Filip Bartman, grupa projektowa 1

Sprawozdanie nr 2

Temat ćwiczenia: Budowa i działanie sieci jednowarstwowej

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz nauka ich rozpoznawania wielkości liter

1. Zadanie do wykonania
2. Wygenerowanie danych uczących i testujących, zawierających 10 dużych i 10 małych liter dowolnie wybranego alfabetu w postaci dwuwymiarowej tablicy.
3. Przygotowanie (implementacja) dwóch jednowarstwowych sieci – każda wg innego algorytmu.
4. Uczenie sieci przy różnych współczynnikach uczenia.
5. Testowanie sieci
6. Syntetyczny opis budowy oraz wykorzystania sieci i algorytmów uczenia.

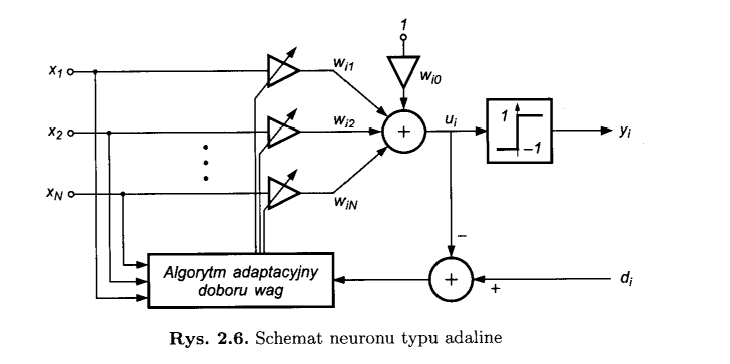
Sztuczna sieć neuronowa jest zbiorem prostych jednostek obliczeniowych, przetwarzających dane, komunikujących się ze sobą i pracujących równolegle.

W moim projekcie użyłem dwóch rodzajów sieci

– sieci Adaline ( opartej na algorytmie Widrowa – Hoffa)

– perceptronu sigmoidalnego

**Algorytm Adaline:**



- określenie losowych wag z zakresu (0,1) i ustawienie początkowo bias = 1,

- wyliczenie sumy wagowej sygnałów wejściowych,

- wyliczenie zmiennej *delta*, czyli różnicy między wartością oczekiwaną, a otrzymaną (wykorzystujemy tutaj wyliczoną wcześniej sumę – funkcja liniowa)

, gdzie:

di- wartość oczekiwana

u-wartość otrzymana

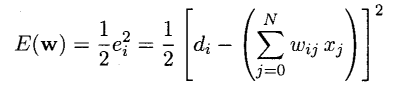
- bipolarnej funkcji aktywacji nie wykorzystujemy w procesie uczenia się,

- dyskretny sposób uaktualniania wag:



η – współczynnik uczenia się

- zdefiniowanie funkcji celu:

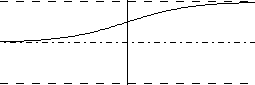


- proces uczenia neuronu Adaline polega na minimalizacji funkcji celu – uaktualniamy wagi aż wyliczona wartość wyjściowa jest odpowiednio małym przybliżeniem aktualnego wyniku, czyli aż error\_MSE (błąd średniokwadratory) będzie dążył do zera. Możemy tu przyjąć określoną, odpowiednio małą liczbę, będącą warunkiem zakończenia procesu uczenia.

- Funkcja aktywacji jest dopiero wykorzystywana przy określaniu wartości wyjściowej w procesie testowania neuronu adaline przy zadanych wartościach wejściowych.

**Neuron sigmoidalny**

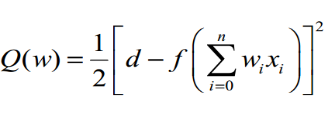
Neuron sigmoidalny posiada strukturę bardzo podobną do modelu McCullocha-Pitsa, moglibyśmy tu zastosować poprzedni schemat z jedną istotną różnicą, jest to funkcja aktywacji. O ile w Adaline zastosowałem funkcję progową unipolarną, o tyle w tym modelu użyłem funkcji sigmoidalnej.

