Scenariusz 2

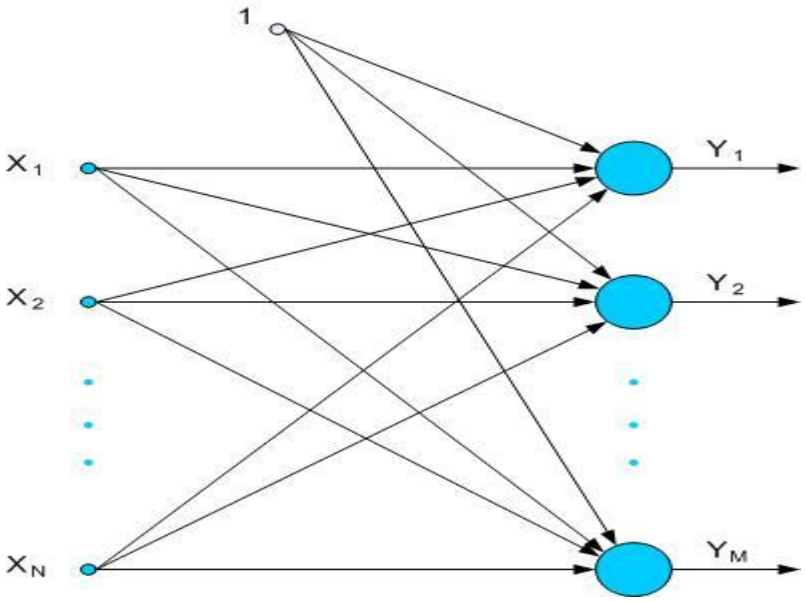
Jan Dudek

Temat ćwiczenia: Budowa i działanie sieci jednowarstwowej

Cel ćwiczenia: Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

1) Syntetyczny opis budowy oraz wykorzystania sieci i algorytmów uczenia

   Dane wejściowe dla sieci stanowi zestaw 22 (11 dużych i 11 małych) liter zapisanych w postaci tablicy 4x5 pikseli. Do każdej z liter przyporządkowana jest informacja czy jest ona mała czy duża.
   Sieć złożona jest z 4 neuronów w jednej warstwie. Jej zadaniem jest zwrócenie 4 wartości binarnych. Zwrócone wartości są porównywane do szablonu (1,0,1,0) dla dużej litery i przeciwnie dla małej. Sieć zorganizowana jest jak na grafice poniżej.



   Cały program składa się z dwóch sieci. Jedna z nich wykorzystuje algorytm Adaline, druga Delta Rule. Rózni je algorytm uczenia.

   a) Adaline:
      Algorytm uczenia prezentuje przypisanie: W[i+1] = W[i] + N(o - y)x
      Gdzie:
      W – waga konkretnego wejścia
      N – współczynnik uczenia
      o – wartość otrzymana
      y – wartość oczekiwana
      x – wartość na konkretnym wejściu
      Funkcją aktywacji jest funkcja sigmoidalna unipolarna:
      $$\frac{1}{1+e^{-x}}$$
   b) Delta Rule:
      Algorytm uczenia prezentuje przypisanie: W[i+1] = W[i] + N(o - y)*g(h)'*x
      Gdzie:

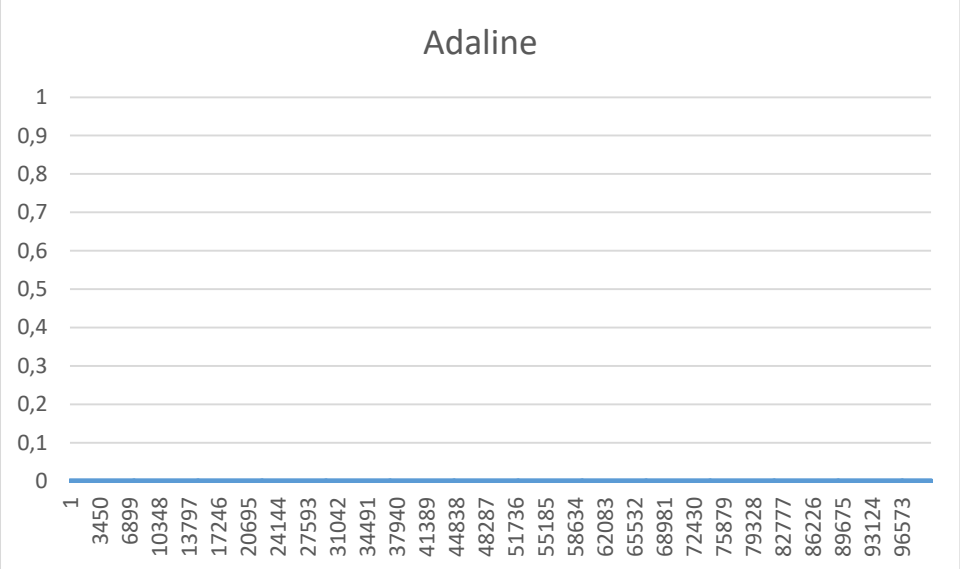
W – waga konkretnego wejścia
N – współczynnik uczenia
o – wartość otrzymana
y – wartość oczekiwana
x – wartość na konkretnym wejściu
g’ – pochodna funkcjiaktywacji
h – suma wejść przemnożonych przez ich wagi
Funkcją aktywacji jest funkcja sigmoidalna unipolarna:

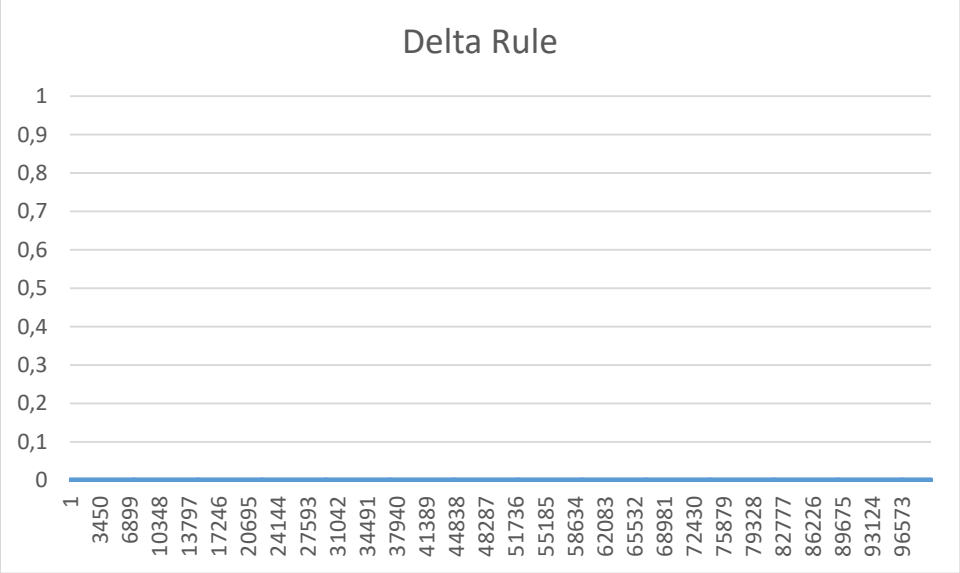
$$\frac{1}{1+e^{-x}}$$

Pochodna funkcji aktywacji prezentuje się następująco:

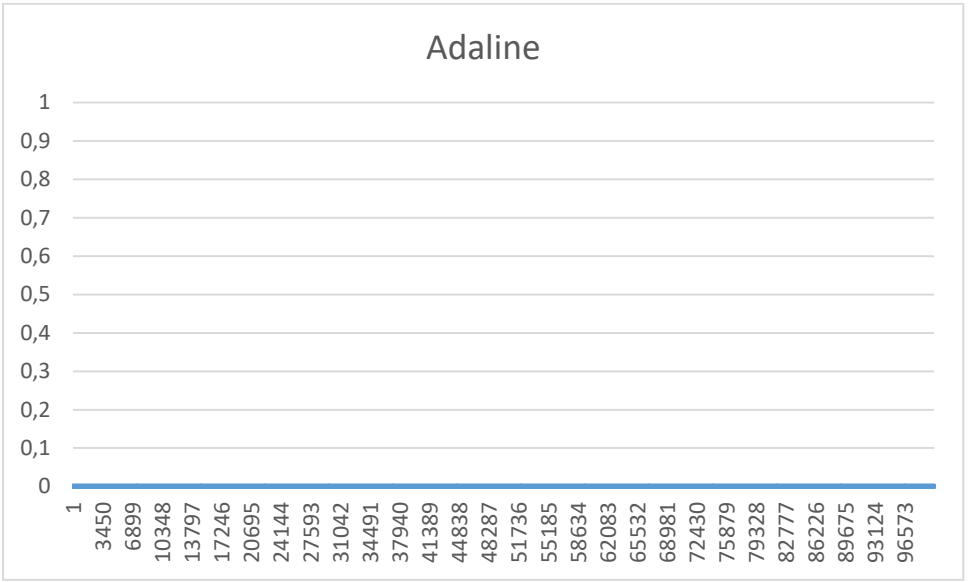
$$\frac{e^{x}}{(e^{x}+1)^{2}}$$

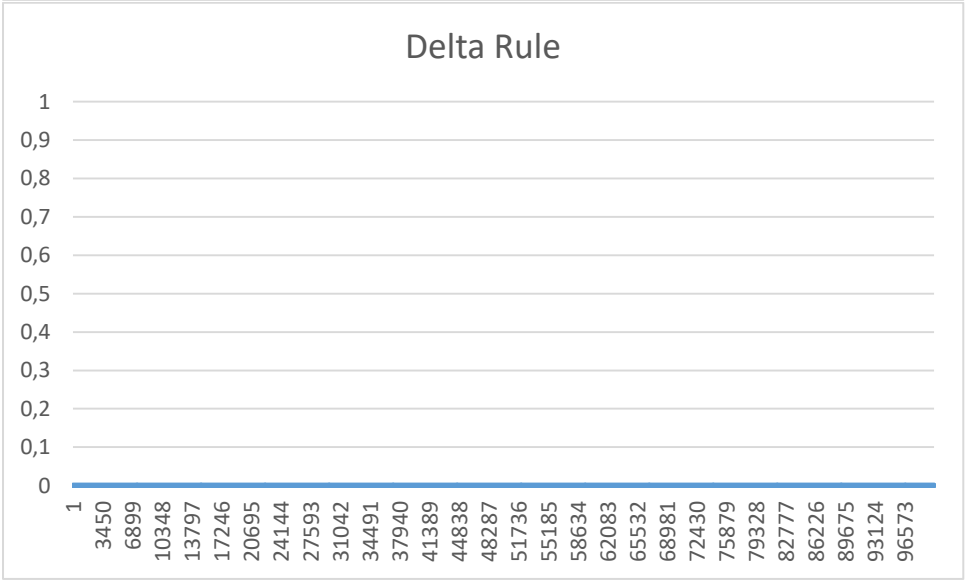
2) Zestawienie otrzymanych wyników
Poniższe wykresy prezentują ilość poprawnych odpowiedzi sieci w kolejnych epokach:
Dla współczynnika uczenia = 0,001



