Systemy Sztucznej Inteligencji

dokumentacja projektu 'Przewidywanie pogody'

Adam Czerwiński, Szymon Joszko, Szymon Kędziora, grupa 2C

6 czerwca 2019

Część I

Opis programu

Nasz program służy do przewidywania pogody dla podanych danych. W programie została wykorzystana sieć neuronowa uczona algorytmem wstecznej propagacji. Wyniki obserwacji pogodowych potrzebne do nauki sieci, a także do testowania zostały dostarczone przez Instytut Meteorologii i Gospodarki Wodnej (`https://dane.imgw.pl/data/dane_pomiarowo_obserwacyjne/dane_meteorologiczne/terminowe/klimat/`), pochodziły one z pięciu miast: Radziechów, Poronina, Skierniewic, Białowieży i Borucina. Każde z tych miast reprezentuje dany region Polski (północ, południe, wschód, zachód, centrum).
Dane potrzebne do przewidywania pogody:

- region
- data
- godzina
- temperatura
- wilgotność
- kierunek wiatru
- prędkość wiatru
- zachmurzenie

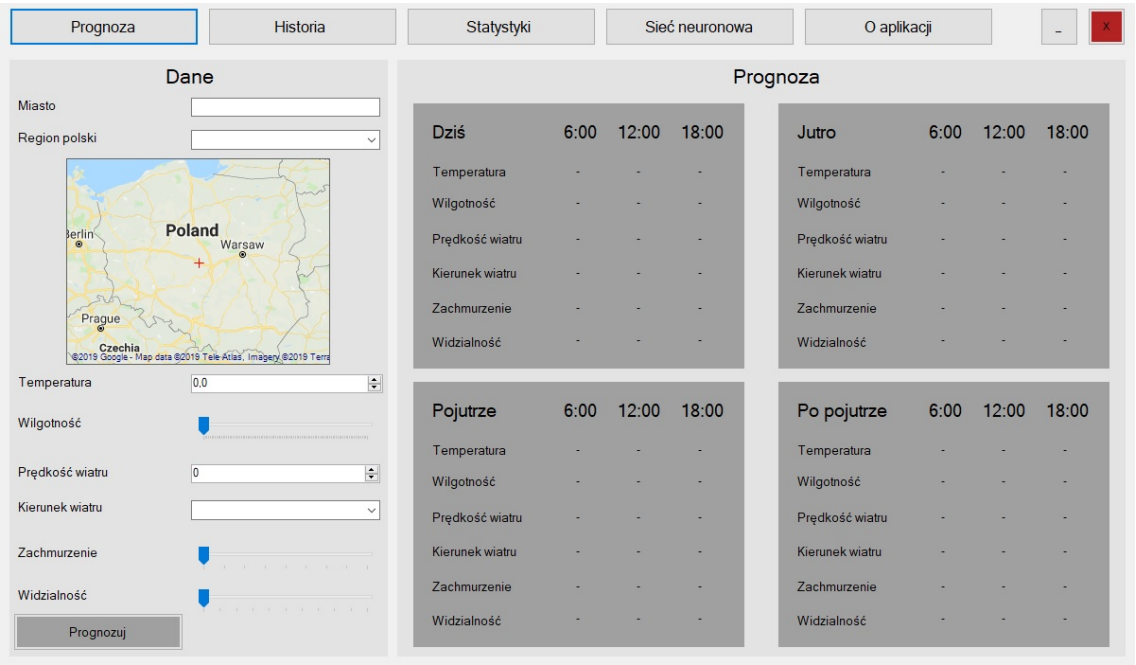
Wykorzystane przez nas technologie:

- C# .NET Framework 4.6.1
- WinForms MVP pattern
- MySQL
- Connector/NET 8.0.15

Użytym przez nas środowiskiem programistycznym jest Microsoft Visual Studio 2017.

Instrukcja obsługi

Po uruchomieniu aplikacji użytkownikowi pokazuje się menu główne.



Rysunek 1: Menu Główne

W panelu po lewej użytkownik uzupełnia obecne dane pogodowe. Większość danych wprowadzana jest za pomocą listy wysuwanej lub suwaków, wpisywana jest tylko nazwa miasta. Po wprowadzeniu danych konsument musi nacisnąć przycisk "Prognozuj", by poznać prognozę na najbliższe 3 dni.

