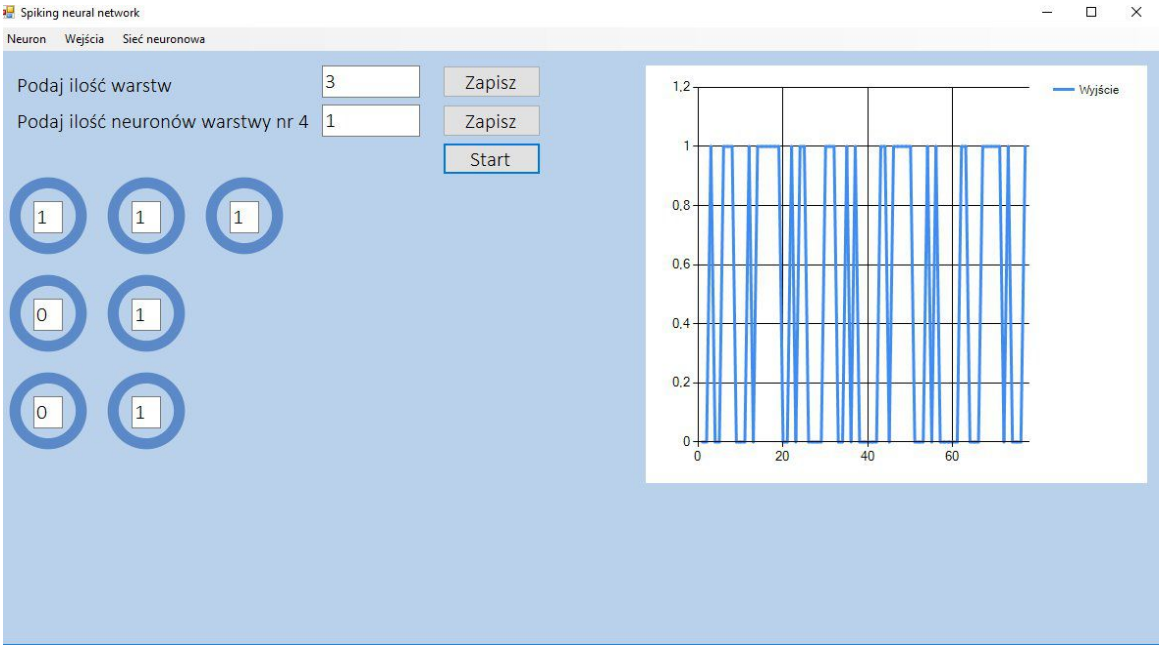
Rys. 2. Screen ekranu neuronu podczas działania aplikacji

Po uruchomieniu aplikacji dostępny jest widok zakładki *neuron*. Z prawej strony znajdują się dwa wykresy: górny przedstawia wartości na wyjściu, a dolny potencjały na membranie. Z lewej strony na górze znajdują się ustawienia dotyczące wyboru częstości impulsów pomiędzy 25%, 50% oraz 75%. Oznacza to częstość występowania 1 na wejściu. Poniżej znajduje się pole do wpisania żądanej liczby wejść. Następnie należy zapisać ustawienia. Kolejnym krokiem jest decyzja czy chcemy aby wagi dla wejść były ustawiane losowo, czy też sami je wpiszemy. W pierwszej sytuacji wciskamy przycisk *"Losuj wagi"*, a w drugiej wpisujemy wagę i klikamy *"Podaj wagę"*, W polu poniżej wypisane zostaną ustawione wartości wag dla wejść. Przycisk *"Start"* służy do rozpoczęcia działania programu, czyli rozpoczyna wykonywanie obliczeń. W polu obok przycisku *"Start"* znajduje się pole w którym można przejrzeć wszystkie cykli i odpowiednie dla nich wartości wejścia, wagi dla wejść, wyjścia i potencjały na membranie.



