

RegressionSolver w chmurze

Błażej Strus, 164489

Opis projektu

Aplikację RegressionSolver stworzyłem na zaliczenie przedmiotu „Programowanie w chmurze – Projekt”, Studia INF:29 U1 S1. Zaimplementowałem prostą sieć neuronową z propagacją wsteczną w celu obliczania regresji liniowej wielomianów. Umożliwia użytkownikowi trenowanie modelu regresji liniowej na podstawie dostarczonych danych, a następnie wykorzystanie tego modelu do prognozowania nowych wartości. Stworzyłem ją by pokazać możliwość tworzenia aplikacji w chmurze.

Regresja liniowa to metoda statystyczna, która pozwala na modelowanie zależności między zmienną zależną a jedną lub większą liczbą zmiennych niezależnych. W przypadku regresji liniowej pierwszego stopnia (czyli prostej regresji liniowej) model opisuje zależność liniową między dwiema zmiennymi – jedną zależną y oraz jedną niezależną x . N -ty stopień, to równanie: $y = \sum_{i=0}^n w_i \cdot x^i$. Celem jest znalezienie funkcji, która najlepiej pasuje do zbioru punktów danych poprzez minimalizację sumy kwadratów różnic między obserwowanymi a przewidywanymi wartościami.

W aplikacji zastosowano ręczną implementację sieci neuronowej, która w tym kontekście odnosi się do procesu optymalizacji wag modelu regresji liniowej. Użytkownik może dostosować proces uczenia, ustawiając stopień wielomianu, liczbę epok (iteracji trenowania) oraz cierpliwość, która determinuje, po ilu epokach bez poprawy model zmniejszy tempo uczenia. Sieć wykorzystuje **metodę spadku gradientu** do aktualizacji wag, co oznacza iteracyjne dostosowywanie wag w kierunku największego spadku funkcji straty (średni błąd kwadratowy), aby znaleźć minimalne wartości funkcji kosztu.

W praktyce, aplikacja ta może służyć jako narzędzie dydaktyczne do zrozumienia podstawowych koncepcji związanych z regresją liniową i uczeniem maszynowym, jak również jako proste narzędzie do przeprowadzania analiz predykcyjnych.

Kod programu w C#

Adres aplikacji na replit.com: <https://replit.com/@BlazejStrus/RegressionSolver>

Kod źródłowy: <https://github.com/blablast/RegressionSolver.git>

Program.cs

Jest główną klasą, która koordynuje działania aplikacji. Uruchamia interakcję z użytkownikiem, pozyskuje dane, inicjalizuje i trenuje model, a następnie pozwala na przewidywanie nowych wartości.

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
namespace RegressionSolver
{
    public class Program
    {
        private readonly static Settings settings = new();
        private static LinearRegression? model;
        public static async Task Main()
        {

```

```

        PrintHeader();
        GetUserInputs();
        PrintSummary();

        var readDataPoints = GetDataPointsInput();
        if (string.IsNullOrEmpty(readDataPoints))
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("No data points provided. Exiting...");
            Console.ResetColor();
            return;
        }
        var dataPoints = ParseDataPoints(readDataPoints, settings.Degree);
        model = new LinearRegression(settings);
        await model.TrainAsync(dataPoints, UpdateProgress, settings.Patience);

        await InteractivePredictionLoop(model);
    }

    private static void PrintHeader()
    {
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine("Linear Regression Model Trainer");
        Console.WriteLine("Author: Blazej Strus (164489)");
        Console.WriteLine("Date: 2024-03-21");
        Console.WriteLine("Version: 1.0");
        Console.WriteLine("This program trains a linear regression model based on user-
provided data\nand allows to predict values, based on trained model.");
        Console.WriteLine();
        Console.ResetColor(); // Reset to default color
    }

    private static void PrintSummary()
    {
        Console.ForegroundColor = ConsoleColor.DarkBlue;
        Console.WriteLine($"Summary: Degree = {settings.Degree}, Epochs =
{settings.Epochs}");
        Console.WriteLine();
        Console.ResetColor(); // Reset to default color
    }