Dodatkowe informacje

Do nauki użyliśmy danych z lat 2015-2017. Natomiast do testów użyliśmy wyników obserwacji z roku 2018 i początku roku 2019.

Część II

Opis działania

Sieć neuronowa składa się z :

- Warstwy wejściowej wykorzystującej X neuronów
- Jednej warstwy ukrytej wykorzystującej Y neuronów
- Warstwy wyjściowej wykorzystującej X neuronów

Wykorzystaną przez nas funkcją aktywacji jest tangens hiperboliczny.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1}$$

Korzystamy również z pochodnej tej funkcji.

$$f'(x) = 1 - f(x)^2 \tag{2}$$

Do ustalenia które ustawienie początkowe sieci jest najlepsze wykorzystujemy błąd średniokwadratowy (MSE).

$$E = \frac{1}{K} \sum_{k=0}^{K} \quad (y_k - d_k)^2 \tag{3}$$

Gdzie:

- y - wartość zwrócona przez sieć nuronową
- d - wartość oczekiwana
- K - ilość neuronów w warstwie wyjściowej

Do normalizacji większości danych użyliśmy metody min-max.

$$y(x) = \frac{x - min}{max - min} \tag{4}$$

Gdzie:

- min - najmniejsza wartość dla jednej z danych np. zachmurzenia
- max - największa wartość dla jednej z danych

Dla daty musieliśmy zmodyfikować powyższą metodę.

$$y(x) = \frac{m * i + d - min}{max - min} + a \tag{5}$$

Gdzie:

- min - najmniejsza wartość, w naszym przypadku 22(chcieliśmy, by zima miała najmniejsze wartości, więc zaczynamy od 22 grudnia

- max - największa wartość, w naszym przypadku 387 lub 388(jest to ilość dni w roku + 22)
- m - numer miesiąca(od 22 grudnia do końca roku ta wartość wynosi 0)
- i - ilość dni w danym miesiącu
- d - numer dnia miesiąca
- a - dodatkowy parametr, chcieliśmy by każda pora roku zaczynała się mniej więcej w kolejnych ćwiartkach. Niestety niekiedy otrzymywane wartości były trochę zbyt niskie więc sztucznie je podwyższyliśmy.

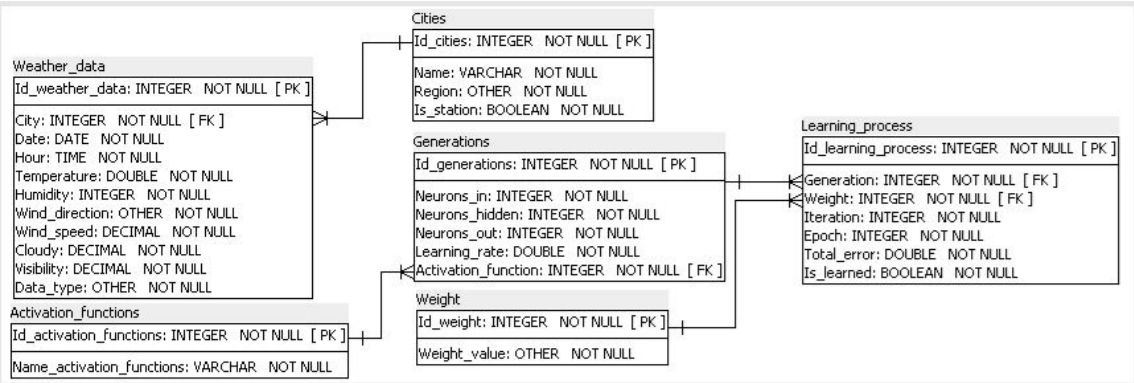
Algorytm

Data: znormalizowane wartości z bazy danych, sieć o najlepszych wagach początkowych
Result: Nowe wagi obliczone algorytmem wstecznej propagacji
for *dla zadanej ilości iteracji* **do**
  for *dla wszystkich danych do nauki* **do**
    //FeedForward()
    for *dla wszystkich neuronów wejściowych* **do**
      Obliczenie wartości neuronu po aktywacji
    end
    //BackProp()
    for *Dla wszystkich neuronów w warstwie* **do**
      Obliczenie nowych wartości wag
    end
  end
end