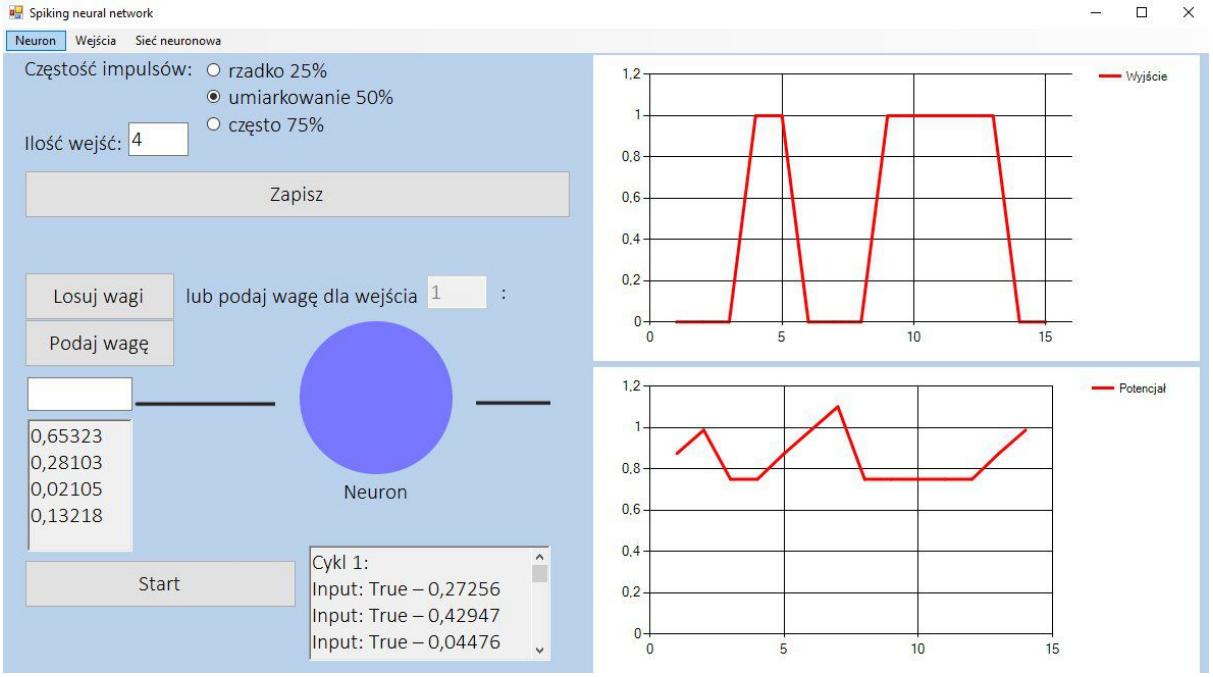
Rys. 3. Screen ekranu wejść

Zakładka o nazwie *"Wejścia"* przedstawia wykres wartości na wejściu, na którym widać jak stany wejść zmieniały się w czasie. Wejścia są umieszczane na jednym wykresie, a kolory są generowane automatycznie. Stan wysoki oznacza, że na wejściu pojawiła się 1, w przeciwnym wypadku 0.