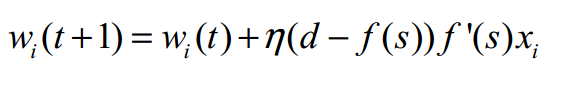


y=1/(1+exp(-beta\*z))

Sam wzór jest bardzo skomplikowany jednak jego pochodna jest bardzo prosta do wyznaczenia. Dzięki temu proces nauki jest bardzo prosty...  
We wzorze pojawia się współczynnik beta. Jego wartość określa 'ściśnięcie' wykresu. Im beta mniejsze tym wykres bardziej ściśnięty, przy beta rosnącym do nieskończoności wykres pokrywa się z wykresem funkcji progowej.

Błąd w tym modelu obliczałem z następującego wzoru:



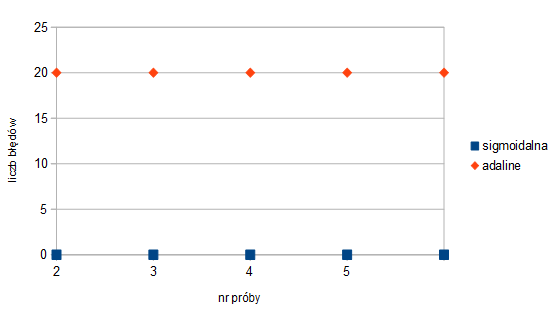
Uaktualnianie wag odbywa się wg tego wzoru:

1. Zestawienie otrzymanych wyników:

Danymi uczącymi perceptronu była tablica 20 liter, każda litera składała się z 35 wartości 0 lub 1.Przeprowadziłem po 5 prób programu dla współczynników uczenia 0,1 ; 0,2 ; 0,5 ; 0,05

Funkcja sigmoidalna:

Wykres od ilości błędnie wyznaczonych liter dla danych testowych :



1. Analiza wyników

* przy małym współczynniku uczenia dochodziło do większej liczby iteracji w funkcji nauczania
* przy regule uczenia delta w sieci sigmoidalnej nie było prawie w ogóle błędów przy danych testujących
* przy modelu adaline pojawiało się bardzo dużo błędów przy danych testujących
* liczba epok przy regule uczenia delta była o wiele większa niż przy modelu adaline
* przy współczynniku uczenia większym niż 0,1 dla modelu Adaline, sieć miała problemy aby się nauczyć