Bazy danych

Korzystamy z bazy danych MySQL uruchomionej na serwerze lokalnym. Składa się ona z pięciu tabel:

- Weather_Data - przechowuje ona dane pogodowe
- Cities - znajdują się tu dane o miastach z których mamy dane
- Activation_functions - przechowuje nazwy funkcji aktywacji
- Generations - posiada informacje o ilości neuronów, wskaźniku uczenia i funkcji aktywacji dla kolejnych generacji sieci
- Weight - znajdują się tu wartości wag
- Learning_process - mamy tu cały proces uczenia, dane o generacji, wagach, ilości iteracji, wielkości błędu i epoce

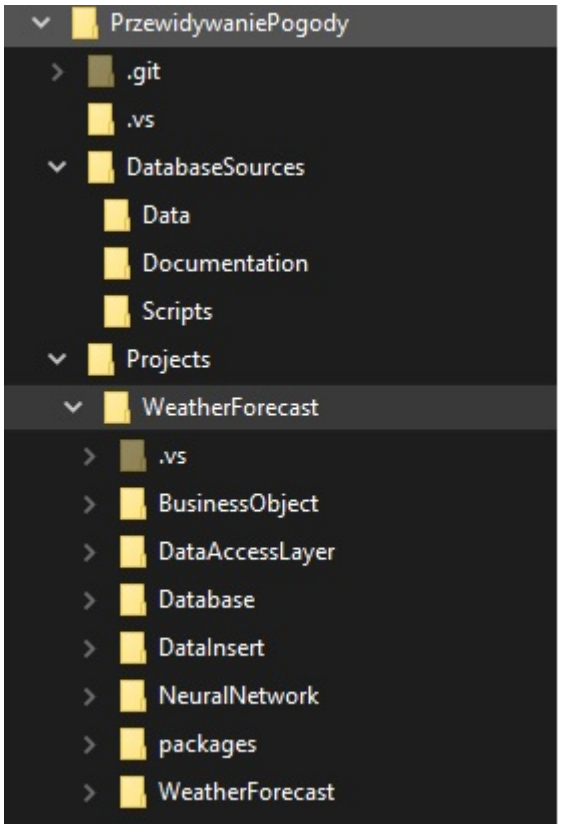


Rysunek 2: Schemat bazy

Lokacja	Rok	Mies	Dzień	Godz	Temp	Wilgoć	Kierunek	Pręd	Zach	Wid
RADZIECHOWY	2016	1	31	18	2.2	73	SSE	4	7	6
PORONIN	2016	1	1	6	-15.0	78	E	1	1	9
BIAŁOWIEŻA	2017	7	4	6	16.2	88	SW	3	6	7

Tablica 1: Przykładowe dane pogodowe

Implementacja



Rysunek 3: Struktura

W katalogu DatabaseSources/Scripts znajduje się 5 plików.

- ssi_database_create.sql - skrypt tworzący pustą bazę danych
- load_database.bat - skrypt tworzący pustą bazę danych
- ssi_database_with_data.sql - wypełnienie tabel Cities oraz Weather_Data danymi
- load_database_with_data.bat - wypełnienie tabel Cities oraz Weather_Data danymi
- mysql.exe - uruchamia powyższe pliki wsadowe

Oraz odpowiednio dwa pliki dla programu webserv:

- load_database.bat
- load_database_with_data.bat

W katalogu DataBaseSources/Data znajduje się również 5 plików.

- bestWeights - wagi najlepszej sieci wybranej przez funkcję FindBestStartupSettings()
- LearningData - dane pogodowe przeznaczone do uczenia sieci
- NeuralNetworkSettings - ilość warstw i neuronów w tych warstwach
- NeuralNetworkWeights - wagi nauczonej sieci
- TestingData - dane pogodowe przeznaczone do testowania sieci

Projekty

- BusinessObject - modele w naszym rozwiązaniu (solution).
- DataAccessLayer - repozytoria dla naszych modeli
- Database - połączenie z bazą danych
- DataInsert - program wrzucający dane pogodowe do bazy danych
- NeuralNetwork - program do uczenia sieci neuronowej
- WeatherForecast - program przewidujący pogodę