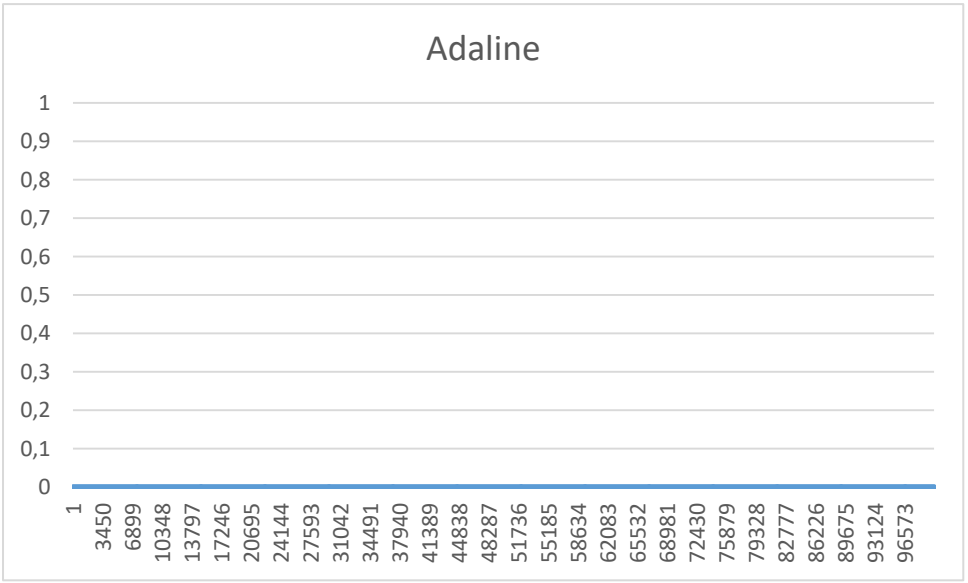


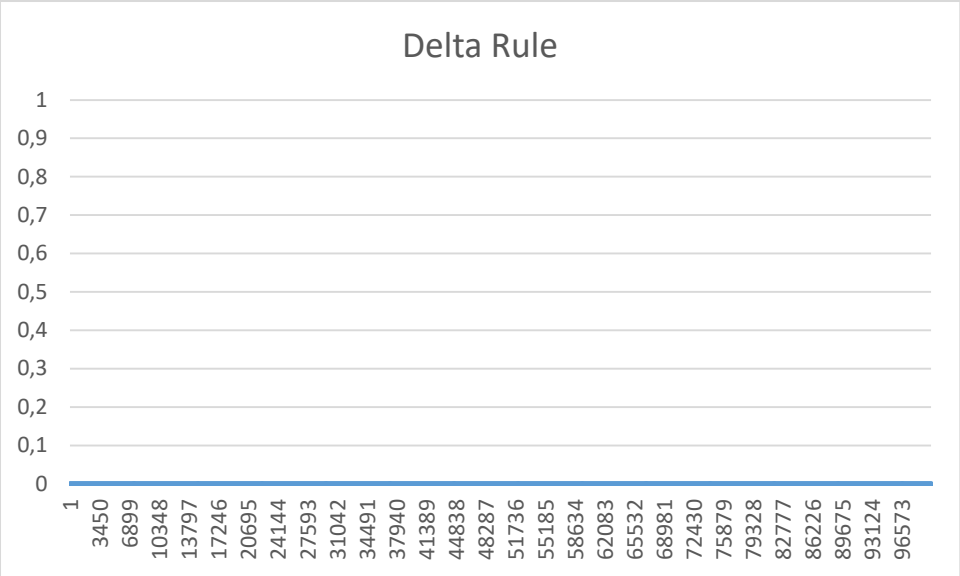
Dla współczynnika uczenia = 0,01:





Dla współczynników uczenia = 0,1:





Wyniki testów dla 1000000 (miliona) epok nie mogą być zaprezentowane z powodów technicznych (Word nie odpowiada po dodaniu wykresu). Takie testy przyniosły jednak wyniki identyczne do powyższych.

3) Analizę i dyskusję błędów uczenia i testowania opracowanych sieci w zależności od wartości współczynnika uczenia oraz wybranego algorytmu:
Jak widać na powyższych przykładach zarówno sieć Adaline jak i DeltaRule nie osiągnęły żadnej sprawności podczas uczenia.

4) Wnioski:
Niepowodzenie w uczeniu sieci wynika najprawdopodobniej z kształtu wykorzystanej czcionki. Jeśli nie jest możliwe wyznaczenie separowalnych przedziałów dziedziny funkcji (zadania) w liczbie neuronów to nie jest możliwe nauczenie sieci rozwiązywania tego zadania. Niepoprawne działanie sieci może też wynikać z niedostosowania funkcji aktywacji do zadania. Możliwe, że odpowiednia translacja funkcji aktywacji mogłoby umożliwić nauczenie sieci.

