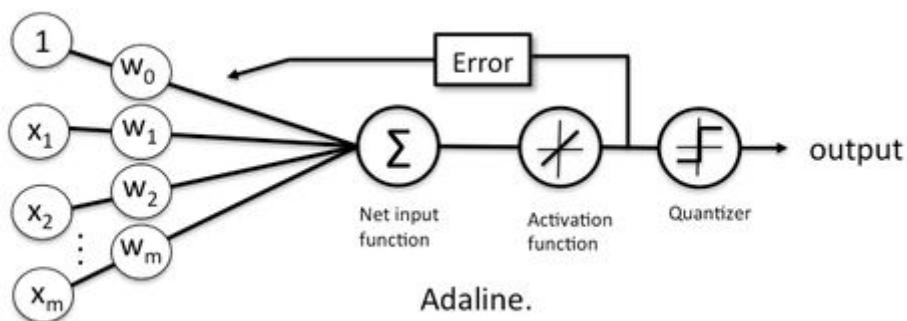
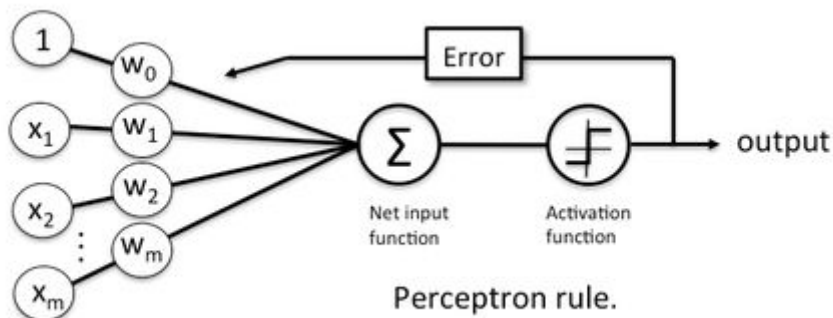


Imię Nazwisko Grupa Michał Słowikowski Gr 4	Temat Scenariusz 2	Data 10.11.2017r.
---	-----------------------	----------------------

Celem ćwiczenia było nauczenie jednowarstwowej sieci neuronowej do rozpoznawania dużych i małych liter

Do ćwiczenia wykorzystałem perceptron oraz Adaptive Linear Neuron czyli Adaline.

Syntetyczny opis algorytmu uczenia



Do wykonania ćwiczenia w pierwszej kolejności wykorzystałem sieć neuronową składającą się z dwóch neuronów sigmoidalnych. Wykorzystałem taki typ neuronu, ponieważ potrzebowałem ciągłych wartości w przedziale $[0,1]$, ze względu na metodykę decydowania o wielkości litery. Unipolarną funkcję sigmoidalną możemy zapisać wzorem:

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

Każdy z nich badał jakiej wielkości jest litera. Pierwszy szukał liter dużych, drugi liter małych. Decyzja czy litera jest duża czy mała była podejmowana na podstawie który neuron zwrócił większą wartość. Im bliżej 1 tym większe prawdopodobieństwo, że litera jest danej wielkości.

Nowe wagi obliczane są przy pomocy funkcji:

$$w = w + \eta * (o - y) * f'(z) * x$$

Gdzie w po prawej stronie to stara waga, η to learning rate, $o-y$ to różnica wyjścia i oczekiwanej, $f'(z)$ to pochodna funkcji aktywacji równa $y*(1-y)$ a x to wejście

W drugim przypadku podobnie wykorzystałem sieć składającą się z dwóch neuronów Adaline. Jako funkcję aktywacji przyjąłem signum. Był on uczony zgodnie z regułą:

$$w(k+1) = w(k) + \eta x(k)[d(k) - w^T(k)x(k)]$$

, gdzie człon $[d(k) - w(k)x(k)]$ odpowiada błędowi neuronu

Perceptron:

Został zbudowany zgodnie z modelem podanym na wykładzie oraz wg. książki Stanisława Osowskiego "Sieci neuronowe do przetwarzania informacji".

Metody publiczne:

- GetResult: Przyjmuje dane wejściowe, zwraca wyliczoną wartość
- Learn: Zajmuje się nauką neuronu. Modyfikuje wagi.

Adaline:

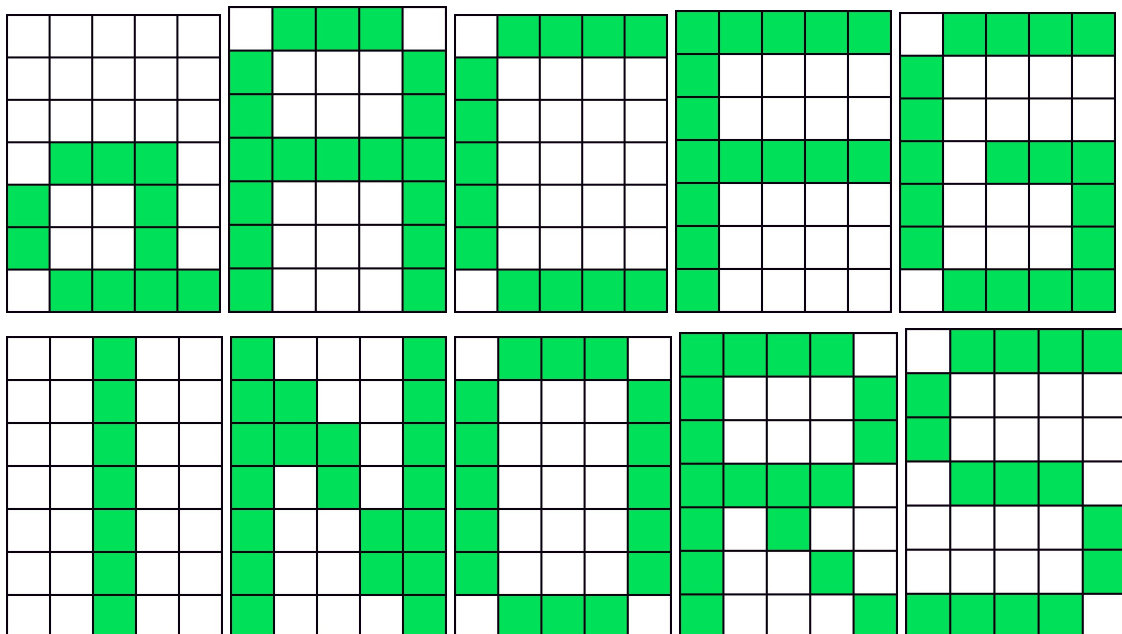
Budowę przypomina perceptron, z zasadniczą różnicą. Błąd jest obliczany przed właściwą funkcją aktywacji (wcześniej jest tylko funkcja liniowa). Dlatego błąd obliczamy na rzeczywistych wynikach. Został zbudowany na podstawie powyższej książki.