    private static void UpdateProgress(int epoch, double loss, string weights, double
learningRate)
    {
        Console.SetCursorPosition(0, Console.CursorTop);
        Console.ForegroundColor = ConsoleColor.DarkYellow;
        string learningRateFormatted = $"{learningRate:E2}".ToLower().Replace("1,00",
"1").Replace("1.00", "1").Replace("e+00", "e+0").Replace("e-00", "e-0").Replace("e+0",
"e+").Replace("e-0", "e-");
        string formattedLoss;
        if (((int)loss).ToString().Length > 5)
        {
            formattedLoss = $"..." + ((int)loss).ToString()[^5..];
        }
        else
        {
            formattedLoss = $"{loss,10:F5}";
        }

        string progress = $"Epoch {epoch,7}: Loss = {formattedLoss} | Weights: {weights} |
Learning Rate: {learningRateFormatted}";
        Console.Write(progress.PadRight(Console.WindowWidth - 1));
        Console.ResetColor();
    }

    private static void GetUserInputs()
    {
        settings.Degree = InputInt($"Enter the degree of polynomial features (1 - for linear,
2 for quadratic, etc.), (default = {settings.Degree}):", settings.Degree);
    }

```

```

        settings.Epochs = InputInt($"Enter the number of epochs (default = {settings.Epochs}):", settings.Epochs);
        settings.Patience = InputInt($"Enter the number of epochs to wait before reducing the learning rate (default patience = {settings.Patience}):", settings.Patience);
    }

    private static int InputInt(string prompt, int defaultValue)
    {
        Console.WriteLine(prompt);
        return int.TryParse(Console.ReadLine(), out int value) ? value : defaultValue;
    }

    private static string? GetDataPointsInput()
    {
        Console.WriteLine("Enter data points (x1,x2,...,xn,y; x1,x2,...,xn,y; ...):");
        return Console.ReadLine();
    }

    private static async Task InteractivePredictionLoop(LinearRegression model)
    {
        while (true)
        {
            Console.WriteLine("Enter new x values to predict y (leave empty and press enter to exit):");
            var newXInput = Console.ReadLine();
            if (string.IsNullOrEmpty(newXInput)) { break; }

            var newXValues = newXInput.Split(',').Select(n => double.TryParse(n, out double x) ? x : 0).ToArray();
            var augmentedXValues = AugmentFeatures(newXValues, settings.Degree);
            var predictedY = model.Predict(augmentedXValues);
            Console.WriteLine($"Predicted y: {predictedY}");
        }
        await Task.CompletedTask;
    }

    private static List<DataPoint> ParseDataPoints(string input, int degree)
        => input.Split(';').Select(point => point.Split(',').Select(s => double.Parse(s, CultureInfo.InvariantCulture)).ToArray()).Select(values => new DataPoint(AugmentFeatures(values[..^1], degree), values[^1])).ToList();

    private static double[] AugmentFeatures(double[] features, int degree)
        => features.SelectMany(f => Enumerable.Range(1, degree).Select(pow => Math.Pow(f, pow))).ToArray();
    }
}

```

LinearRegression.cs

Zawiera logikę odpowiedzialną za przeprowadzenie regresji liniowej.

```

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

namespace RegressionSolver
{
    internal class LinearRegression
    {
        public double[] Weights { get; private set; }
        public List<double> LossHistory = new();
        public double LearningRate { get; set; }
        public double MinLearningRate { get; set; }
        public int Epochs { get; set; }
    }
}

```

```

private CultureInfo culture = CultureInfo.InvariantCulture; // Using InvariantCulture

public LinearRegression(Settings settings)
{
    Epochs = settings.Epochs > 0 ? settings.Epochs : throw new
ArgumentOutOfRangeException(nameof(settings.Epochs));
    LearningRate = settings.LearningRate;
    MinLearningRate = settings.MinLearningRate;
    Weights = Array.Empty<double>();
}

// Computes the mean squared error loss
private double ComputeLoss(IEnumerable<DataPoint> dataPoints)
=> dataPoints.Average(dataPoint => Math.Pow(dataPoint.Label -
Predict(dataPoint.Features), 2));