5) Listing kodu:

```
6) using System;
7) using System.Collections.Generic;
8) using System.IO;
9) using System.Linq;
10) using System.Text;
11) using System.Threading.Tasks;
12)
13) namespace Scenariusz_2
14) {
15)     class Program
16)     {
17)
18)         public static Boolean CompareArray(int[] arr1, int[] arr2)
19)         {
20)             for(int i = 0; i < arr1.Length; i++)
21)             {
22)                 if(arr1[i] != arr2[i])
23)                 {
```

```
24)                     return false;
25)                 }
26)             }
27)             return true;
28)         }
29)
30)         public static int WyrownajWynik(double x)
31)         {
32)             if (x > 0.5)
33)             {
34)                 return 1;
35)             }
36)             else
37)             {
38)                 return 0;
39)             }
40)         }
41)
42)         class InputData
43)         {
44)             public List<int> elements;
45)             public bool isUpper;
46)
47)             public InputData(List<int> list, bool result)
48)             {
49)                 elements = list;
50)                 isUpper = result;
51)             }
52)
53)             public InputData(Boolean isUpper)
54)             {
55)                 elements = new List<int>();
56)                 this.isUpper = isUpper;
57)             }
58)         }
59)
60)
61)         #region Czytanie danych z plików
62)         static void PrepareData(List<InputData> data)
63)         {
64)             int i = 0;
65)             String line;
66)             try
67)             {
68)                 using (StreamReader sr = new StreamReader("upperCase.txt"))
69)                 {
70)                     while (sr.Peek() >= 0)//W tej pętli czytam kolejne linie
    znaków
71)                     {
72)                         data.Add(new InputData(true));
73)
74)                         line = sr.ReadLine();
75)
76)                         for(int j = 0; j < line.Length; j++)//W tej pętli
    czytam pojedyncze znaki
77)                         {
78)
    data[i].elements.Add((int)Char.GetNumericValue(line[j]));
79)                         }
80)                         i++;
81)                     }
82)                 }
```

```
83)             }
84)             catch (Exception e)
85)             {
86)                 Console.WriteLine("The file could not be read:");
87)                 Console.WriteLine(e.Message);
88)             }
89)             try
90)             {
91)                 //i = 20; //tutaj jest coś nie OK
92)                 using (StreamReader sr = new StreamReader("lowerCase.txt"))
93)                 {
94)                     while (sr.Peek() >= 0)//W tej pętli czytam kolejne linie
    znaków
95)                     {
96)                         data.Add(new InputData(false));
97)                         line = sr.ReadLine();
98)
99)                         for (int j = 0; j < line.Length; j++)//W tej pętli
    czytam pojedyncze znaki
100)                                {
101)
    data[i].elements.Add((int)Char.GetNumericValue(line[j]));
102)                                }
103)                                i++;
104)                            }
105)                        }
106)                    }
107)                    catch (Exception e)
108)                    {
109)                        Console.WriteLine("The file could not be read:");
110)                        Console.WriteLine(e.Message);
111)                    }
112)                }
113)                #endregion
114)
115)
116)
117)
118)                ///
    ////////////////////////////////ADALINE////////////////////////////////////
    //////
119)
120)                class NeuronAdaline
121)                {
122)                    public double wynikNeuronu;
123)                    public InputData dane;
124)                    public List<double> W;
125)                    public double N;
126)
127)                    public NeuronAdaline()
128)                    {
129)                        W = new List<double>();
130)                        for(int i = 0; i < 20; i++)
131)                        {
132)                            Random random = new Random();
133)                            W.Add(random.NextDouble());
134)                        }
135)                    }
136)
137)
138)                    double ActivationBlock(double x)
139)                    {
```

```
140)                     /* binarna
141)                     if (x > 1)
142)                     {
143)                         return 1;
144)                     }
145)                     else
146)                     {
147)                         return -1;
148)                     }
149)                     */
150)                     return (1.0/(1.0+Math.Exp(-x)));
151)                 }
152)
153)                 double SumBlock()
154)                 {
155)                     double suma = 0;
156)                     for(int i=0;i<dane.elements.Count;i++)
157)                     {