Funkcje

- BusinessObject
  - ToString() - przeciążona metoda ToString() służy nam do wypisania danych wraz z podpisami
- DataAccessLayer
  - Save() - łączy się z bazą danych, następnie zapisuje w niej dane z listy podanej jako argument

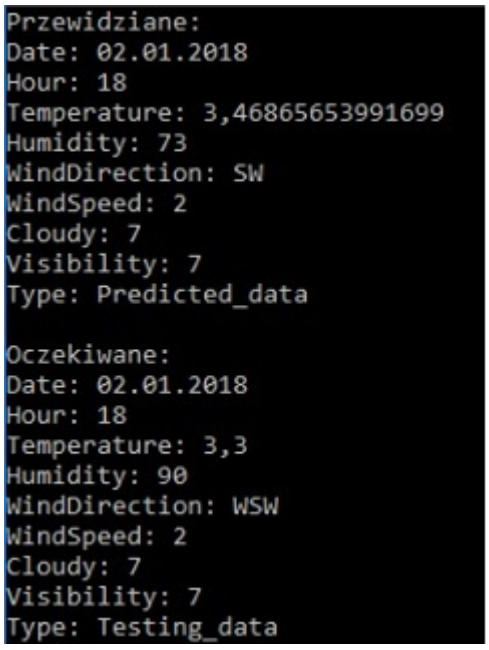
  - AddGeneration() - dodaje konfigurację sieci do bazy
  - AddLearningProcess() - funkcja dodająca proces uczenia do bazy
  - AddWeight() - dodaje wartości wag do bazy
- Database
  - DBConnection() - zapewnia połączenie z naszą bazą danych
  - GetLastIndex() - zwraca ostatni indeks występujący w podanej tabeli
- DataInsert
  - LoadData() - wczytuje dane do nauki i testowania sieci neuronowej z pliku LearningAndTestingData.txt
- NeuralNetwork
  - Calculate() - obliczanie wartości wag po użyciu funkcji aktywacji
  - CalculateDerivative() - aktywacja neuronów funkcją aktywacji
  - InitializeWeights() - przypisanie wagom losowych wartości
  - FeedForward() - obliczanie wartości neuronów w danej warstwie
  - BackPropOutput() - obliczanie wag warstwy wyjściowej przy pomocy metody wstecznej propagacji
  - BackPropHidden() - obliczanie wag warstw ukrytych przy pomocy metody wstecznej propagacji
  - UpdateWeights() - aktualizacja wag
  - BackProp() - użycie trzech powyższych funkcji w celu obliczenia nowych wag w całej sieci
  - GetTotalError() - wyświetlanie błędu wykorzystując warstwę wyściową
  - GetWeights() - pobranie wartości wag z wszystkich warstw
  - SetWeights() - ustawienie wartości wag we wszystkich warstwach
  - Normalize() - normalizacja danych z bazy
  - Denormalize() - denormalizacja danych otrzymanych w procesie uczenia (temperatury, wilgotności, kierunku i prędkości wiatru, zachmurzenia i widoczności
  - SetupNetwork() - pobiera ustawienia początkowe sieci z plików
  - ConfigureNetwork() - tworzy sieć o ustawieniach początkowych pobranych za pomocą powyższej funkcji
  - ForecastNextThreeDays() - dla podanych danych zwraca tablicę z dniami i godzinami dla których sieć ma przewidzieć pogodę
  - ForecastNextWeather() - zwraca przewidzianą pogodę
  - Shuffle() - tasowanie danych, by zapobiec przeuczeniu się sieci
  - GroupData() - grupuje dane

  - GetWeightsAsString() - pobiera wagi sieci, która daje najmniejszy TotalError
  - FindBestStartupSettings() - wczytuje kilka ustawień początkowych dla sieci i wybiera najlepsze (dające najmniejszy TotalError)
  - LearnNetwork() - wykorzystuje metodę wstecznej propagacji do zmiany wartości wag
  - TestNetwork() - testuje działanie sieci używając wcześniej przygotowanych danych testowych