// Makes a prediction based on the input features and learned weights
public double Predict(double[] features)
=> Math.Round(Weights[0] + features.Select((t, i) => t * Weights[i + 1]).Sum(), 10);

// Trains the model using the provided data points
public async Task TrainAsync(List<DataPoint> dataPoints, Action<int, double, string,
double> updateProgress, int patience)
{
    int featureCount = dataPoints.First().Features.Length;
    var random = new Random();
    Weights = Enumerable.Range(0, featureCount + 1).Select(_ => random.NextDouble() *
0.01 - 0.005).ToArray();
    string numberFormat = "F5";
    double previousLoss = double.MaxValue;
    int patienceCounter = 0;
    var minLearningRate = 1e-7;

    for (int epoch = 0; epoch < Epochs; epoch++)
    {
        UpdateWeights(dataPoints);
        double totalLoss = ComputeLoss(dataPoints);
        LossHistory.Add(totalLoss);
        if (totalLoss >= previousLoss)
        {
            patienceCounter++;
            if (patienceCounter >= patience)
            {
                if (LearningRate == minLearningRate)
                {
                    updateProgress(epoch, totalLoss, string.Join(", ", Weights.Select(w
=> w.ToString(numberFormat, culture))), LearningRate);
                    PrintError("Training stopped due to patience limit.");
                    break;
                }
                else
                {
                    LearningRate = Math.Max(LearningRate / 10, MinLearningRate);
                    patienceCounter = 0;
                }
            }
        }
        else
        {
            patienceCounter = 0;
        }
        if (totalLoss < previousLoss)
        {
            updateProgress(epoch, totalLoss, string.Join(", ", Weights.Select(w =>
w.ToString(numberFormat, culture))), LearningRate);
        }
        previousLoss = totalLoss;
        if (double.IsNaN(totalLoss))
    }
}

```

```

        {
            //PrintError("Training stopped due to NaN loss.");
            //break;
        }

    }

    Console.WriteLine(); // Ensure the final output starts on a new line
    DisplayFinalWeights(numberFormat);
    await Task.CompletedTask;
}

private static void PrintError(string message)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine($"{message}");
    Console.ResetColor();
}

// Updates the weights based on gradients calculated from the data points
private void UpdateWeights(List<DataPoint> dataPoints)
{
    double[] totalGradients = new double[Weights.Length];

    foreach (var dataPoint in dataPoints)
    {
        double error = dataPoint.Label - Predict(dataPoint.Features);
        totalGradients[0] += -2 * error; // Gradient for bias
        for (int i = 1; i < Weights.Length; i++)
        {
            totalGradients[i] += -2 * error * dataPoint.Features[i - 1]; // Gradient for
each feature weight
        }
    }

    for (int i = 0; i < Weights.Length; i++)
    {
        Weights[i] -= (LearningRate / dataPoints.Count) * totalGradients[i]; // Update
weights
    }
}

// Displays the final weights after training
private void DisplayFinalWeights(string numberFormat)
{
    // Constructing the equation string
    StringBuilder equation = new("y = ");
    for (int i = 0; i < Weights.Length; i++)
    {
        // Append each weight and its corresponding x term, excluding the first weight
(intercept)
        equation.Append((i > 0)
            ? $" {(Weights[i] >= 0 ? "+" : "-")}"
            : $"{Math.Abs(Weights[i]).ToString(numberFormat, culture)} * x^{i}"
            : $"{Weights[i].ToString(numberFormat, culture)}");
    }

    StringBuilder sbHeader = new StringBuilder("|");
    StringBuilder sbValues = new StringBuilder("| ");
    var j = 0;
    foreach (var weight in Weights)
    {
        sbHeader.Append($" w{j.ToString().PadRight(12)}|");
        sbValues.Append(weight.ToString(numberFormat, culture).PadLeft(12)).Append(" |");
        j++;
    }

    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine($"Equation: {equation}");
    Console.WriteLine("Final weights:");

```

```
        Console.WriteLine(sbHeader.ToString());
        Console.WriteLine(sbValues.ToString());
        Console.ResetColor();
    }
}
```

DataPoint.cs

Reprezentuje pojedynczy punkt danych, składający się z wektora cech i etykiety (wartości y).