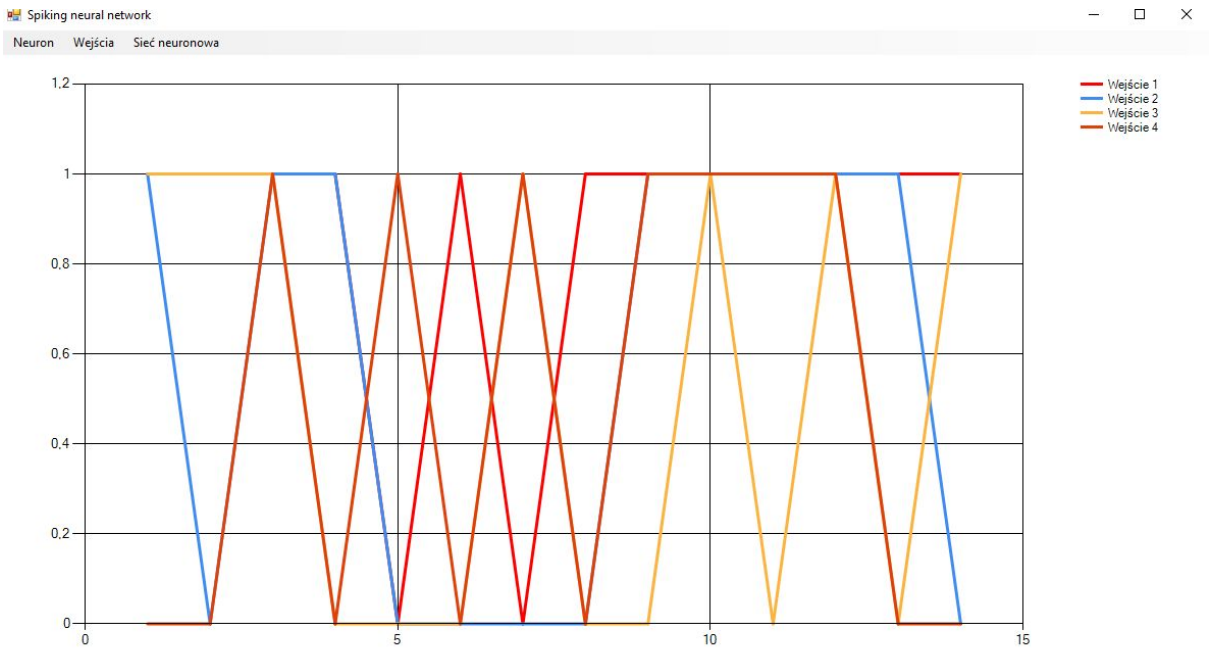
Rys. 4. Screen ekranu sieci neuronowej podczas działania aplikacji

Ostatnia zakładka to *"Sieć neuronowa"*. Ekran sieci neuronowej to prosty kreator sieci neuronowej do symulacji. Należy podać liczbę warstw sieci, a następnie liczbę neuronów w poszczególnych warstwach. Przycisk *start* uruchamia symulację. Po prawej stronie znajduje się wykres wyjścia sieci neuronowej. Neurony wygenerowanej sieci są reprezentowane przez okręgi, w środku których znajduje się pole tekstowe, gdzie wyświetlane jest aktualnie wyjście neuronu.



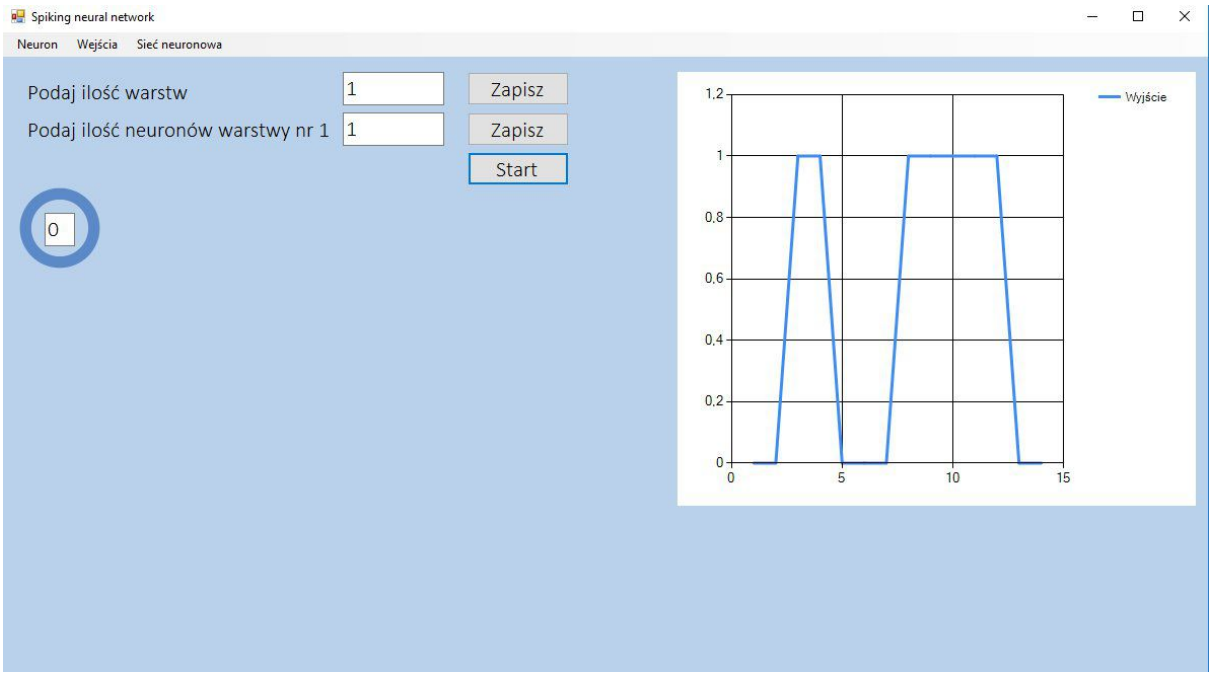
Rys. 5. Screen ekranu neuronu podczas działania aplikacji, gdy sieć składa się z jednego neuronu