1. Wnioski
   * współczynnik uczenia wpływa na liczbę iteracji(epok) uczenia: im większy tym mniej epok
   * model sieci sigmoidalnej przy regule uczenia się delta jest o wiele bardziej sukcesywny w nauce niż model adaline
   * model adaline uczy się niemal kilkadziesiąt razy szybciej niż model sieci sigmoidalnej
   * liczba epok jest również zależna od wylosowanych początkowo wag
   * różnica pomiędzy wagami końcowymi przy mniejszym i większym współczynniku uczenia spowodowana jest wartością współczynnika wykorzystywaną do aktualizowania wag
2. Listing kodu:
3. **Adaline.java**
4. **import java.util.Random;**
5. **public class** Adaline {
6. **public double**[] **weight**; *//tablica wag*
7. **public static int**[][] *set* = { *//tablica danych uczacych*
8. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, *//A*
9. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,1,1,0}, *//C*
10. {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1}, *//E*
11. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *// G*
12. {0,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,0}, *//I*
13. {1,0,0,0,1,1,0,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0,1,0,0,0,1}, *//K*
14. {1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, *//M*
15. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//O*
16. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0,0,1}, *//R*
17. {1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0}, *//T*
18. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,1}, *//a*
19. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1}, *//d*
20. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,0,1}, *//h*
21. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1}, *//l*
22. {0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0}, *//p*
23. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,1}, *//u*
24. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,0,1,0,0,1,0,1,0,0,0,0,1,0,0,0}, *//r*
25. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1}, *//c*
26. {0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,1,0}, *//f*
27. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0}}; *//n*
28. **public static int**[] *setOutput* = {1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0}; *//tablica oczekiwanych wynikow*
29. **public static char**[] *setLetters* = {**'A'**,**'C'**,**'E'**,**'G'**,**'I'**,**'K'**,**'M'**,**'O'**,**'R'**,**'T'**,**'a'**,**'d'**,**'h'**,**'l'**,**'p'**,**'u'**,**'r'**,**'c'**,**'f'**,**'n'**};
30. **public static int**[][] *setTest* = { *//tablica danych testowych*
31. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0}, *//B*
32. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0}, *//D*
33. {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0}, *//F*
34. {1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, *// H*
35. {1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//J*
36. {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1}, *//L*
37. {1,0,0,0,1,1,0,0,0,1,1,1,0,0,1,1,0,1,0,1,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1}, *//N*
38. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0}, *//P*
39. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//S*
40. {1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//U*
41. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1}, *//b*
42. {0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,0,0,1,1,1,1}, *//e*
43. {0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,1,0}, *//f*
44. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,0}, *//o*
45. {0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,1,0,0}, *//s*
46. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1}, *//m*
47. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0}, *//w*
48. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,1,0,0,1,1,0,0,0,1,0,1}, *//k*
49. {0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1}, *//g*
50. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0}}; *//n*
51. **public static char**[] *setTestLetters* = {**'B'**,**'D'**,**'F'**,**'H'**,**'J'**,**'L'**,**'N'**,**'P'**,**'S'**,**'U'**,**'b'**,**'e'**,**'f'**,**'o'**,**'s'**,**'m'**,**'w'**,**'k'**,**'g'**,**'n'**};
52. **public int numberOfSets**; *//liczba liter*
53. **public int numberOfWeights**; *//liczba wag*
54. **public double learningRate**; *//wspolczynnik uczenia*
55. **public double mainError**; *//blad globalny*
56. **public double delta**;
57. *//konstruktor*
58. **public** Adaline() {
59. **delta**=0;
60. **numberOfWeights** = 35;
61. **numberOfSets** = 20;
62. **learningRate** = 0.1;
63. **mainError**=0.0;
64. **weight** = **new double**[**numberOfWeights**];
65. }
66. *//ustalanie wag jako wartosci losowcyh*
67. **public void** establishingWeights(){
68. Random randomWeight = **new** Random();
69. **for**(**int** i=0;i<**numberOfWeights**;i++) {
70. **weight**[i] = randomWeight.nextDouble(); *//losowanie wag z zakresu od 0 do 1*
71. }
72. }
73. *//funkcja aktywacji - funkcja progowa*
74. **public boolean** activationFunction(**double** sum){
75. **if**(sum > 0.5)
76. **return true**;
77. **else**
78. **return false**;
79. }
80. *//funkcja sumujaca*
81. **public double** sumFunction(**int** letter[], **double**[]weights)
82. {
83. **double** sum = 0.0;