```
using System;
namespace RegressionSolver
{
    public class DataPoint
    {
        public double[] Features { get; }
        public double Label { get; }

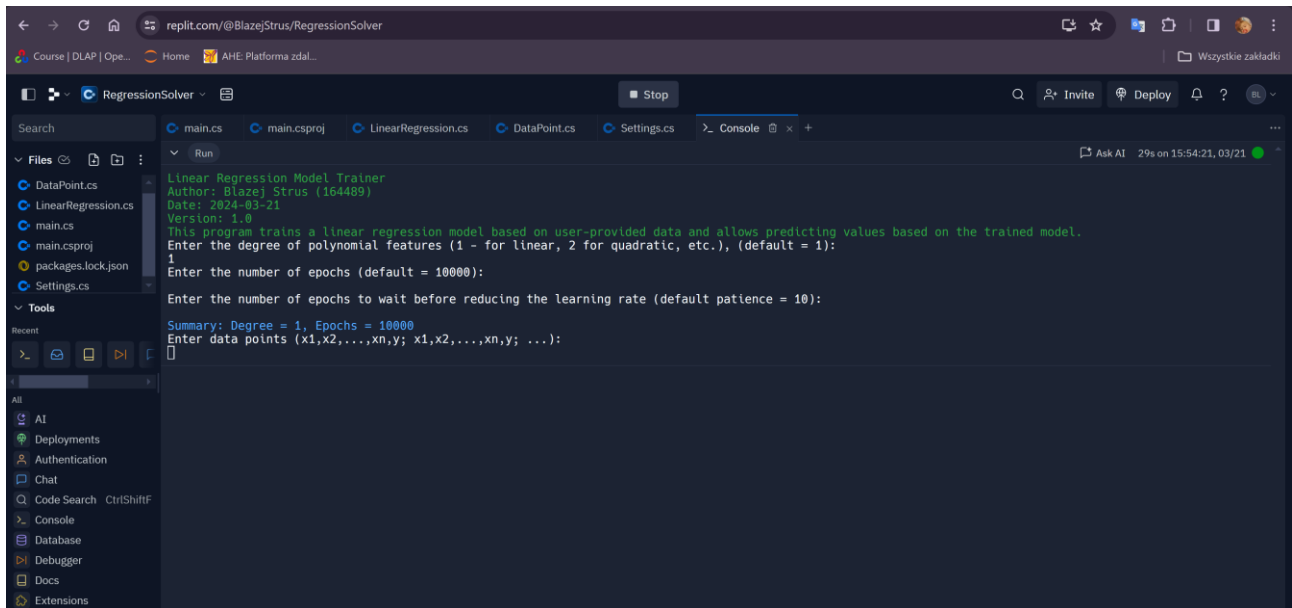
        public DataPoint(double[] features, double label)
        {
            Features = features ?? throw new ArgumentNullException(nameof(features));
            Label = label;
        }
    }
}
```

Settings.cs

Przechowuje ustawienia konfiguracyjne dla modelu, które użytkownik może określić.