Rysunek przedstawia ustawienia dotyczące neuronu dla sieci składającej się z jednego neuronu. Ustawiono częstość impulsów na 50% oraz liczbę wejść w wysokości 4. Wagi zostały wygenerowane losowo. Z prawej strony widoczne są wykresy wyjścia i potencjału.



Rys. 6. Screen ekranu wejść podczas działania aplikacji, gdy sieć składa się z jednego neuronu

Na rysunku widać wykres wartości kolejnych wejść dla sieci składającej się z jednego neuronu względem czasu. Kolory są generowane losowo. Stan wysoki oznacza wartość 1, a niski 0.



Rys. 7. Screen ekranu sieci neuronowej, gdy sieć składa się z jednego neuronu

Przykładowa sieć składa się z jednego neuronu. Na ekranie sieci neuronowej wyświetlane jest wyjście sieci neuronowej, które jest takie samo jak wyjście pojedynczego neuronu. Sieć składająca się z pojedynczego neuronu jest uważana za najprostszą.

VI. Tekst programu

SpikingPerceptron.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Testy
{
    class SpikingPerceptron
    {
        private bool[] input;
        private float[] weight;
        private bool output;
        private float potentialOnTheMembrane = 0.75f;
        private float timeOfLastTrue = 1;
        private Random rand = new
Random((int)DateTime.Now.Ticks);

        public bool[] Input { get => input; set => input = value;
}
        public float[] Weight { get => weight; set => weight =
value; }
        public bool Output { get => output; set => output =
value; }
        public float PotentialOnTheMembrane { get =>
potentialOnTheMembrane; set => potentialOnTheMembrane = value; }
        public float TimeOfLastTrue { get => timeOfLastTrue; set
=> timeOfLastTrue = value; }

        public SpikingPerceptron()
        {
            input = new bool[2];
        }
        public SpikingPerceptron(int NOinput)
        {
            input = new bool[NOinput];
            Weight = new float[NOinput];
        }
        public void Start()
        {
            Output = Activate();
        }
        private float sum()
        {
            float s = 0;
            for (int i = 0; i < input.Length; i++) s +=
(Convert.ToInt32(input[i]) * weight[i]);
```

```
            return s;
        }
        private bool Activate()
        {
            if (sum() < PotentialOnTheMembrane)
            {
                for (int i = 0; i < weight.Length; i++)
                {
                    PotentialOnTheMembrane += (weight[i] /
timeOfLastTrue) / 8;
                }
                TimeOfLastTrue += 0.1f;

                return false;
            }
            else
            {
                PotentialOnTheMembrane = 0.75f;
                TimeOfLastTrue = 1;
                return true;
            }
        }
        public void weightGen()
        {
            weight = new float[input.Length];
            for (int i = 0; i < weight.Length; i++)
            {
                    int ran = rand.Next(1,
(int)(potentialOnTheMembrane*1.1* 10000));
                    weight[i] = (float)(ran) / 10000f;
            }
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Text.RegularExpressions;
using System.Threading;
```