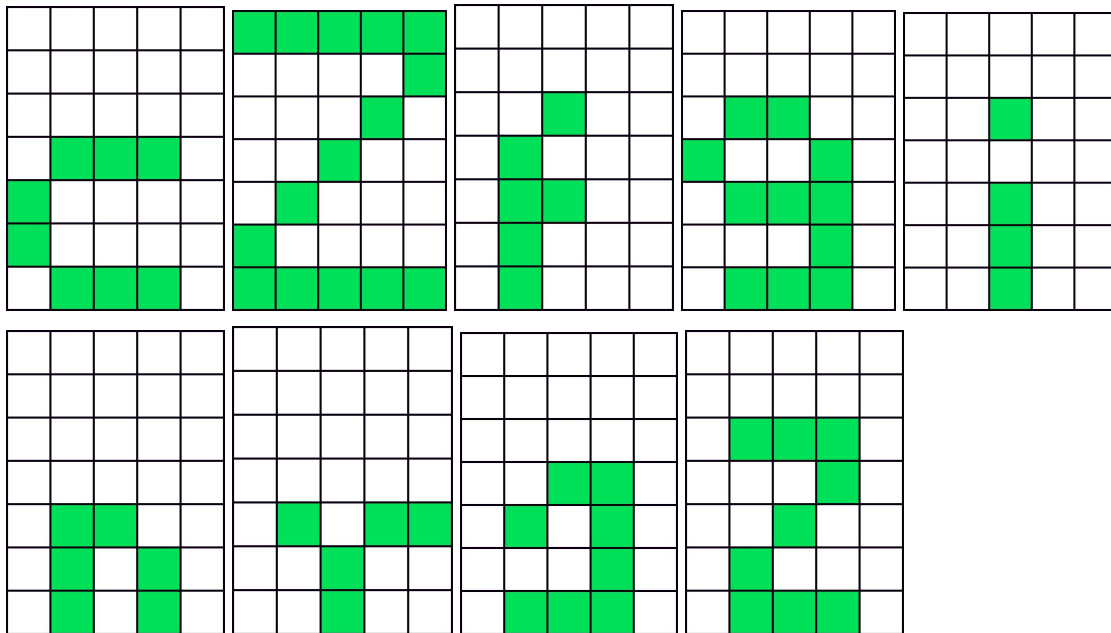
Metody publiczne:

- GetResult: Przyjmuje dane wejściowe, zwraca faktyczny wynik (przed funkcją aktywacji)
- Test: Zwraca dane, które przeszły przez funkcję aktywacji.
- Learn: Zajmuje się nauką neuronu. Modyfikuje wagi.

W obydwu przypadkach zastosowano progową funkcję aktywacji (albo litera jest mała, albo duża).

Zestaw danych testowych



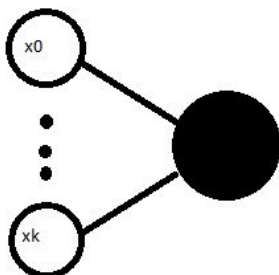


(przykładowe litery)

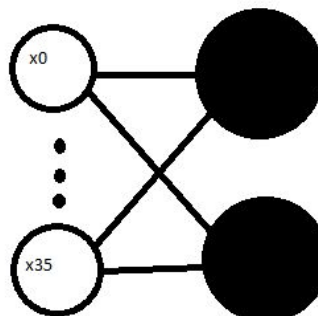
Testy

Testy przeprowadziłem dla dwóch przypadków. W wersji pierwszej korzystałem tylko z jednego neurona. W drugiej użyłem dwóch.

Wersja 1



Wersja 2



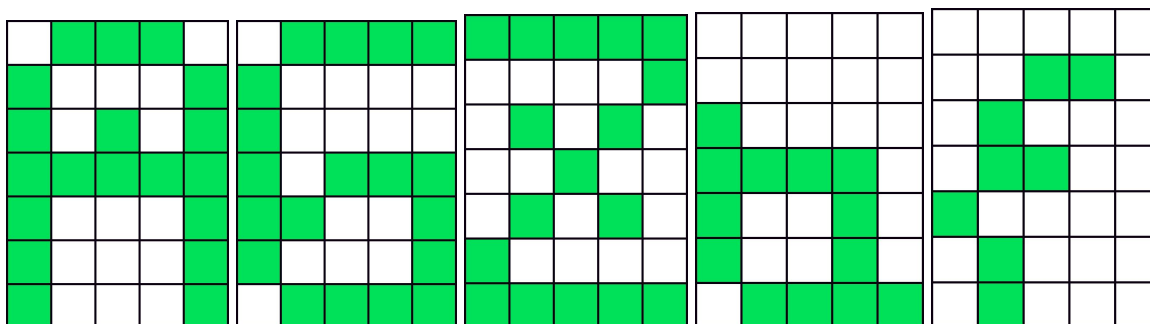
Wersja 1

Testy zostały przeprowadzane ze względu na 2 kryteria:

- współczynnik uczenia się
- ilość obszarów

Obszary były wyznaczane poprzez podzielenie tablicy na wycinki. Jeżeli w danym wycinku znajdował się zakolorowany piksel, wtedy wycinek zwraca wartość 1, jeżeli wszystkie piksele były białe, zwracał wartość 0.

Na koniec (po zakończeniu uczenia) neurony były poddawane jeszcze testowi, w którym litery były zaszumiane (niektóre piksele miały złe wartości). Ten zestaw danych nie był wykorzystywany podczas uczenia.



Najpierw postanowiłem przeprowadzić testy dla tylko jednego neuronu (próbuję rozpoznawać czy litera jest duża czy nie)

Wyniki:

Dla 1 fragmentu nie udało się ukończyć nauki pozytywnie, dla żadnego współczynnika uczenia.

(Dla wszystkich testów 50% trafionych).

Dla 9 fragmentów (3X i 3Y)

Nie udało się pozytywnie zakończyć nauki w żadnym przypadku.

Współczynnik uczenia 0.001.

Perceptron 1 epoka - 50% 51 epoka - 70% 52 epoka - 80% 72 epoka - 90% Wynik testu - 50% (wszystkie 0)	Adaline 1 epoka - 50% 13 epoka - 65% 14 epoka - 75% 15 epoka - 80% Wynik testu - 75%
--	---

Dla współczynnika 0.01

Perceptron 1 epoka - 75% 2 epoka - 80% 7 epoka - 95% 11 epoka - 90% Wynik testu - 50% (wszystkie 0)	Adaline 1 epoka - 50% 3 epoka - 60% 4 epoka - 75% 5 epoka - 80% Wynik testu - 75%
--	--