```
using System;
namespace RegressionSolver
{
    internal class Settings
    {
        private int degree = 1;
        public int Patience { get; set; } = 6;
        public int Degree
        {
            get => degree;
            set
            {
                degree = value;
                LearningRate = Math.Pow(1e-2, degree);
                MinLearningRate = 1e-5 * LearningRate;
                Epochs = (int)(1000 * Math.Pow(10, degree));
            }
        }
        public int Epochs { get; set; } = 10000;
        public double LearningRate { get; set; }
        public double MinLearningRate { get; set; }
    }
}
```

Przebieg programu

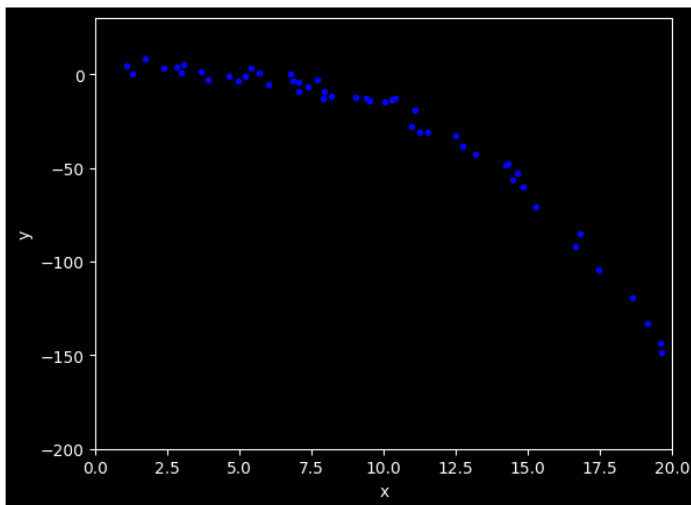


Rysunek 1. Interakcja początkowa - ustawienia modelu

Dla celów testowych wygenerowałem 50 punktów funkcji:

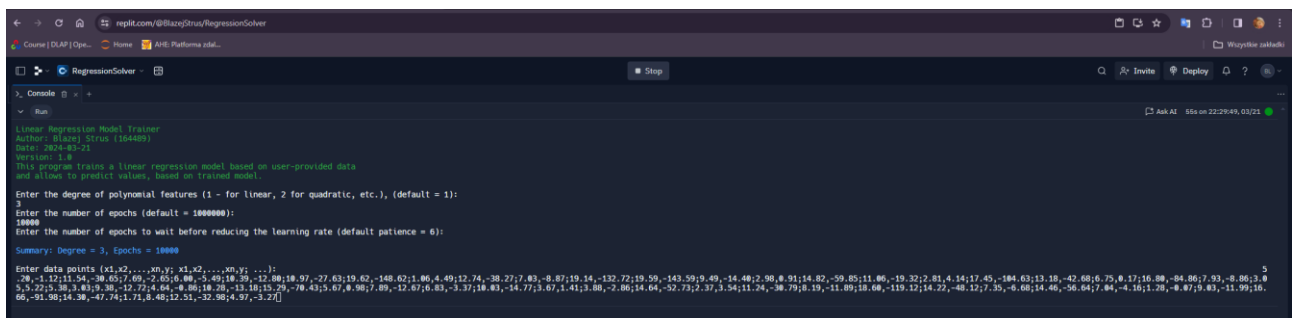
$$y = -1 + 0.02 \cdot x + 0.02 \cdot x^2 - 0.02 \cdot x^3 + \text{noise}$$

dla x w przedziale <0; 20>



5.20,-1.12;11.54,-30.65;7.69,-2.65;6.00,-5.49;10.39,-12.80;10.97,-27.63;19.62,-148.62;1.06,4.49;12.74,-38.27;7.03,-8.87;19.14,-132.72;19.59,-143.59;9.49,-14.40;2.98,0.91;14.82,-59.85;11.06,-19.32;2.81,4.14;17.45,-104.63;13.18,-42.68;6.75,0.17;16.80,-84.86;7.93,-8.86;3.05,5.22;5.38,3.03;9.38,-12.72;4.64,-0.86;10.28,-13.18;15.29,-70.43;5.67,0.98;7.89,-12.67;6.83,-3.37;10.03,-14.77;3.67,1.41;3.88,-2.86;14.64,-52.73;2.37,3.54;11.24,-30.79;8.19,-11.89;18.60,-119.12;14.22,-48.12;7.35,-6.68;14.46,-56.64;7.04,-4.16;1.28,-0.07;9.03,-11.99;16.66,-91.98;14.30,-47.74;1.71,8.48;12.51,-32.98;4.97,-3.27

Rysunek 2. Dane



Rysunek 3. Start treningu

```
RegressionSolver

> Console
Run

Linear Regression Model Trainer
Author: Blazej Strus (164489)
Date: 2024-03-21
Version: 1.0
This program trains a linear regression model based on user-provided data
and allows to predict values, based on trained model.