```
using System.Windows.Forms.DataVisualization.Charting;
using System.Collections;

namespace Testy
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        Random rand = new Random();
        SpikingPerceptron perceptron;
        bool[] input;
        float[] weight;
        int licznik_wag = 0;
        int licznik_cykli = 0;
        Thread wizualizacja = null;
        bool czy_losowac = true;
        bool stworz_series = false;
        List<int> tablica = new List<int>();
        List<SpikingPerceptron[]> siec = new
List<SpikingPerceptron[]>();
        TextBox[] textboxes = new TextBox[1000];

        private void wizualizuj()
        {
            while(true)
            {
                string append = "";
                licznik_cykli++;
                append += string.Format("Cykl {0}:\n",
licznik_cykli);
                int freq = 1;

                bool[] table1 = { false, false, false, true };
                bool[] table2 = { false, false, true, true };
                bool[] table3 = { false, true, true, true };

                if (radioButton1.Checked == true) freq = 0;
                if (radioButton2.Checked == true) freq = 1;
                if (radioButton3.Checked == true) freq = 2;

                for (int j = 0; j < input.Length; j++)
                {
                    switch (freq)
                    {
                        case 0:
                            input[j] =
```

```
Convert.ToBoolean(table1[rand.Next(0, 4)]);
                            break;
                        case 1:
                            input[j] =
Convert.ToBoolean(table2[rand.Next(0, 4)]);
                            break;
                        case 2:
                            input[j] =
Convert.ToBoolean(table3[rand.Next(0, 4)]);
                            break;
                    }
                }
                perceptron.Input = input;
                perceptron.Weight = weight;
                perceptron.Start();
                if (stworz_series == false)
                {
                    for (int j = 2; j < input.Length+1; j++)
                    {
                        chart3.Series.Add("Wejście " + j);
                        chart3.Series["Wejście " + j].ChartType =
SeriesChartType.Line;
                        chart3.Series["Wejście " + j].BorderWidth
= 3;
                    }
                    stworz_series = true;
                }
                for (int j = 0; j < input.Length; j++)
                {
                    append += string.Format("Input: {0} - {1}\n",
perceptron.Input[j], perceptron.Weight[j]);
                    AddPointToChart_wejscia(input[j]);
                }
                append += string.Format("Output: {0} \n",
perceptron.Output);
                append += string.Format("Potencjał na membranie:
{0} \n", perceptron.PotentialOnTheMembrane);
                append += "=================\n";
                AppendrichTextBox1(append);
                AddPointToChart_wyjscia(perceptron.Output);

AddPointToChart_membrana(perceptron.PotentialOnTheMembrane);
                //siec
                for (int i = 0; i < tablica[0]; i++)
                {
                    siec.ElementAt(0)[i].Input = input;
                    if(i!=0)
                    {
                        siec.ElementAt(0)[i].weightGen();
                    }
```

```
                    else siec.ElementAt(0)[i].Weight = weight;
                }
                 for (int i = 1; i < Int32.Parse(textBox2.Text);
i++)
                 {
                     for (int j = 0; j < tablica[i]; j++)
                     {
                         bool[] oldinput = new bool[tablica[i -
1]];
                         for (int z = 0; z < tablica[i - 1]; z++)
                         {
                             oldinput[z] = siec[i -
1].ElementAt(z).Output;
                         }
                         siec[i].ElementAt(j).Input = oldinput;
                        siec[i].ElementAt(j).weightGen();
                     }
                 }
                int txt = 0;
                foreach (var warstwa in siec)
                {
                    foreach(var neuron in warstwa)
                    {
                        neuron.Start();
                        Appendtextbox(neuron.Output ? "1" : "0",
txt);
                        txt++;
                    }
                }

AddPointToChart_Wyjście(siec.ElementAt(siec.Count-1)[0].Output);
                if (czy_losowac)losuj();
                Thread.Sleep(1000);
            }
        }

        public void Appendtextbox(string value, int index)
        {
            if (InvokeRequired)
            {
                this.Invoke(new
Action<string,int>(Appendtextbox), new object[] { value, index
});
                return;
            }
            textboxes[index].Text = value;
        }
        public void AppendrichTextBox1(string value)
        {
            if (InvokeRequired)
```