158)                         suma += dane.elements[i]* W[i];
159)                     }
160)                     return suma;
161)                 }
162)
163)                 public void PoprawWagi(int d)
164)                 {
165)                     for(int i=0;i<W.Count;i++)
166)                     {
167)                         // waga = stara waga + (wspolczynnik uczenia X
    wartosc na wejsciu neuronu (?) X (błąd sredniokwadratowy))
168)                         //
169)                         /*double blad = (d - wynikNeuronu) * (d -
    wynikNeuronu);
170)                         if (blad != 0)
171)                             blad = blad / 2;
172)                        */
173)                         W[i] += N * (d-wynikNeuronu) * dane.elements[i];
174)
175)
176)                     }
177)
178)                 }
179)
180)
181)
182)                 public double Licz()
183)                 {
184)                     //wykonaj obliczenia i zwróć wynik
185)                     double suma = SumBlock();
186)                     //Console.WriteLine("SUMA: "+suma);
187)                     double wynik = ActivationBlock(suma);
188)                     wynikNeuronu = wynik;
189)                     return wynik;
190)                 }
191)             }
192)
193)
194)             /// ////////////////////////////////////DELTA
    RULE//////////////////////////////////////////////
195)
196)
197)             class NeuronDelta
198)             {
```

```
199)                    public double wynikNeuronu;
200)                    public InputData dane;
201)                    public List<double> W;
202)                    public double N;
203)
204)                    public NeuronDelta()
205)                    {
206)                        W = new List<double>();
207)                        for (int i = 0; i < 20; i++)
208)                        {
209)                            Random random = new Random();
210)                            W.Add(random.NextDouble());
211)                        }
212)                    }
213)
214)
215)                    double ActivationBlock(double x)
216)                    {
217)
218)                        return (1.0 / (1.0 + Math.Exp(-x)));
219)                    }
220)
221)                    double ActivationDerivative(double x)
222)                    {
223)
224)                        return (Math.Exp(x) / (Math.Exp(x) + 1) * (Math.Exp(x) +
    1));
225)                    }
226)
227)                    double SumBlock()
228)                    {
229)                        double suma = 0;
230)                        for (int i = 0; i < dane.elements.Count; i++)
231)                        {
232)                            suma += dane.elements[i] * W[i];
233)                        }
234)                        return suma;
235)                    }
236)
237)                    public void PoprawWagi(int d)
238)                    {
239)                        for (int i = 0; i < W.Count; i++)
240)                        {
241)                            // waga = stara waga + (wspolczynnik uczenia X
    wartosc na wejsciu neuronu (?) X (błąd sredniokwadratowy))
242)                            //
243)                            /*double blad = (d - wynikNeuronu) * (d -
    wynikNeuronu);
244)                            if (blad != 0)
245)                                blad = blad / 2;
246)                           */
247)                            W[i] += N * (d -
    wynikNeuronu)*ActivationDerivative(wynikNeuronu) * dane.elements[i];
248)
249)
250)                        }
251)
252)                    }
253)
254)
255)
256)                    public double Licz()
```

```
257)                 {
258)                     //wykonaj obliczenia i zwróć wynik
259)                     double suma = SumBlock();
260)                     //Console.WriteLine("SUMA: "+suma);
261)                     double wynik = ActivationBlock(suma);
262)                     wynikNeuronu = wynik;
263)                     return wynik;
264)                 }
265)             }
266)
267)
268)
269)             //
    ////////////////////////////////////////////////////////////////////////////
    //////////////////MAIN
270)
271)             static void Main(string[] args)
272)             {
273)                 List<InputData> inputDataList = new List<InputData>();
274)                 PrepareData(inputDataList);
275)
276)                 // Adaline
277)                 {
278)                     List<NeuronAdaline> AdaList = new List<NeuronAdaline>();
279)                     AdaList.Add(new NeuronAdaline());
280)                     AdaList.Add(new NeuronAdaline());
281)                     AdaList.Add(new NeuronAdaline());
282)                     AdaList.Add(new NeuronAdaline());
283)                     for (int i = 0; i < AdaList.Count; i++)
284)                     {
285)                         AdaList[i].N = 0.1;
286)                     }
287)                     int[] upperCase = { 1, 0, 1, 0 };
288)                     int[] lowerCase = { 0, 1, 0, 1 };
