|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Jarosz Wojciech | Gr. 1 | PSI | Sprawozdanie 4 | Temat: Uczenie sieci regułą Hebba |

1. Cel ćwiczenia:

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

1. Wykonane zadania:
   1. Wygenerowano danych uczących i testujących, zawierających 4 czarno-białe, różne emotikony, wymiar 8x8 pikseli dla jednej emotikony.
   2. Przygotowano (implementacja) sieć oraz regułę Hebba w wersji z i bez współczynnika zapominania.
   3. Nauczono sieć dla różnych współczynników uczenia i zapominania.
   4. Przetestowano sieć.

Emotikony: 1. „Happy”, 2. „Sad”, 3. „Smile”, 4. „Neutral face”

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

1. Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia:

Metoda Hebba polega na uczeniu sieci bez nauczyciela. Oznacza to, że sieć nie otrzymuje żadnych informacji o oczekiwanym wyniku. Sieć ma za zadanie obserwować różne, otrzymane sygnały, nie wiedząc co należy z nimi zrobić. Na podstawie otrzymanych danych wejściowych sieć musi wykryć zależności pomiędzy nimi i określać ich znaczenie samodzielnie.

Algorytm:

Modyfikacja wag:

,gdzie :

* p – wartość wejściowa neuron
* a – sygnał wyjściowy neuronu
* α – współczynnik uczenia
* w – wagi

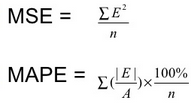
W tym przypadku nie podajemy wzorcowej wartości wyjściowej – bez nauczyciela.

Wadą powyższego rozwiązanie jest niekontrolowany wzrost wag – z tego powodu przeprowadzono

modyfikację uczenia – wprowadzenie współczynnika zapominania:

* β – współczynnik zapominania

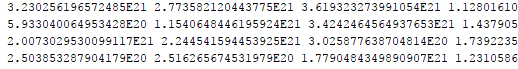
Błędy uczenia:



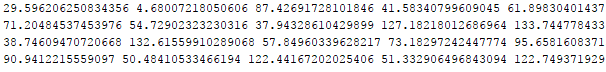
1. Zestawienie otrzymanych wyników:

Przykładowe wagi dla algorytmu bez współczynnika zapominania po wyuczeniu sieci:

Współczynnik uczenia: 0.01

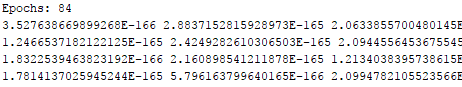


Współczynnik uczenia: 0.001

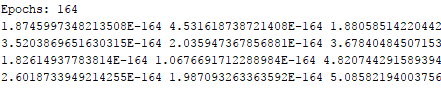


Przykładowe wagi dla algorytmu z współczynnikiem zapominania po wyuczeniu sieci:

Współczynnik zapominania = 0.01;



Współczynnik zapominania = 0.1:



Testowanie dla zniekształconych danych:

Test 0:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

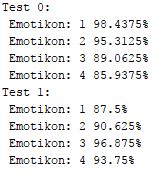
Test 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

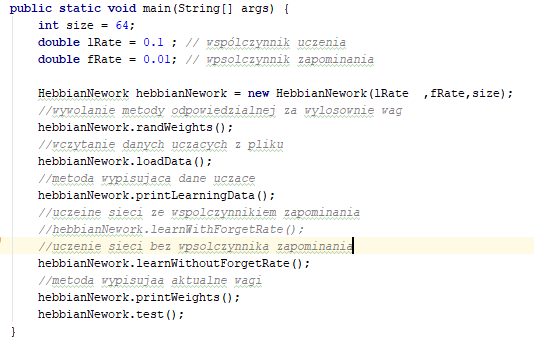
Wyniki testów:

Test 0 – zniekształcona tylko o jeden piksel w mało znaczącym miejscu emotka „happy”

Test 1 – zniekształcona emotka „sad” w kluczowym dla rozpoznania miejscu

1. „Happy”, 2. „Smile”, 3. „Sad”, 4.”Neutral Face”

1. Analiza i dyskusja błędów:
   1. Uczenie sieci ze współczynnikiem zapominania trwało znacznie krócej niż uczenie sieci bez tego współczynnika – liczba epok wynosiła zazwyczaj maksymalną, ustawioną liczbę epok
   2. Po zmianie współczynnika uczenia na większy następował wzrost wag wyuczonej sieci
   3. Dla mało zniekształconych emotikonek testowanie dawało wyniki jednoznaczne
   4. Dla zniekształconych emotikonek w miejscach kluczowych dla rozpoznania, testowanie zwracało podobne wyniki dla kilku z nich
2. Wnioski:
   1. Współćzynnik zapominania pozwalał na kontrolowanie wag – nie następował znaczny ich wzrost, natomiast w każdej epoce wagi były odpowiednio zmniejszane
   2. Im mniejszy współczynnik uczenia tym mniejsze wagi
   3. Im bardziej zniekształcone emotikony, tym trudniej o analizę zwróconych przez sieć wyników
   4. Dla otrzymanych wyników bardzo ważne jest, w których miejscach zostały zniekształcone. Jeżeli zniekształcona została obwódka – możliwe było wskazanie poprawnej odpowiedzi, a jeżeli została zniekształcona mimika twarzy- wyniki były bardzo niejednoznaczne.
3. Listning kodu:



**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.Random;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* Created by Wojtek on 06.12.2017.  
 \*/***public class** HebbianNework {  
  
 **double lRate**;  
 **double fRate**;  
 **int size**;  
 **int epoch**;  
 **double e**;  
 **double**[][] **learningData**; *//dane uczace* **double**[][] **testingData**; *//dane testujace* **double**[][] **weights**;*//wagi* **double**[] **a**; *//sygnal wyjsciowy* **public** HebbianNework(**double** lRate, **double** fRate, **int** size) {  
 **this**.**lRate** = lRate;  
 **this**.**fRate** = fRate;  
 **this**.**size** = size;  
 **learningData** = **new double**[size][4];  
 **testingData** = **new double**[size][2];  
 **weights** = **new double**[size][4];  
 **epoch** = 0;  
 **e** = 0.0;  
 **a** = **new double**[size];  
 **for**(**int** i=0;i<size;i++){  
 **a**[i] = 1.;  
 }  
 }  
*//metoda generujaca wagi* **void** randWeights(){  
 **for**(**int** i=0; i< **size**;i++){  
 **for** (**int** j=0;j<4;j++){  
 **weights**[i][j] = Math.*random*();  
 }  
 }  
 }  
*//wczytanie danych uczących z pliku* **void** loadData(){  
 **double** n=Math.*sqrt*(**size**);  
 File file = **new** File(**"learning\_data.txt"**);  
 Scanner in = **null**;  
 String[] result;  
 **try** {  
 in = **new** Scanner(file);  
 **for**(**int** i=0;i<4;i++){  
 *//while (in.hasNextLine()){* String line = in.nextLine();  
 result = line.split(**"\\s"**);  
 **for**(**int** j=0; j< 64; j++){  
 **learningData**[j][i]=Double.*parseDouble*(result[j]);  
 }  
 }  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
*//wypisanie dannych uczacych* **void** printLearningData(){  
 System.***out***.println(**"Learning Data:"**);  
 **for**(**int** i=0;i<4;i++){  
 **for**(**int** j=0; j< 64; j++){  
 System.***out***.print(**learningData**[j][i]+**" "**);  
 }  
 System.***out***.println(**""**);  
 }  
 }  
*//metoda odpowiedzialna za uczenie sieci ze współczynnikiem zapominania* **void** learnWithForgetRate() {  
 **double** tmp = 0.0;  
 **double** error = 0.0;  
 **do** {  
 **for** (**int** i = 0; i < 4; ++i) {  
 **e** = 0.0;  
 **for** (**int** j = 0; j < **size**; ++j) {  
 tmp = **a**[j];  
 *//sygnal wyjsciowy* **a**[j] = (**weights**[j][i] \* **learningData**[j][i]);  
 *//nowe wagi = stare wagi \* wsp zapominania \* sygnal wyjsciowy \* wsp uczenia* **weights**[j][i] = **weights**[j][i] \* **fRate** + **lRate** \* **a**[j] \* **learningData**[j][i];  
 *//bledy* **if** (error == Math.*abs*(tmp - **a**[j])) **break**;  
 error = Math.*abs*(tmp - **a**[j]);  
 **e** = **e** + Math.*pow*(error, 2);  
 }  
 *//bledy MSE i MAPE* **double** MSE = Math.*pow*(**e**,2)/(**size**);  
 **double** MAPE = (**e**\*100/**size**);  
 System.***out***.println(**" MSE: "** + MSE + **" MAPE: "** + MAPE + **"%"**);  
  
 }  
 **epoch**++;  
 }  
 **while** (**e** != 0 && **epoch** < 5000) ;  
 System.***out***.println(**"Learning finished!!!\nEpochs: "** + **epoch**);  
  
  
 }  
*// metoda odpowedzialna za uczenie sieci bez wpółczynnika zapominania* **void** learnWithoutForgetRate(){  
 **double** tmp = 0.0;  
 **double** error = 0.0;  
 **do**{  
 **for**(**int** i=0;i<4;++i){  
 **e** = 0.0;  
 **for**(**int** j=0;j<**size**;++j){  
 tmp = **a**[j];  
 *//sygnal wyjsciowy* **a**[j] = **weights**[j][i]\***learningData**[j][i];  
 *//nowe wagi = stare wagi \* wsp zapominania \* sygnal wyjsciowy \* wsp uczenia* **weights**[j][i] = **weights**[j][i]+**lRate**\***a**[j]\***learningData**[j][i];  
 *//bledy* **if**(error==Math.*abs*(tmp-**a**[j])) **break**;  
 error = Math.*abs*(tmp - **a**[j]);  
 **e** = **e** + Math.*pow*(error, 2);  
  
 }  
 *//bledy MSE i MAPE* **double** MSE = Math.*pow*(**e**,2)/(**size**);  
 **double** MAPE = (**e**\*100/**size**);  
 System.***out***.println(**" MSE: "** + MSE + **" MAPE: "** + MAPE + **"%"**);  
 }  
  
 **epoch**++;  
 }**while**(**e**!=0 && **epoch**<5000);  
 System.***out***.println(**"Learning finished!!!\nEpochs: "**+**epoch**);  
  
  
 }  
  
   
 **void** printWeights(){  
 **for**(**int** i =0; i<4;i++) {  
 **for** (**int** j = 0; j < **size**; j++) {  
 System.***out***.print(**weights**[j][i] + **" "**);  
 }  
 System.***out***.println(**""**);  
 }  
 }  
  
}