```
            {
                this.Invoke(new
Action<string>(AppendrichTextBox1), new object[] { value });
                return;
            }
            richTextBox1.Text += value;
        }
        public void SetrichTextBox2(string value)
        {
            if (InvokeRequired)
            {
                this.Invoke(new Action<string>(SetrichTextBox2),
new object[] { value });
                return;
            }
            richTextBox2.Text = value;
        }

        public void AddPointToChart_wyjscia(bool Y)
        {
            if (InvokeRequired)
            {
                this.Invoke(new
Action<bool>(AddPointToChart_wyjscia), new object[] { Y });
                return;
            }
            chart1.Series["Wyjście"].Points.AddY(Y);
        }
        public void AddPointToChart_membrana(float Y)
        {
            if (InvokeRequired)
            {
                this.Invoke(new
Action<float>(AddPointToChart_membrana), new object[] { Y });
                return;
            }
            chart2.Series["Potencjał"].Points.AddY(Y);
        }

        int licznik_wejscia=1;

        public void AddPointToChart_wejscia(bool Y)
        {
            if (InvokeRequired)
            {
                this.Invoke(new
Action<bool>(AddPointToChart_wejscia), new object[] { Y });
                return;
            }
            chart3.Series["Wejście
```

```
"+licznik_wejscia].Points.AddY(Y);
            licznik_wejscia++;
            if (licznik_wejscia > input.Length) licznik_wejscia =
1;
        }

        private void button4_Click(object sender, EventArgs e)
        {
            perceptron = new
SpikingPerceptron(Int32.Parse(textBox1.Text));
            input = new bool[Int32.Parse(textBox1.Text)];
            weight = new float[Int32.Parse(textBox1.Text)];
            textBox4.Text = "1";
        }
        private bool start = false;
        private void button1_Click_1(object sender, EventArgs e)
        {
            if(start)
            {
                button1.Text = "Start";
                button7.Text = "Start";
                wizualizacja.Abort();
            }
            else
            {
                button1.Text = "Stop";
                button7.Text = "Stop";
                wizualizacja = new Thread(wizualizuj);
                wizualizacja.Start();
            }
            start = !start;
        }
        private void button2_Click(object sender, EventArgs e)
        {
            czy_losowac = true;
            losuj();
        }
        private void losuj()
        {
            string setText = "";
            int maxi = 100000;
            int current = Math.Max((int)(100000 *
perceptron.PotentialOnTheMembrane * 1.1), 100000);
            for (int i = 0; i < weight.Length; i++)
            {
                if (i + 1 != weight.Length)
                {
                    int ran = rand.Next(1, current);
                    weight[i] = (float)(ran) / (float)maxi;
                    current -= ran;
```

```
                    setText += weight[i].ToString() + "\n";
                }
                else
                {
                    weight[i] = (float)current / (float)maxi;
                    setText += weight[i].ToString();
                }
            }
            SetrichTextBox2(setText);
        }
        private void button3_Click(object sender, EventArgs e)
        {
            czy_losowac = false;
            if (licznik_wag<weight.Length)
            {
                if (textBox5.Text == "") textBox5.Text = "0";
                textBox5.Text = textBox5.Text.Replace('.', ',');

                weight[licznik_wag] =float.Parse(textBox5.Text);
                richTextBox2.Text +=
weight[licznik_wag].ToString() + "\n";
                textBox4.Text = (licznik_wag + 1).ToString();
                licznik_wag++;
            }
            else
            {
                richTextBox2.Text = "";
                licznik_wag = 0;
                button3_Click(sender, e);
            }
        }
        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (MessageBox.Show("Czy na pewno chcesz zakończyć
program?", "Spiking perceptron",
    MessageBoxButtons.YesNo) == DialogResult.Yes)
            {

if(wizualizacja!=null)if(wizualizacja.IsAlive)wizualizacja.Abort(
);
            }
            else
            {
                e.Cancel = true;
            }
        }
        private void wejściaToolStripMenuItem_Click(object
sender, EventArgs e)
        {
```