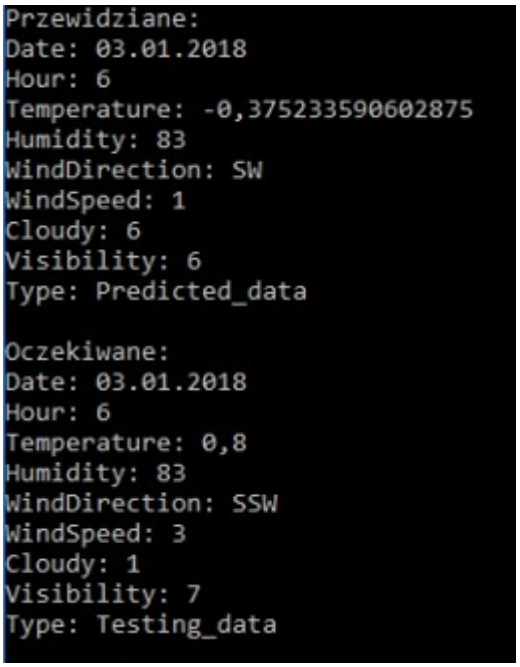
- WeatherForecast
  - GetHour() - pobiera aktualną godzinę z urządzenia klienta

Testy

Przykład warunków pogodowych przewidzianych przez naszą sieć i autentyczne dane pogodowe z tych dni.