Dla współczynnika 0.01

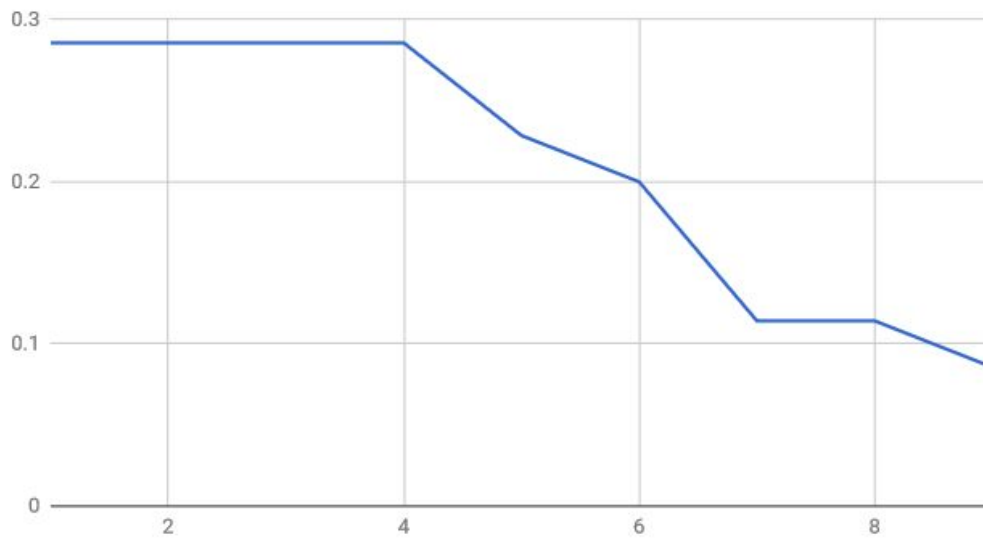
Perceptron 1 epoka - 75% 2 epoka - 90% 3 epoka - 85% 4 epoka - 90% Wynik testu - 50% (wszystkie 0)	Adaline 1 epoka - 90% 30 epoka - 95% Wynik testu - 75%
---	---

Dla 35 fragmentów (5X i 7Y)