```
            panel1.Visible = true;
            panel2.Visible = false;
        }
        private void neuronToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            panel1.Visible = false;
            panel2.Visible = false;
        }

        PictureBox[] boxes = new PictureBox[1000];
        int warstwa = 1;
        int ilosc_warstw;

        private void button5_Click(object sender, EventArgs e)
        {
            ilosc_warstw = Int32.Parse(textBox2.Text);
        }

        private void siećNeuronowaToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            panel2.Visible = true;
            panel1.Visible = false;
        }

        PictureBox[] boxes2 = new PictureBox[1000];
        int location1 = 12;
        int location2 = 124;
        int UstawianaWarstwa = 0;
        int poprzedniaIloscneuronow = 0;
        int sumaNeuronow = 0;

        private void button6_Click(object sender, EventArgs e)
        {
            if (warstwa < Int32.Parse(textBox2.Text) + 1)
            {
                int ilosc_neuronow = Int32.Parse(textBox3.Text);
                for(int
i=sumaNeuronow;i<ilosc_neuronow+sumaNeuronow;i++)
                {
                    textboxes[i] = new TextBox();
                    textboxes[i].Location = new
System.Drawing.Point(location1 + 30, location2 + 30);
                    textboxes[i].Margin = new
System.Windows.Forms.Padding(4, 5, 4, 5);
                    textboxes[i].Name = "textBox" + (5 +
Int32.Parse(textBox2.Text) + 1);
                    textboxes[i].Size = new
System.Drawing.Size(30, 33);
```

```
                    panel2.Controls.Add(textboxes[i]);

                    boxes2[i] = new PictureBox();
                    boxes2[i].Image =
global::Testy.Properties.Resources._367725c65f62e46d7d9c4fdf5d1c8
b3d;
                    boxes2[i].Location = new
System.Drawing.Point(location1, location2);
                    boxes2[i].Name = "pictureBox" + (3 +
Int32.Parse(textBox2.Text) + 1);
                    boxes2[i].Size = new System.Drawing.Size(89,
88);
                    boxes2[i].Margin = new
System.Windows.Forms.Padding(4, 5, 4, 5);
                    boxes2[i].TabStop = false;
                    panel2.Controls.Add(boxes2[i]);

                    location2 += 100;
                }
                tablica.Add(Int32.Parse(textBox3.Text));

                if(UstawianaWarstwa==0)
                {
                    siec.Add(new SpikingPerceptron[tablica[0]]);

                    for (int i = 0; i < tablica[0]; i++)
                    {
                        siec.ElementAt(0)[i] = new
SpikingPerceptron(tablica[0]);
                    }
                }
                else
                {
                    siec.Add(new
SpikingPerceptron[tablica[UstawianaWarstwa]]);

                    for (int i = 0; i <
tablica[UstawianaWarstwa]; i++)
                    {
                        siec.ElementAt(UstawianaWarstwa)[i] = new
SpikingPerceptron(tablica[UstawianaWarstwa - 1]);
                    }
                }
                poprzedniaIloscneuronow = ilosc_neuronow;
                UstawianaWarstwa++;
                sumaNeuronow += ilosc_neuronow;
                if (ilosc_warstw-1 < UstawianaWarstwa)
                {
                    UstawianaWarstwa = 0;
                }
```

```
                warstwa++;
                location2 = 124;
                location1 += 100;
                label3.Text = "Podaj ilość neuronów warstwy nr "
+ (UstawianaWarstwa + 1).ToString();
            }
        }

        private void button7_Click(object sender, EventArgs e)
        {
            button1_Click_1(sender, e);
        }

        public void AddPointToChart_Wyjście(bool Y)
        {
            if (InvokeRequired)
            {
                this.Invoke(new
Action<bool>(AddPointToChart_Wyjście), new object[] { Y });
                return;
            }
            chart4.Series["Wyjście"].Points.AddY(Y);
        }
    }
}
```