289)                     int[] resultCase = { 0, 0, 0, 0 };
290)
291)                     int iloscEpok = 100000;
292)                     string[] wyniki = new string[iloscEpok];
293)                     int iloscPoprawnychOdpowiedzi = 0;
294)                     int iloscBlednychOdpowiedzi = 0;
295)                     double wynikEpoki = 0;
296)                     Boolean isCorrect = true;
297)
298)
299)                     for (int x = 1; x < iloscEpok; x++)
300)                     {
301)                         iloscPoprawnychOdpowiedzi = 0;
302)                         iloscBlednychOdpowiedzi = 0;
303)                         wynikEpoki = 0;
304)                         //Wszystkie obiegi pętli to w sumie jedna epoka:
305)                         for (int i = 0; i < inputDataList.Count; i++)
306)                         {
307)                             isCorrect = true;
308)                             for (int j = 0; j < AdaList.Count; j++)
309)                             {
310)                                 AdaList[j].dane = inputDataList[i];
311)                                 resultCase[j] =
    WyrownajWynik(AdaList[j].Licz());
312)
313)
314)                                 //sprawdz czy wynik jest poprawny:
315)                                 if (inputDataList[i].isUpper)
```

```
316)                                    {
317)                                        if (CompareArray(resultCase, upperCase))
318)                                        {
319)                                            //Console.WriteLine("TRUE");
320)                                            iloscPoprawnychOdpowiedzi++;
321)                                        }
322)                                        else
323)                                        {
324)                                            //Console.WriteLine("False upper");
325)                                            iloscBlednychOdpowiedzi++;
326)                                            isCorrect = false;
327)                                        }
328)                                    }
329)                                    else if (!inputDataList[i].isUpper)
330)                                    {
331)                                        if (CompareArray(resultCase, lowerCase))
332)                                        {
333)                                            // Console.WriteLine("TRUE");
334)                                            iloscPoprawnychOdpowiedzi++;
335)                                        }
336)                                        else
337)                                        {
338)                                            //Console.WriteLine("False lower");
339)                                            iloscBlednychOdpowiedzi++;
340)                                            isCorrect = false;
341)                                        }
342)                                    }
343)                                    else
344)                                    {
345)                                        // Console.WriteLine("False else");
346)                                        isCorrect = false;
347)                                        iloscBlednychOdpowiedzi++;
348)                                    }
349)                                }
350)                                //popraw wagi
351)                                if (!isCorrect)
352)                                {
353)                                    for (int j = 0; j < AdaList.Count; j++)
354)                                    {
355)                                        if (AdaList[j].dane.isUpper)
356)                                        {
357)                                            AdaList[j].PoprawWagi(upperCase[j]);
358)                                        }
359)                                        else
360)                                        {
361)                                            AdaList[j].PoprawWagi(lowerCase[j]);
362)                                        }
363)                                    }
364)                                }
365)
366)                                //Console.WriteLine("Wyniki: " + resultCase[0] +
    resultCase[1] + resultCase[2] + resultCase[3]);
367)                                //Console.WriteLine(AdaList[0].W[0]);
368)                            }
369)                            double wynik = (iloscPoprawnychOdpowiedzi / 88) *
    100;
370)                            //Console.WriteLine("Koniec Epoki nr: "+x+"\t =
    "+wynik);
371)                            wyniki[x] = wynik.ToString();
372)                            wynik = 0;
373)                        }
374)                        File.WriteAllLines("outputAdaline.txt", wyniki);
```

```
375)                     Console.WriteLine("KoniecAdaLine");
376)                 }
377)
378)                 // Delta Rule
379)                 {
380)                     List<NeuronDelta> DeltaList = new List<NeuronDelta>();