Współczynnik uczenia 0.001

Perceptron

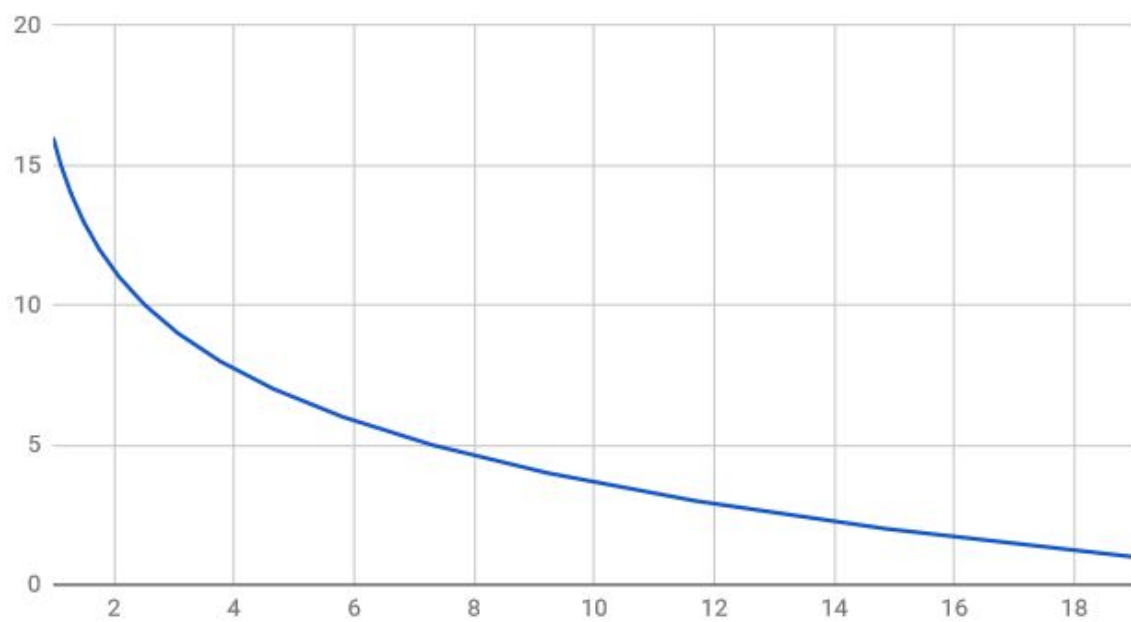
Wykres błędu od epok



Wynik testu - 85%

Adaline

Wykres błędu od epok

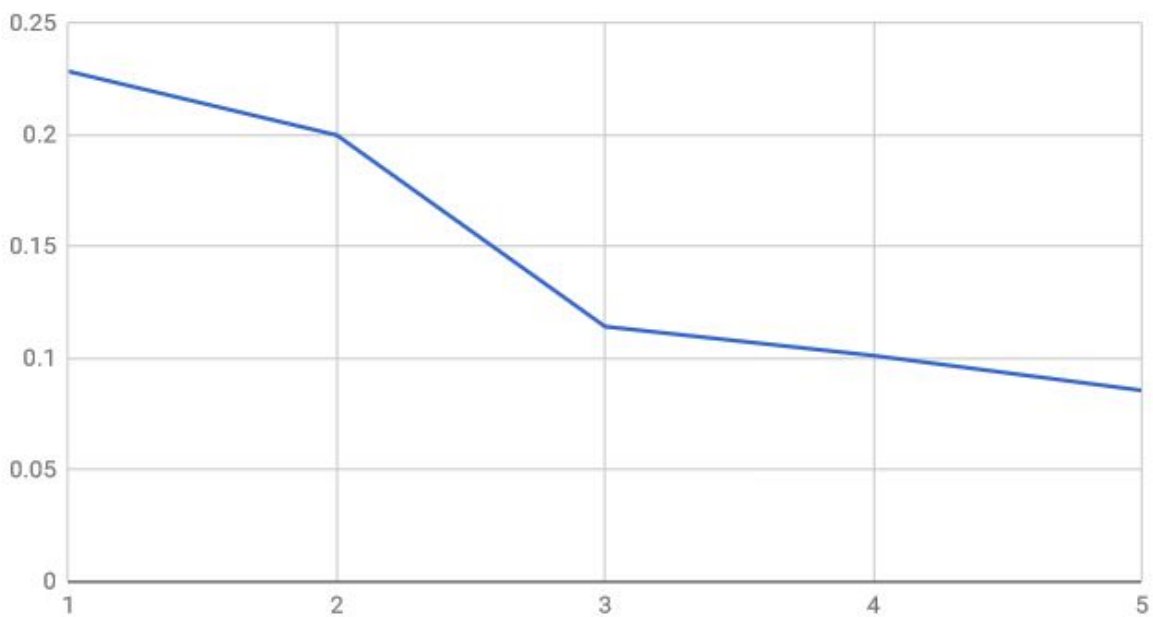


Wynik testu 80%

Współczynnik uczenia 0.01

Perceptron

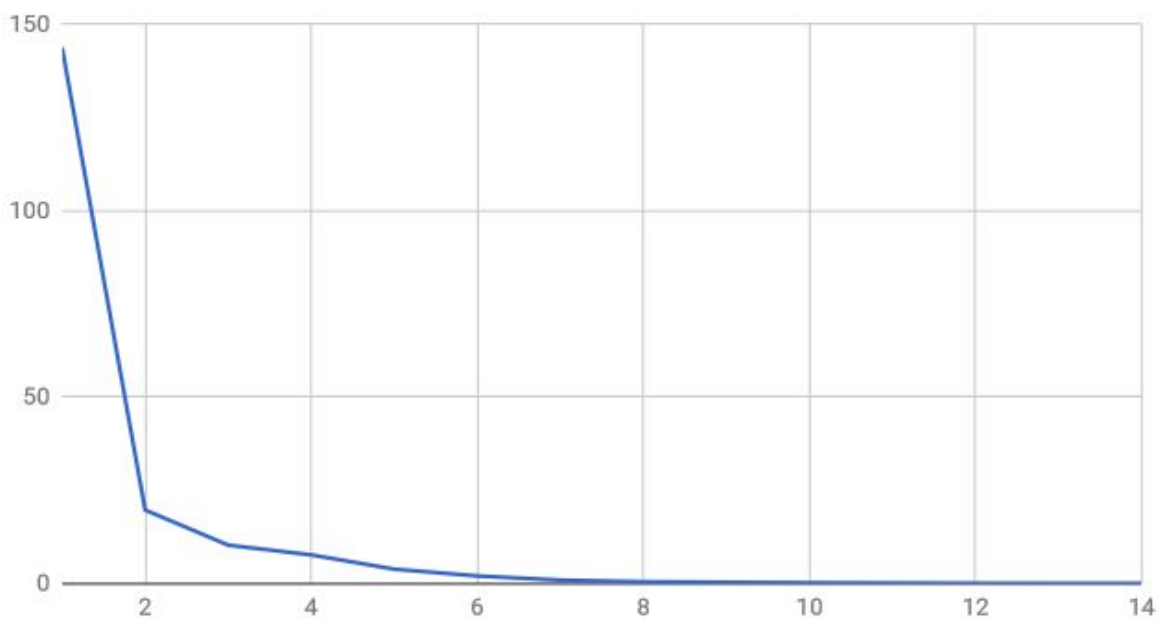
Wykres błędu od epok



Wynik Testu - 85%

Adaline

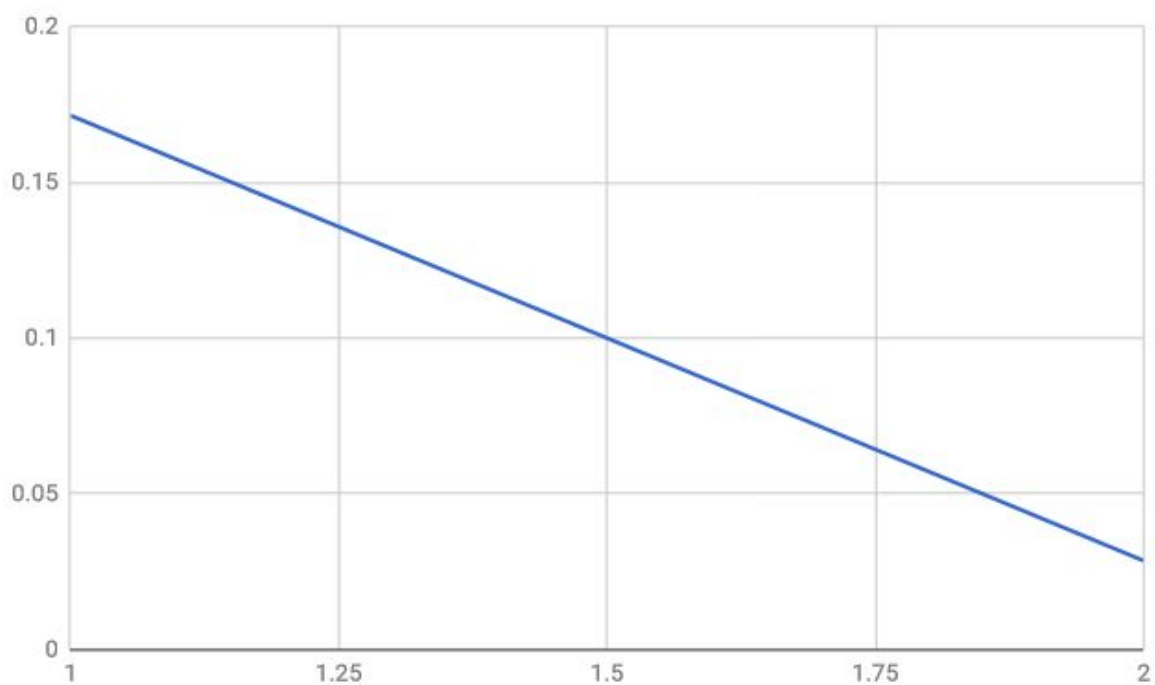
Wykres zależności wielkości błędu od epoki