84. **for**(**int** i=0; i < **numberOfWeights**; i++)
85. sum += letter[i] \* weights[i];
86. **return** sum;
87. }
88. **public void** learning() {
89. System.***out***.println(**"\n** **ADALINE \n"**);
90. *//zmienna, stwierdzajaca czy blad jest mozliwy do zaakceptowania*
91. **boolean** mayErrorBe = **false**;
92. *//numer epoki*
93. **int** iteration=0;
94. *//ustalanie wag poczatkowych*
95. establishingWeights();
96. **for**(**int** i=0;i<**numberOfWeights**;i++)
97. System.***out***.println(**"Weights are: w"** + i + **" = "**+ **weight**[i]);
98. System.***out***.println(**"\n"**);
99. *//podczas jednej epoki nauka z całego zestawu danych uczacych*
100. **do**{
101. iteration++; *// zwiekszenie numeru iteracji(epoki)*
102. **mainError**=0.0; *//zerowanie glownego bledu w celu sprawdzenia bledow podczas jednej iteracji*
103. **for**(**int** i=0;i<**numberOfSets**;i++) {
104. *//obliczanie roznicy pomiedzy wynikiem oczekiwanym a wynikiem otrzymanym*
105. **delta** = *setOutput*[i] - sumFunction(*set*[i], **weight**);
106. *//aktualizowanie wag*
107. **for**(**int** j=0; j < **numberOfWeights**;j++)
108. **weight**[j] +=**learningRate**\***delta**\**set*[i][j];
109. *//aktualizowanie bledu glownego*
110. **mainError**+=**delta**\***delta**;
111. }
112. **mainError**/=2;
113. **if**(**mainError**>0.001) {
114. mayErrorBe = **false**;
115. }
116. **else** {
117. mayErrorBe = **true**;
118. }
119. }**while**(!mayErrorBe);
120. System.***out***.println(**"ITERATION: "** + iteration );
121. System.***out***.println(**"Main error: "** + **mainError**);
122. **for**(**int** i=0;i<**numberOfWeights**;i++)
123. System.***out***.println(**"Weights are: w"** + i + **" = "**+ **weight**[i]);
124. System.***out***.println(**"\n"**);
125. }
126. *//funkcja testujaca*
127. **public void** testing()
128. {
129. System.***out***.print(**"Letters from learning set: \n"**);
130. **for**(**int** i=0;i<**numberOfSets**;i++) {
131. System.***out***.print(**"Letter "**+ *setLetters*[i] +**" is: "**);
132. **if**(activationFunction(sumFunction(*set*[i], **weight**))) {
133. System.***out***.println(**"big"**);
134. }**else**{
135. System.***out***.println(**"small"**);
136. }
137. }
138. System.***out***.print(**"Letters from testing set: \n"**);
139. **for**(**int** i=0;i<**numberOfSets**;i++) {
140. System.***out***.print(**"Letter "**+ *setTestLetters*[i] +**" is: "**);
141. **if**(activationFunction(sumFunction(*setTest*[i], **weight**))) {
142. System.***out***.println(**"big "**);
143. }**else**{
144. System.***out***.println(**"small "**);
145. }
146. }
147. }
148. }
149. **DeltaSigmoidal.java**
150. **import java.util.Random;**
151. **public class** DeltaSigmoidal {
152. **public double**[] **weight**; *//tablica wag*
153. **public static int**[][] *set* = { *//tablica danych uczacych*
154. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, *//A*
155. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,1,1,0}, *//C*
156. {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1}, *//E*
157. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *// G*
158. {0,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,0}, *//I*
159. {1,0,0,0,1,1,0,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0,1,0,0,0,1}, *//K*
160. {1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, *//M*
161. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//O*
162. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0,0,1}, *//R*
163. {1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0}, *//T*
164. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,1}, *//a*
165. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1}, *//d*
166. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,0,1}, *//h*
167. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1}, *//l*
168. {0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0}, *//p*
169. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,1}, *//u*
170. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,0,1,0,0,1,0,1,0,0,0,0,1,0,0,0}, *//r*
171. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1}, *//c*
172. {0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,1,0}, *//f*
173. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0}}; *//n*
174. **public static int**[] *setOutput* = {1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0}; *//tablica oczekiwanych wynikow*
175. **public static char**[] *setLetters* = {**'A'**,**'C'**,**'E'**,**'G'**,**'I'**,**'K'**,**'M'**,**'O'**,**'R'**,**'T'**,**'a'**,**'d'**,**'h'**,**'l'**,**'p'**,**'u'**,**'r'**,**'c'**,**'f'**,**'n'**};
176. **public static int**[][] *setTest* = { *//tablica danych testowych*
177. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0}, *//B*
178. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0}, *//D*
179. {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0}, *//F*
180. {1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1}, *// H*
181. {1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//J*
182. {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1}, *//L*
183. {1,0,0,0,1,1,0,0,0,1,1,1,0,0,1,1,0,1,0,1,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1}, *//N*
184. {1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0}, *//P*