Rysunek 4: Przykład danych otrzymanych z sieci



Rysunek 5: Przykład danych uzyskanych z sieci

Pełen kod aplikacji

```
using BusinessObject;
using Database;
using DataInsert;
using NeuralNetwork.ActivationFunctions;
using System;
using System.IO;
using DataAccessLayer;

namespace NeuralNetwork
{
    public class Program
    {
        #region Ustawienia sieci do uczenia
        public const float LearningRate = 0.05f;
        //Funkcja aktywacji najlepiej
        public static ActivationFunctionClient ActivactionFunction = new Act
        //liczba iteracji dla sieci
        private const int iterations = 150;
        private const int iterationsForStartupSettings = 30;
```

```
        //region*5, windDirection*4, date, hour, temp, humidity, windSpeed,
        private const int neuronsInput = 16;
        private const int neuronsHidden = 25;
        //windDirection*4, temp, humidity, windSpeed, cloudy, visibility
        private const int neuronsOutput = 9;
        #endregion

        /// <summary>
        /// Tasuje dane
        /// </summary>
        /// <param name="wdn">Pogrupowane dane znormalizowane</param>
        /// <returns>Zwraca tablice dwuwymiarowa wszystko potasowane ale pogr
        private static WeatherDataNormalized[][] Shuffle(WeatherDataNormaliz
        {
            int sumaDlugosci = 0;
            for (int i = 0; i < wdn.GetLength(0); i++)
                sumaDlugosci += wdn[i].GetLength(0);

            WeatherDataNormalized[][] potasowane = new WeatherDataNormalized

            //wrzucenie wszystkiego do jednej tablicy
            int l = 0;
            int k = 0;
            for (int j = 0; j < sumaDlugosci; j++, k++)
            {
                if (wdn[l].GetLength(0) * (l + 1) == j)
                {
                    k = 0;
                    l++;
                }
                potasowane[j] = new WeatherDataNormalized[2];
                potasowane[j][0] = wdn[l][k, 0];
                potasowane[j][1] = wdn[l][k, 1];
            }

            Random random = new Random();
            int zakres = potasowane.Length;
//rozmiar listy
            WeatherDataNormalized[] temp;                       //pomocnicza zm
            int rnd;                                            //jaki indeks za


            for (int i = 0; i < zakres; i++)
            {
                rnd = random.Next(0, zakres);
```

```
            temp = potasowane[i];
            potasowane[i] = potasowane[rnd];
            potasowane[rnd] = temp;
        }

        return potasowane;
    }

    /// <summary>
    /// Grupuje dane, [i][j,0] to warstwa wejsciowa, a [i][j,1] to wartwa
    /// </summary>
    /// <param name="weatherDataBGN">Ktore dane grupuje, Before Grouping
    /// <param name="weatherDataGN">Do czego pogrupowac, Grouped Normali
    private static void GroupData(WeatherDataNormalized[][] weatherDataBG
    {
        for (int i = 0; i < weatherDataGN.GetLength(0); i++)
        {
            weatherDataGN[i] = new WeatherDataNormalized[weatherDataBGN[
            for (int j = 0; j < weatherDataGN[i].GetLength(0); j++)
            {
                weatherDataGN[i][j, 0] = weatherDataBGN[i][j];
                weatherDataGN[i][j, 1] = weatherDataBGN[i][j + 1];
            }
        }
    }

    /// <summary>
    /// Grupuje dane, [i][j,0] to warstwa wejsciowa, a [i][j,1] to wartwa
    /// </summary>
    /// <param name="weatherDataBG">Ktore dane grupuje, Before Grouping<
    /// <param name="weatherDataG">Do czego pogrupowac, Grouped</param>
    private static void GroupData(WeatherData[][] weatherDataBG, Weather
    {
        for (int i = 0; i < weatherDataG.GetLength(0); i++)
        {
            weatherDataG[i] = new WeatherData[weatherDataBG[i].Length -
            for (int j = 0; j < weatherDataG[i].GetLength(0); j++)
            {
                weatherDataG[i][j, 0] = weatherDataBG[i][j];
                weatherDataG[i][j, 1] = weatherDataBG[i][j + 1];
            }
        }
    }

    static void Main(string[] args)
```

```
{
    ReadDataFile rd = new ReadDataFile();
    Normalization normalization = new Normalization();
    Denormalization denormalization = new Denormalization();

    //glowna siec do uczenia
    Network network = new Network(new int[] { neuronsInput, neuronsH

    #region Krok 1 - Uczenie sieci
    string weightsAsString = "";
    rd.LoadData("..\\..\\..\\..\\..\\DatabaseSources\\Data\\LearningI
        BusinessObject.DataTypes.Learning_data);

    WeatherData[][] weatherDataL = rd.WeatherLearningGroupedDatas;
    WeatherData[][,] weatherDataLG = new WeatherData[weatherDataL.Ge
    GroupData(weatherDataL, weatherDataLG);

    WeatherDataNormalized[][] weatherDataLN = new WeatherDataNormali
    for (int i = 0; i < weatherDataLN.Length; i++)
        weatherDataLN[i] = normalization.Normalize(weatherDataL[i],

    WeatherDataNormalized[][,] weatherDataLGN = new WeatherDataNorma
    GroupData(weatherDataLN, weatherDataLGN);