Wynik Testu - 95%

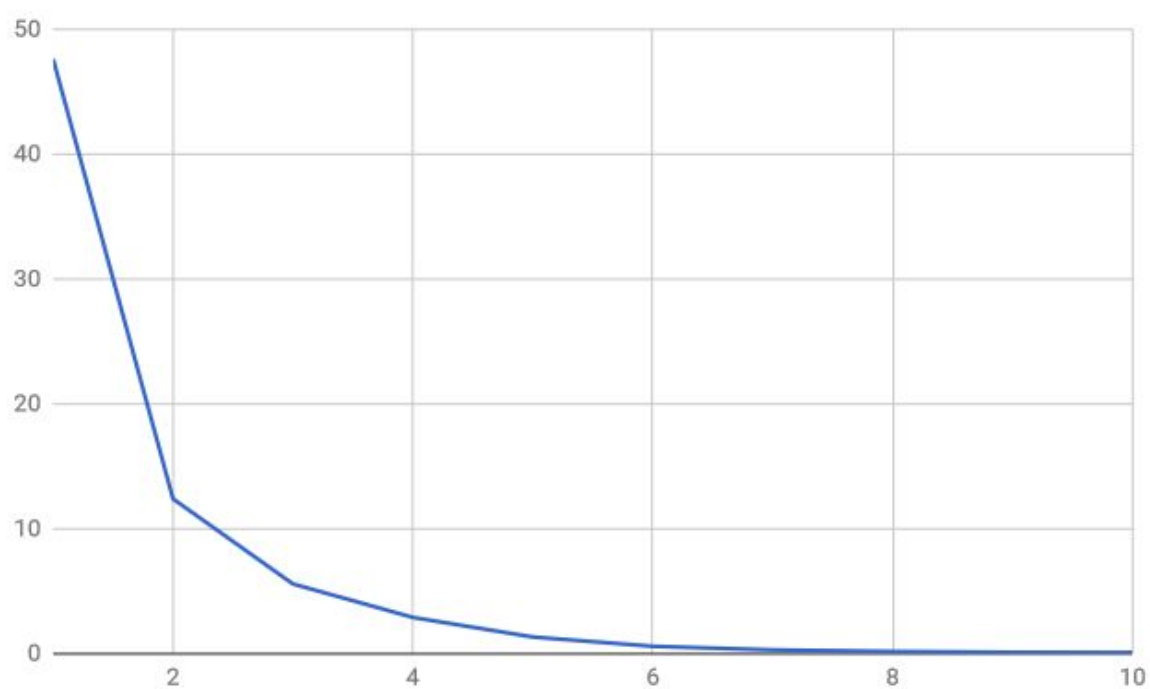
Współczynnik uczenia 0.1

Perceptron



Wynik Testu 90%

Adeline



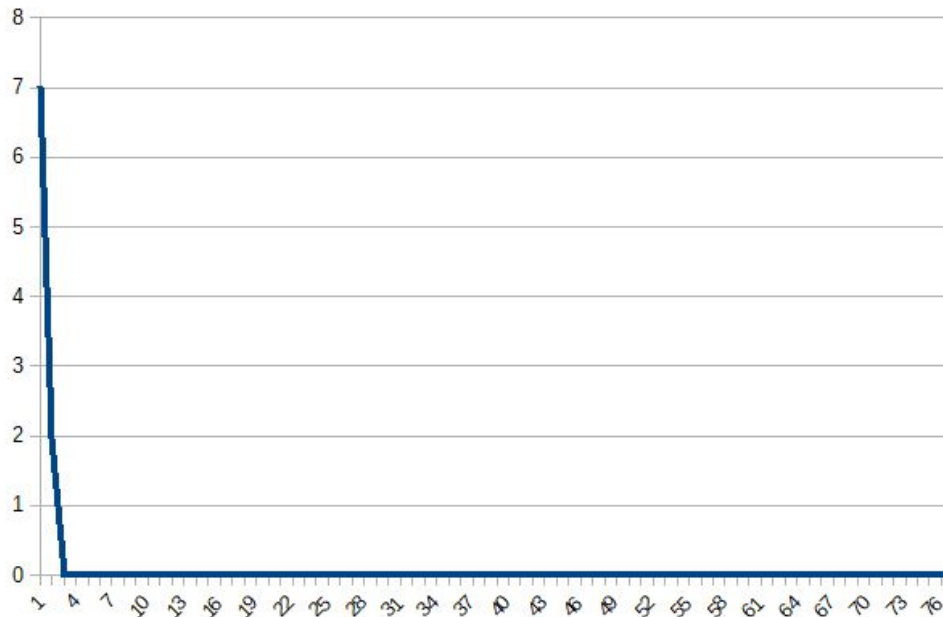
Wynik Testu 90%

Wersja 2

Następnie zmieniłem taktykę. Użyłem 2 perceptronów. Jeden uczył się rozpoznawać duże litery, drugi natomiast próbował rozpoznawać duże. Postanowiłem również nie dzielić obszaru na fragmenty ale wykorzystać wszystkie 35 wejść. To podejście okazało się dużo lepsze od poprzedniego.

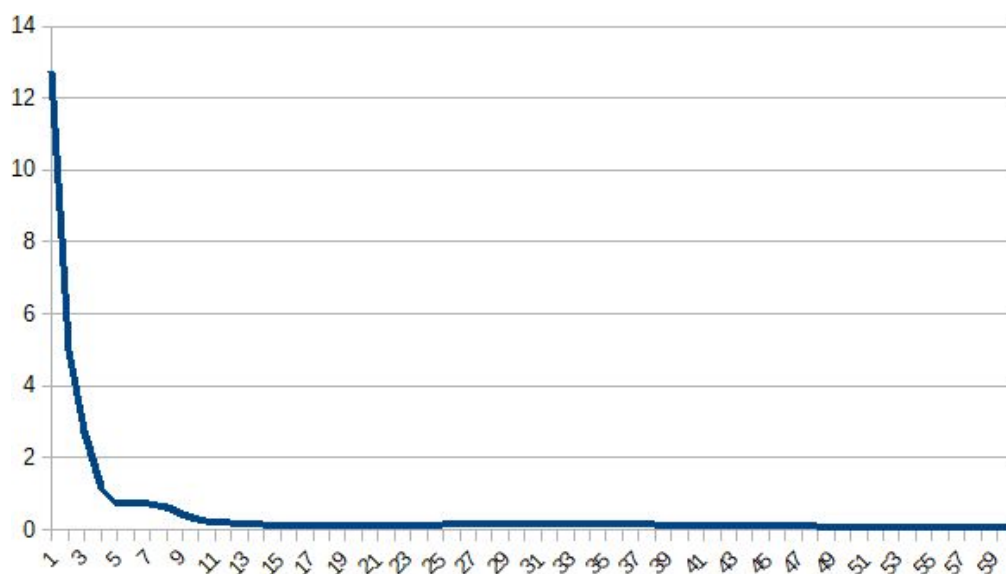
Współczynnik uczenia 0.1;

Perceptron:



Wynik Testu - 100%

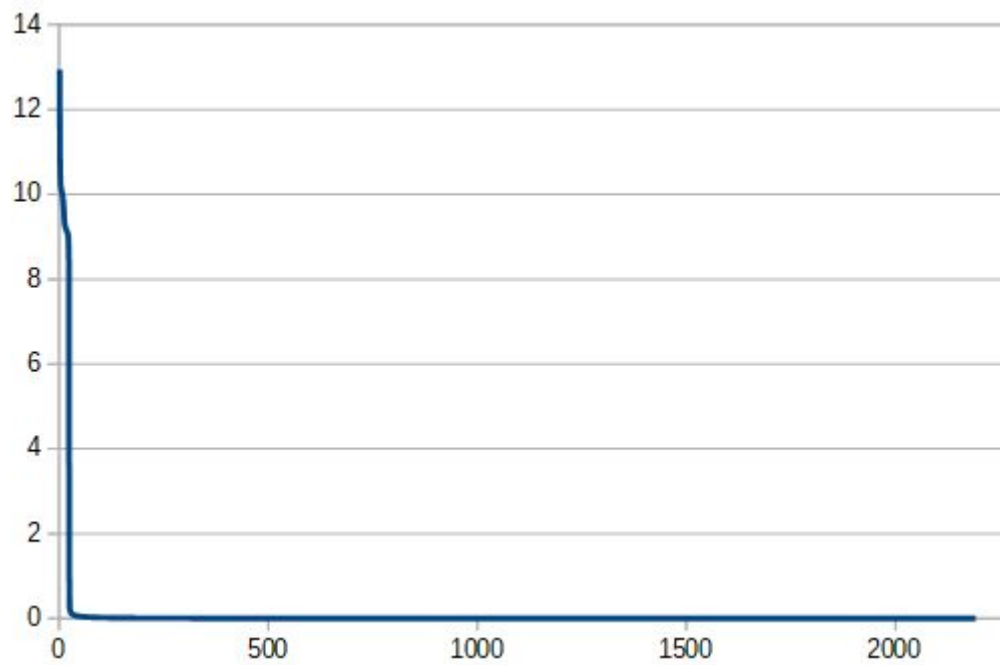
Adaline:



Wynik Testu - 95%

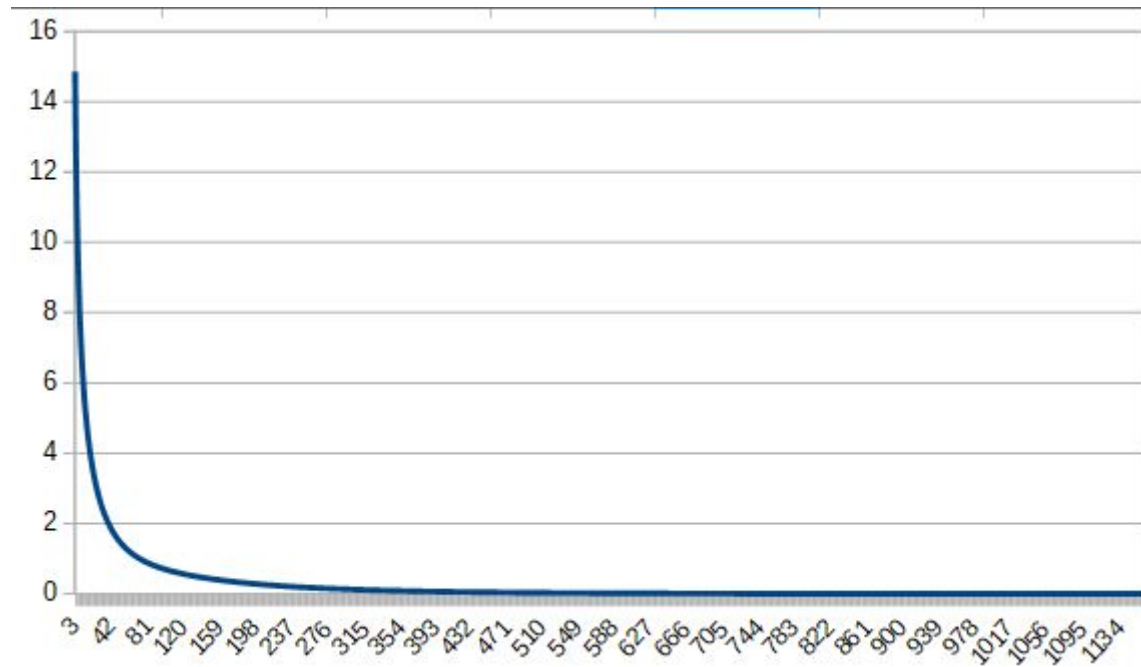
Współczynnik uczenia: 0.01

Perceptron:



Wynik Testu - 100%

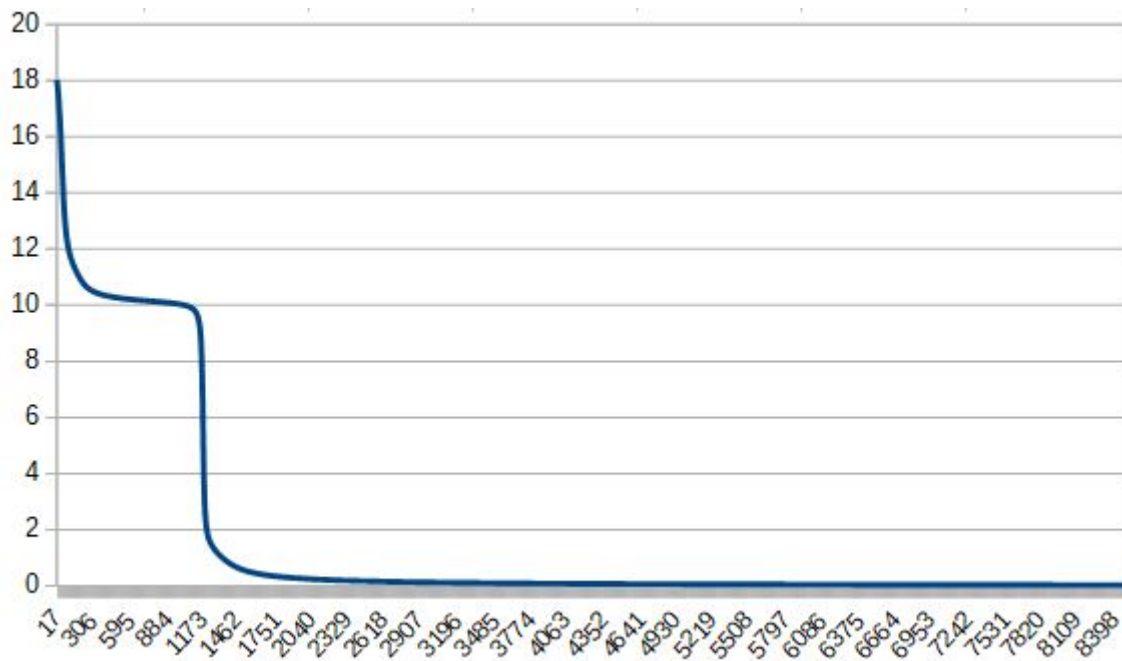
Adaline:



Wynik Testu - 100%

Współczynnik uczenia: 0.001

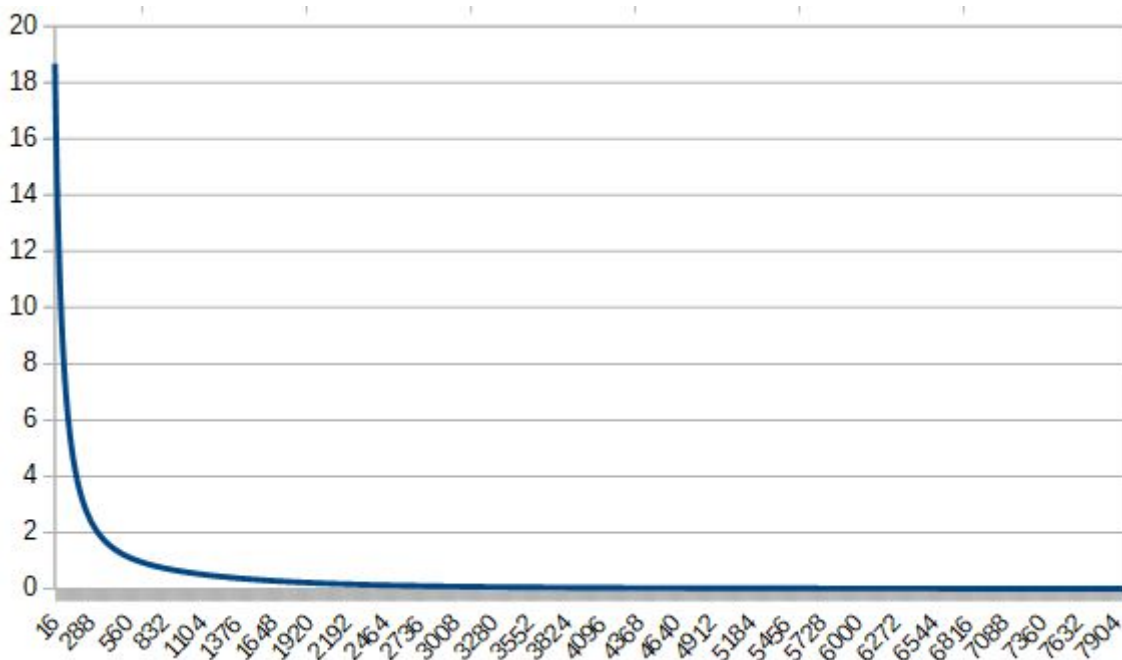
Perceptron:



(neurony skończyły naukę po 25423 epokach, nie uwzględniłem tego na wykresie, ponieważ arkusz kalkulacyjny nie był w stanie przyjąć takiej ilości danych)

Wynik testu - 95%

Adaline

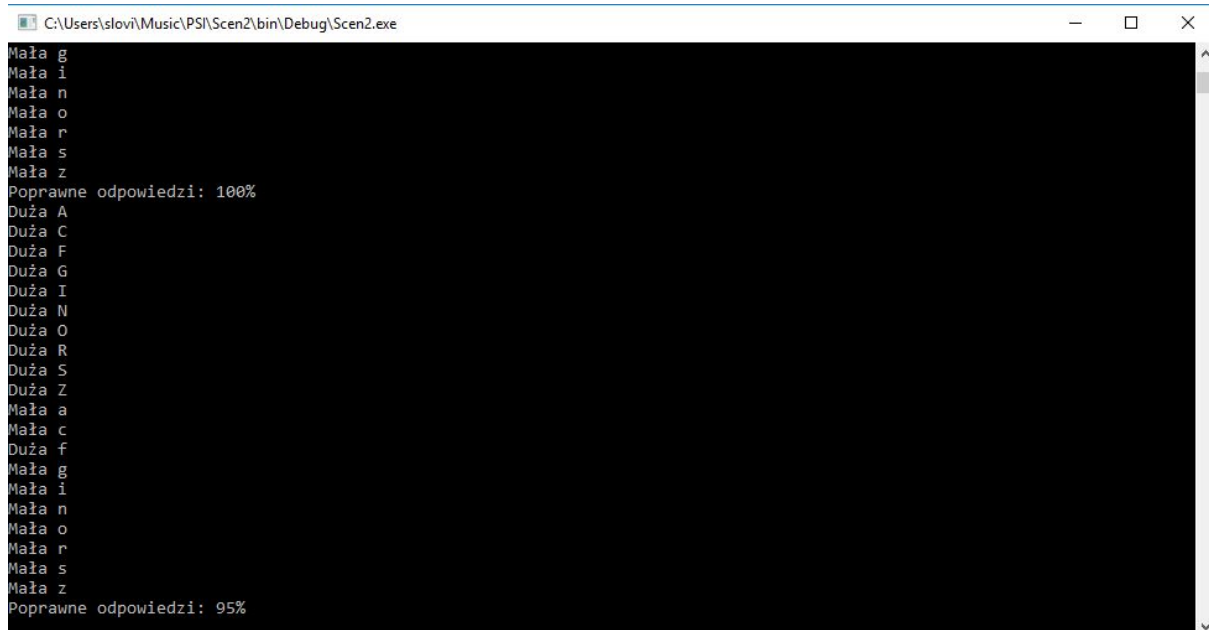


Wynik testu - 95%

Analiza wyników:

Jak można zauważyć jeden neuron nie był w stanie nauczyć się rozpoznawania liter. Sieć złożona z dwóch neuronów zawsze była w stanie nauczyć się rozpoznawać litery z zestawu

na którym się uczyła oraz z bardzo wysokim prawdopodobieństwem przewidywać mocno zaszumione litery. Jedyną literą z którą sieci zdawały się mieć czasem problem to mała litera 'f'. Czasami zaszumiona jej wersja była błędnie oznaczana jako litera duża.



```
C:\Users\slovi\Music\PSI\Scen2\bin\Debug\Scen2.exe
Mała g
Mała i
Mała n
Mała o
Mała r
Mała s
Mała z
Poprawne odpowiedzi: 100%
Duża A
Duża C
Duża F
Duża G
Duża I
Duża N
Duża O
Duża R
Duża S
Duża Z
Mała a
Mała c
Duża f
Mała g
Mała i
Mała n
Mała o
Mała r
Mała s
Mała z
Poprawne odpowiedzi: 95%
```

Można również zwrócić uwagę na fakt, że sieć złożona z neuronów Adaline uczyła się znacznie szybciej od drugiej wersji.

Wnioski:

- Złożone problemy mogą wymagać większej ilości neuronów w warstwie.
- Błędne przewidywanie zaszumionej małej litery 'f' mogło być spowodowane podobieństwem jej do dużej litery.
- Błąd ten można by rozwiązać poprzez uwzględnienie zaszumionej litery f podczas uczenia
- Na wyniki bardzo duży wpływ ma początkowy zestaw danych. Ważne jest aby danych było dużo i uwzględniały różne przypadki.
- Perceptron uczył się wolniej od Adaline. Po logach widać, że zwalniał szczególnie (o wiele bardziej niż adaline) w końcowych fazach uczenia.
- Najlepszym współczynnikiem uczenia był zdecydowanie 0.1. Mniejsze nie zwiększały dokładności, zaś wydłużały okres nauki.

```

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

namespace Scen2
{
    class Program
    {
        static void Main(string[] args)
        {
            double learningRate = 0.1;

            Console.WriteLine("Number of Width Fields: " + Letters.NumberOfFieldsX + " Number of Height Fields: " + Letters.NumberOfFieldsY);

            Console.WriteLine("-----Perceptron-----");
            PercetronTrainer trainer = new PercetronTrainer(2, learningRate);
            trainer.Train();
            Console.WriteLine("Testowanie nauki na danych poprawnych");
            trainer.Test(Letters.LettersData);
            Console.WriteLine("Testowanie nauki na danych zaszumionych");
            trainer.Test(Letters.CorrruptedLettersData);
            Console.ReadLine();

            AdalineTrainer aTrainer = new AdalineTrainer(learningRate);
            aTrainer.Train();
            aTrainer.Test(Letters.LettersData);
            aTrainer.Test(Letters.CorrruptedLettersData);
            Console.ReadLine();
        }
    }
}