Enter the degree of polynomial features (1 - for linear, 2 for quadratic, etc.), (default = 1):
3
Enter the number of epochs (default = 1000000):
10000
Enter the number of epochs to wait before reducing the learning rate (default patience = 6):

Summary: Degree = 3, Epochs = 10000

Enter data points (x1,x2,...,xn,y; x1,x2,...,xn,y; ...):
.20,-1.12;11.54,-30.65;7.69,-2.65;6.00,-5.49;10.39,-12.80;10.97,-27.63;19.62,-148.62;1.06,4.49;12.74,-38.1
5,5.22;5.38,3.03;9.38,-12.72;4.64,-0.86;10.28,-13.18;15.29,-70.43;5.67,0.98;7.89,-12.67;6.83,-3.37;10.03,-
66,-91.98;14.30,-47.74;1.71,8.48;12.51,-32.98;4.97,-3.27
Epoch      60: Loss = 16.80956 | Weights: 0.00483, 0.00195, -0.00239, -0.01867 | Learning Rate: 1e-7

RegressionSolver

> Console
Run

Linear Regression Model Trainer
Author: Blazej Strus (164489)
Date: 2024-03-21
Version: 1.0
This program trains a linear regression model based on user-provided data
and allows to predict values, based on trained model.

Enter the degree of polynomial features (1 - for linear, 2 for quadratic, etc.), (default = 1):
3
Enter the number of epochs (default = 1000000):
10000
Enter the number of epochs to wait before reducing the learning rate (default patience = 6):

Summary: Degree = 3, Epochs = 10000

Enter data points (x1,x2,...,xn,y; x1,x2,...,xn,y; ...):
.20,-1.12;11.54,-30.65;7.69,-2.65;6.00,-5.49;10.39,-12.80;10.97,-27.63;19.62,-148.62;1.06,4.49;12.74,-38.1
5,5.22;5.38,3.03;9.38,-12.72;4.64,-0.86;10.28,-13.18;15.29,-70.43;5.67,0.98;7.89,-12.67;6.83,-3.37;10.03,-
66,-91.98;14.30,-47.74;1.71,8.48;12.51,-32.98;4.97,-3.27
Epoch      2537: Loss = 14.59197 | Weights: 0.00059, 0.00351, 0.01995, -0.01999 | Learning Rate: 1e-7
```

Rysunek 4. Obliczanie wag


```

RegressionSolver

> Console x +
  Run

Linear Regression Model Trainer
Author: Blazej Strus (164489)
Date: 2024-03-21
Version: 1.0
This program trains a linear regression model based on user-provided data
and allows to predict values, based on trained model.

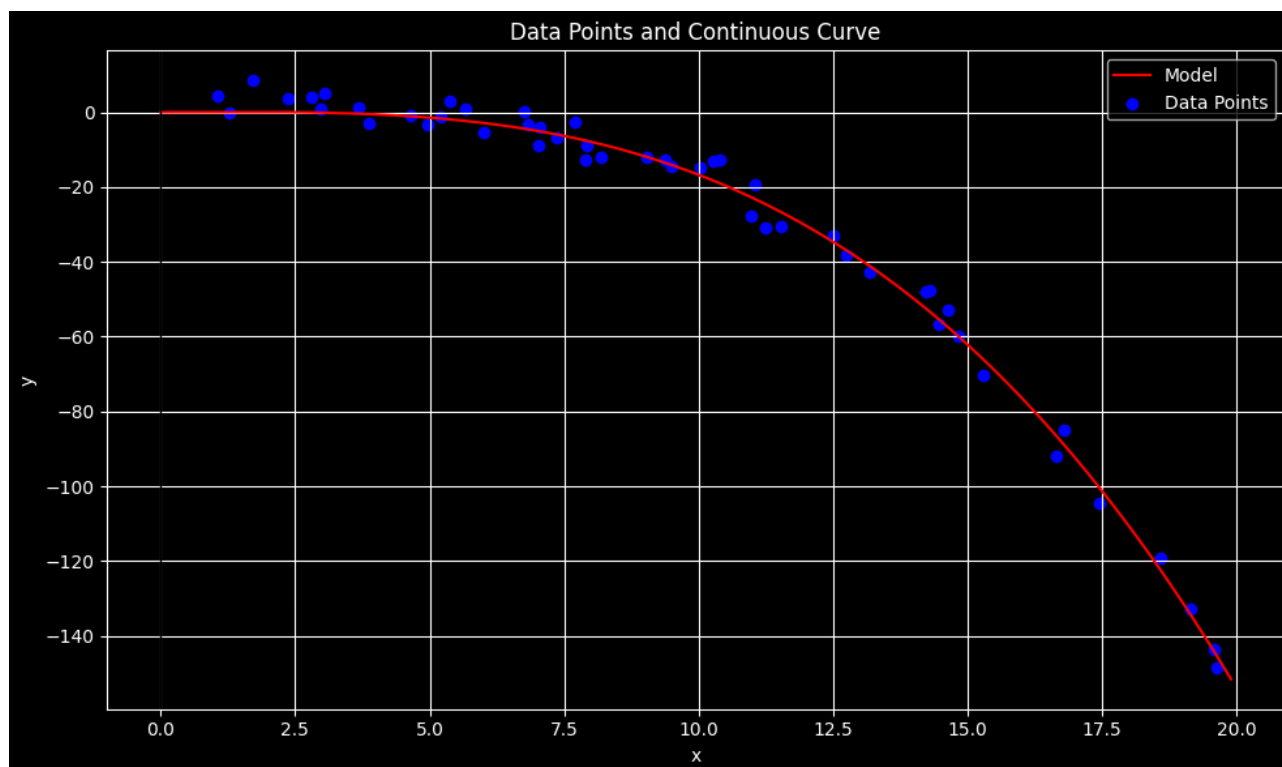
Enter the degree of polynomial features (1 - for linear, 2 for quadratic, etc.), (default = 1):
3
Enter the number of epochs (default = 1000000):
10000
Enter the number of epochs to wait before reducing the learning rate (default patience = 6):