    WeatherDataNormalized[][] weatherDataLGNS = Shuffle(weatherDataLG

    #region Krok 1.5 - wybieranie wag i danych do nauki
    //Wybieranie najlepszych wag dla punktu startowego
    const int numberOfNetworksToTest = 30;
    Network bestNetwork = null;
    WeatherDataNormalized[][] bestDataset = null;

    Console.WriteLine($"Wybieranie najlepszych ustawien startowych z
    int bestIndex = FindBestStartupSettings(numberOfNetworksToTest,

    Console.WriteLine(Environment.NewLine + $"Siec o indeksie {bestIn
        $" a jej blad MSE wynosi {bestNetwork.GetTotalError()}" + En

    float[][,] bestWeights = bestNetwork.GetWeights();

    //zapisuje wagi do pliku
    weightsAsString = GetWeightsAsString(bestWeights);
    File.WriteAllText("..\\..\\..\\..\\..\\DatabaseSources\\Data\\be

    network.SetWeights(bestWeights);
    weatherDataLGNS = bestDataset;
```

```
        #endregion

        Console.WriteLine("Rozpoczynam nauke glownej sieci...");
        LearnNetwork(weatherDataLGNS, network);
        //zapisuje wagi nauczone do pliku
        weightsAsString = GetWeightsAsString(network.GetWeights());
        File.WriteAllText("..\\..\\..\\..\\..\\DatabaseSources\\Data\\Ne

        //zapisuje ustawienia sieci do pliku
        string networkSettings = neuronsInput.ToString() + " " + neuronsI
        File.WriteAllText("..\\..\\..\\..\\..\\DatabaseSources\\Data\\Ne
        #endregion

        #region Krok 2 - Testowanie sieci

        rd.LoadData("..\\..\\..\\..\\..\\DatabaseSources\\Data\\TestingD
            BusinessObject.DataTypes.Testing_data);

        WeatherData[][] weatherDataT = rd.WeatherTestingGroupedDatas;
        WeatherData[][,] weatherDataTG = new WeatherData[weatherDataT.Ge
        GroupData(weatherDataT, weatherDataTG);
        WeatherDataNormalized[][] weatherDataTN = new WeatherDataNormali
        for (int i = 0; i < weatherDataTN.Length; i++)
            weatherDataTN[i] = normalization.Normalize(weatherDataT[i],

        WeatherDataNormalized[][,] weatherDataTGN = new WeatherDataNorma
        GroupData(weatherDataTN, weatherDataTGN);
        TestNetwork(denormalization, network, weatherDataTG, weatherDataT

        #endregion

        Console.WriteLine(Environment.NewLine + Environment.NewLine);
        Console.WriteLine("Koniec pracy programu. Nacisnij dowolny przyc
        Console.ReadKey();
    }

    /// <summary>
    /// Testuje siec
    /// </summary>
    /// <param name="denormalization"></param>
    /// <param name="network"></param>
    /// <param name="weatherDataTG"></param>
    /// <param name="weatherDataTGN"></param>
    private static void TestNetwork(Denormalization denormalization, Netw
    {
```

```
for (int i = 0; i < weatherDataTGN.GetLength(0); i++)
{
    for (int j = 0; j < weatherDataTGN[i].GetLength(0); j++)
    {
        //wartosci neuronow
        float[] PredictedNeuronsN = network.FeedForward(new floa
            (float)weatherDataTGN[i][j,0].WindDirection[0], (flo
             (float)weatherDataTGN[i][j,0].Date,(float)weatherDat

        //Oczekiwane dane pogodowe
        WeatherData expectedWeatherData = weatherDataTG[i][j, 1]

        //windDirection*4, temp, humidity, windSpeed, cloudy, vi
        WeatherDataNormalized predictedWeatherDataN = new Weathe
            PredictedNeuronsN[4], PredictedNeuronsN[5], Predicted
        WeatherData predictedWeatherData = denormalization.Denor

        predictedWeatherData.DataType = DataTypes.Predicted_data
        predictedWeatherData.CityId = expectedWeatherData.CityId

        //przewiduje na kolejne 6 lub 12 godzin
        if (weatherDataTG[i][j, 0].Hour == 6)
            predictedWeatherData.Hour = 12;
        else if (weatherDataTG[i][j, 0].Hour == 12)
            predictedWeatherData.Hour = 18;
        else
            predictedWeatherData.Hour = 6;


        int rok = weatherDataTG[i][j, 0].Date.Year;
        int miesiac = weatherDataTG[i][j, 0].Date.Month;
        int dzien = weatherDataTG[i][j, 0].Date.Day;

        //jezeli przewidzialo na kolejny dzien
        if (predictedWeatherData.Hour == 6)
        {
            dzien++;

            int[] Days = new int[] { 31, 28, 31, 30, 31, 30, 31,
            int[] LeapYearDays = new int[] { 31, 29, 31, 30, 31,


            //jezeli nieprzestepny
            if (rok % 4 != 0)
            {
```

```
                    //To znaczy ze przekroczyl zakres dni w danym mi
                    if (dzien - Days[miesiac - 1] > 0)
                    {
                        miesiac++;
                        dzien = 1;
                    }
                }
                else
                {
                    if (dzien - LeapYearDays[miesiac - 1] > 0)
                    {
                        dzien = 1;
                        miesiac++;
                    }
                }


            }

            if (miesiac == 13)
            {
                miesiac = 1;
                rok += 1;
            }