```

```

namespace Scen2
{
    class PerceptronTrainer
    {
        private Perceptron[] layer;
        private double[] results;
        private int biasID;
        private double learningRate;
        private int Max = 100000000;

        public PerceptronTrainer(int numberOfPerceptrons, double learningRate)
        {
            this.learningRate = learningRate;
            biasID = Letters.NumberOfFields;
            layer = new Perceptron[numberOfPerceptrons];
            results = new double[numberOfPerceptrons];

            Random r = new Random();
            double[] weights = new double[Letters.NumberOfFields + 1];

            for(int i = 0; i < numberOfPerceptrons; i++)
            {
                for (int j = 0; j < weights.Length; j++)
                {
                    weights[j] = r.NextDouble();
                }
                layer[i] = new Perceptron(weights);
            }
        }

        public void Train()
        {
            int counter = 0;
            int[] input;
            int expected;
            double result;
            double mseError;
            do
            {
                mseError = 0.0;
                for (int i = 0; i < Letters.Expected.Length; i++)
                {
                    for(int j = 0; j < 2; j++)
                    {
                        input = Letters.GetLetter(i, Letters.LettersData);
                        expected = GetExpected(Letters.Expected[i], j);
                        layer[j].Learn(input, expected, learningRate);
                        result = layer[j].GetResult(input);
                        mseError += Math.Pow((expected - result), 2.0);
                    }
                }
                Console.WriteLine(mseError);
                counter++;
            } while (counter < Max && mseError > 0.001);
        }

        public void Test(int[,] dataSet)
        {
            double rate = Letters.Expected.Length;
            int[] input;
            int expected;
            for(int i = 0; i < Letters.Expected.Length; i++)
            {
                input = Letters.GetLetter(i, dataSet);
                expected = Letters.Expected[i];

                if (Act(input) != expected)
                    rate--;

                if(Act(input) == 1)
                {
                    Console.WriteLine("Duzia");
                }
                else
                {
                    Console.WriteLine("Mała");
                }
            }
            Console.WriteLine("Poprawne odpowiedzi: " + rate/20 * 100 + "%");
        }
    }
}

```

```

private int GetLetterSizeFromLayer(int[] input)
{
    for(int i = 0; i < results.Length; i++)
    {
        results[i] = layer[i].GetResult(input);
    }

    if(results[0] >= results[1])
    {
        return 0;
    }

    return 1;
}

private int GetExpected(int expected, int nr)
{
    if(nr == 0)
    {
        return expected;
    }
    else if(expected > 0)
    {
        return 0;
    }

    return 1;
}

public int Act(int[] input)
{
    if(layer[0].GetResult(input) > layer[1].GetResult(input))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
}

```

```

namespace Scen2
{
    class Perceptron
    {
        protected double[] weights;
        protected double bias = 1;
        protected double treshold = 0;

        public Perceptron(double[] weights)
        {
            this.weights = new double[weights.Length];
            Array.Copy(weights, this.weights, weights.Length);
        }

        public double GetResult(int[] input)
        {
            double sum = InputSummary(input);
            return PerceptronActivation(sum);
        }

        public void Learn(int[] input, double expected, double lr)
        {
            double result = GetResult(input);
            double delta = expected - result;

            double error = delta * Derive(InputSummary(input));

            for (int i = 0; i < input.Length; i++)
                weights[i] += error * lr * input[i];

            weights[Letters.NumberOfFields] += lr * error;
        }

        public void Printweights()
        {
            for (int i = 0; i < Letters.NumberOfFields; i++)
            {
                Console.WriteLine("Weight " + i + " : " + weights[i]);
            }

            Console.WriteLine("BIAS: " + bias + " WEIGHT: " + weights[Letters.NumberOfFields]);
            Console.ReadLine();
        }

        protected double InputSummary(int[] input)
        {
            double sum = 0;
            for (int i = 0; i < input.Length; i++)
            {
                sum += weights[i] * input[i];
            }

            return sum + weights[input.Length] * bias;
        }

        protected double PerceptronActivation(double sum)
        {
            return 1 / (1 + Math.Exp(-sum));
        }

        protected double Derive(double x)
        {
            return Math.Exp(-x) / Math.Pow((Math.Exp(-x) + 1), 2);
        }
    }
}

```



```

namespace Scen2
{
    class AdalineTrainer
    {
        private Adaline[] layer;
        private const int numberOfNeurons = 2;
        private double[] results;
        private int biasID;
        private double learningRate;
        private int Max = 100000000;

        public AdalineTrainer(double learningRate)
        {
            this.learningRate = learningRate;
            biasID = Letters.NumberOfFields;
            layer = new Adaline[numberOfNeurons];
            results = new double[numberOfNeurons];

            Random r = new Random();
            double[] weights = new double[Letters.NumberOfFields + 1];

            for (int i = 0; i < numberOfNeurons; i++)
            {
                for (int j = 0; j < weights.Length; j++)
                {
                    weights[j] = r.NextDouble();
                }
                layer[i] = new Adaline(weights);
            }
        }

        public void Train()
        {
            int counter = 0;
            int[] input;
            int expected;
            double result;
            double mseError;
            do
            {
                mseError = 0.0;
                for (int i = 0; i < Letters.Expected.Length; i++)
                {
                    for (int j = 0; j < 2; j++)
                    {
                        input = Letters.GetLetter(i, Letters.LettersData);
                        expected = GetExpected(Letters.Expected[i], j);
                        layer[j].Learn(input, expected, learningRate);
                        result = layer[j].GetResult(input);
                        mseError += Math.Pow((expected - result), 2.0);
                    }
                }
                Console.WriteLine(mseError);
                counter++;
            } while (counter < Max && mseError > 0.001);
            Console.WriteLine(counter);
        }
    }
}

```



```

public void Train()
{
    int counter = 0;
    int[] input;
    int expected;
    double result;
    double mseError;
    do
    {
        mseError = 0.0;
        for (int i = 0; i < Letters.Expected.Length; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                input = Letters.GetLetter(i, Letters.LettersData);
                expected = GetExpected(Letters.Expected[i], j);
                layer[j].Learn(input, expected, learningRate);
                result = layer[j].GetResult(input);
                mseError += Math.Pow((expected - result), 2.0);
            }
        }
        Console.WriteLine(mseError);
        counter++;
    } while (counter < Max && mseError > 0.001);
    Console.WriteLine(counter);
}

public void Test(int[, ,] dataSet)
{
    double rate = Letters.Expected.Length;
    int[] input;
    int expected;
    for (int i = 0; i < Letters.Expected.Length; i++)
    {
        input = Letters.GetLetter(i, dataSet);
        expected = Letters.Expected[i];

        if (Act(input) != expected)
            rate--;

        if (Act(input) == 1)
        {
            Console.Write("Duża ");
        }
        else
        {
            Console.Write("Mała ");
        }

        Console.WriteLine(Letters.Character[i]);
    }
    Console.WriteLine("Poprawne odpowiedzi: " + rate / 20 * 100 + "%");
}

private int GetExpected(int expected, int nr)
{
    if (nr == 0)
    {
        if (expected == 0)
            return -1;
        return expected;
    }
    else if (expected > 0)
    {
        return -1;
    }

    return 1;
}

```

```
public int Act(int[] input)
{
    if (layer[0].GetResult(input) > layer[1].GetResult(input))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```