185. {0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//S*
186. {1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0}, *//U*
187. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1}, *//b*
188. {0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,0,0,1,1,1,1}, *//e*
189. {0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,1,0}, *//f*
190. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,0}, *//o*
191. {0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,1,0,0}, *//s*
192. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1}, *//m*
193. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0}, *//w*
194. {0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,1,0,0,1,1,0,0,0,1,0,1}, *//k*
195. {0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0,1,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1}, *//g*
196. {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0}}; *//n*
197. **public static char**[] *setTestLetters* = {**'B'**,**'D'**,**'F'**,**'H'**,**'J'**,**'L'**,**'N'**,**'P'**,**'S'**,**'U'**,**'b'**,**'e'**,**'f'**,**'o'**,**'s'**,**'m'**,**'w'**,**'k'**,**'g'**,**'n'**};
198. **public int numberOfSets**; *//liczba liter*
199. **public int numberOfWeights**; *//liczba wag*
200. **public double learningRate**; *//wspolczynnik uczenia*
201. **public double mainError**; *//blad globalny*
202. **public double delta**;
203. **public double output**;
204. *//konstruktor*
205. **public** DeltaSigmoidal() {
206. **delta**=0;
207. **numberOfWeights** = 35;
208. **numberOfSets** = 20;
209. **learningRate** = 0.1;
210. **mainError**=0;
211. **output**=0;
212. **weight** = **new double**[**numberOfWeights**];
213. }
214. *//ustalanie wag jako wartosci losowcyh*
215. **public void** establishingWeights(){
216. Random randomWeight = **new** Random();
217. **for**(**int** i=0;i<**numberOfWeights**;i++) {
218. **weight**[i] = randomWeight.nextDouble(); *//losowanie wag z zakresu od 0 do 1*
219. }
220. }
221. *//funkcja aktywacji - funkcja sigmoidalna*
222. **public double** activationFunction(**double** sum){
223. **return** ( 1 / ( 1 + Math.*exp*(-1.0 \* sum)));
224. }
225. *//*
226. **public double** sumFunction(**int** letter[], **double**[]weights)
227. {
228. **double** sum = 0.0;
229. **for**(**int** i=0; i < **numberOfWeights**; i++)
230. sum += letter[i] \* weights[i];
231. **return** sum;
232. }
233. **public void** learning() {
234. System.***out***.println(**"\n** **DELTA SIGMOIDAL \n"**);
235. *//zmienna, stwierdzajaca czy blad jest mozliwy do zaakceptowania*
236. **boolean** mayErrorBe = **false**;
237. *//numer epoki*
238. **int** iteration=0;
239. *//ustalanie wag poczatkowych*
240. establishingWeights();
241. **for**(**int** i=0;i<**numberOfWeights**;i++)
242. System.***out***.println(**"Weights are: w"** + i + **" = "**+ **weight**[i]);
243. System.***out***.println(**"\n"**);
244. *//podczas jednej epoki nauka z całego zestawu danych uczacych*
245. **do**{
246. iteration++; *// zwiekszenie numeru iteracji(epoki)*
247. **mainError**=0.0; *//zerowanie glownego bledu w celu sprawdzenia bledow podczas jednej iteracji*
248. **for**(**int** i=0;i<**numberOfSets**;i++) {
249. **output**=activationFunction(sumFunction(*set*[i], **weight**));
250. *//obliczanie roznicy pomiedzy wynikiem oczekiwanym a wynikiem otrzymanym*
251. **delta** = *setOutput*[i] - **output**;
252. *//aktualizowanie wag*
253. **for**(**int** j=0; j < **numberOfWeights**;j++)
254. **weight**[j] +=**learningRate**\***delta**\**set*[i][j]\*(1-**output**)\***output**;
255. *//aktualizowanie bledu glownego*
256. **mainError**+=**delta**\***delta**;
257. }
258. **mainError**/=2;
259. **if**(**mainError**>0.001) {
260. mayErrorBe = **false**;
261. }
262. **else** {
263. mayErrorBe = **true**;
264. }
265. }**while**(!mayErrorBe);
266. System.***out***.println(**"ITERATION: "** + iteration );
267. System.***out***.println(**"Main error: "** + **mainError**);
268. **for**(**int** i=0;i<**numberOfWeights**;i++)
269. System.***out***.println(**"Weights are: w"** + i + **" = "**+ **weight**[i]);
270. System.***out***.println(**"\n"**);
271. }
272. *//funkcja testujaca*
273. **public void** testing()
274. {
275. System.***out***.print(**"Letters from learning set: \n"**);
276. **for**(**int** i=0;i<**numberOfSets**;i++) {
277. System.***out***.print(**"Letter "**+ *setLetters*[i] +**" is: "**);
278. **if**(activationFunction(sumFunction(*set*[i], **weight**))>0.5) {
279. System.***out***.println(**"big"**);
280. }**else**{
281. System.***out***.println(**"small"**);
282. }
283. }
284. System.***out***.print(**"Letters from testing set: \n"**);
285. **for**(**int** i=0;i<**numberOfSets**;i++) {
286. System.***out***.print(**"Letter "**+ *setTestLetters*[i] +**" is: "**);
287. **if**(activationFunction(sumFunction(*setTest*[i], **weight**))>0.5) {
288. System.***out***.println(**"big"**);
289. }**else**{
290. System.***out***.println(**"small"**);
291. }
292. }
293. }
294. }
295. **main.java**
296. **public class** main {
297. **public static void** main(String[] args) {
298. Adaline adaline = **new** Adaline();
299. adaline.learning();
300. adaline.testing();
301. DeltaSigmoidal deltasigmoidal = **new** DeltaSigmoidal();
302. deltasigmoidal.learning();
303. deltasigmoidal.testing();
304. }
305. }