            predictedWeatherData.Date = new DateTime(rok, miesiac, d

            Console.WriteLine("Przewidziane: ");
            Console.WriteLine(predictedWeatherData.ToString() + Envi
            Console.WriteLine("Oczekiwane: ");
            Console.WriteLine(expectedWeatherData.ToString() + Enviro
            Console.WriteLine("Wcisnij przycisk, by kontynowac..." +
            Console.ReadKey();
        }
    }
}

/// <summary>
/// Pobiera wagi sieci, ktora daje najmniejszy TotalError
/// </summary>
/// <param name="weights">Wagi danej sieci</param>
/// <returns>Wagi jako string. Warstwa oddzielona jest srednikiem</re
private static string GetWeightsAsString(float[][,] weights)
{
    string weightsAsString = "";
```

```
        for (int i = 0; i < weights.GetLength(0); i++)
        {
            for (int j = 0; j < weights[i].GetLength(0); j++)
                for (int k = 0; k < weights[i].GetLength(1); k++)
                    weightsAsString = weightsAsString + weights[i][j, k]

            //aby na koncu nie dawalo srednika
            if (i != weights.GetLength(0) - 1)
                weightsAsString += ";";
        }

        return weightsAsString;
    }

    /// <summary>
    /// Znajduje najlepsze ustawienia sieci.
    /// </summary>
    /// <param name="ileSieci">ilosc sieci do przeszukania</param>
    /// <param name="weatherDataLGN">Znormalizowane dane do nauki, pogrup
    /// <param name="bestNetwork">referencja do ktorej zostanie przypisan
    /// <param name="bestDataset">referencja do ktorej zostanie przypisan
    /// <returns>Zwraca indeks najlepszej sieci</returns>
    private static int FindBestStartupSettings(int ileSieci, ref Weather
    {
        int percent = 0;
        Network[] networks = new Network[ileSieci];
        WeatherDataNormalized[][][] datasets = new WeatherDataNormalized

        for (int i = 0; i < ileSieci; i++)
        {
            networks[i] = new Network(new int[] { neuronsInput, neuronsH
            datasets[i] = Shuffle(weatherDataLGN);

            for (int k = 0; k < iterationsForStartupSettings; k++)
            {
                for (int j = 0; j < datasets[i].GetLength(0); j++)
                {
                    networks[i].FeedForward(new float[neuronsInput] {dat
                        (float)datasets[i][j][0].WindDirection[0], (floa
                         (float)datasets[i][j][0].Date,(float)datasets[i
                    });
                    networks[i].BackProp(new float[neuronsOutput] {(floa
                         (float)datasets[i][j][1].Temperature,(float)dat
                    });
                }
```

```
                    if (((float)k / iterationsForStartupSettings) * 100 > pe
                        percent++;

                    Console.Write($"\rSiec {i + 1}/{networks.Length}\tTotal
");


                    //Po 3% pobiera TotalError i przechodzi do kolejnej siec
                    if (percent >= 3)
                    {
                        percent = 0;
                        break;
                    }

                }
            }

            //poczatkowe przypisanie
            bestNetwork = networks[0];
            bestDataset = datasets[0];
            int indexOfBestNetwork = 0;

            //znalezienie najlepszej sieci i najlepszego datasetu
            for (int i = 0; i < ileSieci; i++)
            {
                if (networks[i].GetTotalError() < bestNetwork.GetTotalError(
                {
                    bestNetwork = networks[i];
                    bestDataset = datasets[i];
                    indexOfBestNetwork = i;
                }
            }

            return indexOfBestNetwork;
        }

        /// <summary>
        /// Uczenie sieci
        /// </summary>
        /// <param name="weatherDataLearning">Dane do nauki</param>
        /// <param name="network">Siec ktora uczy</param>
        private static void LearnNetwork(WeatherDataNormalized[][] weatherDa
        {
            int percent = 0;
            for (int i = 0; i < iterations; i++)
```

```
            {
                for (int j = 0; j < weatherDataLearning.Length; j++)
                {
                    network.FeedForward(new float[neuronsInput] {weatherData
                        (float)weatherDataLearning[j][0].WindDirection[0], (
                         (float)weatherDataLearning[j][0].Date,(float)weathe
                    });
                    network.BackProp(new float[neuronsOutput] {(float)weathe
                         (float)weatherDataLearning[j][1].Temperature,(float
                    });
                }

                Console.Write($"\r{percent}%    ");
                Console.Write($"\r{percent}% \t Total Error: {network.GetTot
");

                if (((float)i / iterations) * 100 > percent)
                    percent++;

            }
        }

    }
}


/*
 * L - learning data
 * T - testing data
 * G - grouped data
 * N - normalised data
 * S - shuffled data
*/
```