381)                     DeltaList.Add(new NeuronDelta());
382)                     DeltaList.Add(new NeuronDelta());
383)                     DeltaList.Add(new NeuronDelta());
384)                     DeltaList.Add(new NeuronDelta());
385)                     for (int i = 0; i < DeltaList.Count; i++)
386)                     {
387)                         DeltaList[i].N = 0.1;
388)                     }
389)                     int[] upperCase = { 1, 0, 1, 0 };
390)                     int[] lowerCase = { 0, 1, 0, 1 };
391)                     int[] resultCase = { 0, 0, 0, 0 };
392)
393)                     int iloscEpok = 100000;
394)                     string[] wyniki = new string[iloscEpok];
395)                     int iloscPoprawnychOdpowiedzi = 0;
396)                     int iloscBlednychOdpowiedzi = 0;
397)                     double wynikEpoki = 0;
398)                     Boolean isCorrect = true;
399)
400)
401)                     for (int x = 1; x < iloscEpok; x++)
402)                     {
403)                         iloscPoprawnychOdpowiedzi = 0;
404)                         iloscBlednychOdpowiedzi = 0;
405)                         wynikEpoki = 0;
406)                         //Wszystkie obiegi pętli to w sumie jedna epoka:
407)                         for (int i = 0; i < inputDataList.Count; i++)
408)                         {
409)                             isCorrect = true;
410)                             for (int j = 0; j < DeltaList.Count; j++)
411)                             {
412)                                 DeltaList[j].dane = inputDataList[i];
413)                                 resultCase[j] =
    WyrownajWynik(DeltaList[j].Licz());
414)
415)
416)                                 //sprawdz czy wynik jest poprawny:
417)                                 if (inputDataList[i].isUpper)
418)                                 {
419)                                     if (CompareArray(resultCase, upperCase))
420)                                     {
421)                                         //Console.WriteLine("TRUE");
422)                                         iloscPoprawnychOdpowiedzi++;
423)                                     }
424)                                     else
425)                                     {
426)                                         //Console.WriteLine("False upper");
427)                                         iloscBlednychOdpowiedzi++;
428)                                         isCorrect = false;
429)                                     }
430)                                 }
431)                                 else if (!inputDataList[i].isUpper)
432)                                 {
433)                                     if (CompareArray(resultCase, lowerCase))
434)                                     {
435)                                         // Console.WriteLine("TRUE");
```

```
                                        iloscPoprawnychOdpowiedzi++;
                                    }
                                    else
                                    {
                                        //Console.WriteLine("False lower");
                                        iloscBlednychOdpowiedzi++;
                                        isCorrect = false;
                                    }
                                }
                                else
                                {
                                    // Console.WriteLine("False else");
                                    isCorrect = false;
                                    iloscBlednychOdpowiedzi++;
                                }
                            }
                            //popraw wagi
                            if (!isCorrect)
                            {
                                for (int j = 0; j < DeltaList.Count; j++)
                                {
                                    if (DeltaList[j].dane.isUpper)
                                    {
DeltaList[j].PoprawWagi(upperCase[j]);
                                    }
                                    else
                                    {
DeltaList[j].PoprawWagi(lowerCase[j]);
                                    }
                                }
                            }

                            //Console.WriteLine("Wyniki: " + resultCase[0] +
resultCase[1] + resultCase[2] + resultCase[3]);
                            //Console.WriteLine(AdaList[0].W[0]);
                        }
                        double wynik = (iloscPoprawnychOdpowiedzi / 88) *
100;
                        //Console.WriteLine("Koniec Epoki nr: "+x+"\t =
"+wynik);
                        wyniki[x] = wynik.ToString();
                        wynik = 0;
                    }
                    File.WriteAllLines("outputDeltaRule.txt", wyniki);
                    Console.WriteLine("Koniec Delta Rule");
                }


                Console.ReadLine();
            }
        }
    }
```