Summary: Degree = 3, Epochs = 10000

Enter data points (x1,x2,...,xn,y; x1,x2,...,xn,y; ...):
.20,-1.12;11.54,-30.65;7.69,-2.65;6.00,-5.49;10.39,-12.80;10.97,-27.63;19.62,-148.62;1.06,4.49;12.74,-38.27;7.03
5,5.22;5.38,3.03;9.38,-12.72;4.64,-0.86;10.28,-13.18;15.29,-70.43;5.67,0.98;7.89,-12.67;6.83,-3.37;10.03,-14.77;
66,-91.98;14.30,-47.74;1.71,8.48;12.51,-32.98;4.97,-3.27
Epoch 9999: Loss = 13.18301 | Weights: 0.00188, 0.00921, 0.04923, -0.02173 | Learning Rate: 1e-7
Equation: y = 0.00188 + 0.00921 * x^1 + 0.04923 * x^2 - 0.02173 * x^3
Final weights:
| w0 | w1 | w2 | w3 |
| 0.00188 | 0.00921 | 0.04923 | -0.02173 |
Enter new x values to predict y (leave empty and press enter to exit):

```

Rysunek 5. Wagi obliczone - można spróbować predykcji



Rysunek 6. Dane wejściowe i model obliczony przez zaimplementowaną sieć.

```
Equation: y = 0.00188 + 0.00921 * x^1 + 0.04923 * x^2 - 0.02173 * x^3
```

Rysunek 7. Obliczone wagi

Wnioski

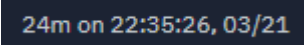
Uruchamianie kodu w chmurze, na platformie Replit.com, ma szereg zalet i wad, które zauważam:

Zalety:

- Dostępność z każdego miejsca z podłączonego do internetu.
- Łatwość współpracy.
- Brak konieczności konfiguracji środowiska.
- Skalowalność.
- Bezpieczeństwo.

Wady:

- Zależność od połączenia z internetem.
- Prywatność.
- Koszty.
- Ograniczenia zasobów.
- Opóźnienia.

Najbardziej dokuczliwą wadą są opóźnienia. To co lokalnie na komputerze zajmuje około 2sek, w darmowym planie redlit.com zajmuje 24 min: . Jednakże istnieją również doskonałe platformy jak <https://colab.research.google.com/> czy <https://www.kaggle.com/>, na których programując w Pythonie mamy dostęp nawet do GPU, co w przypadku trenowania sieci neuronowych jest ogromnym plusem.