|  |  |
| --- | --- |
| **Scenariusz 6** | |
| **Temat ćwiczenia:** | Budowa i działanie sieci Kohonena dla WTM |
| **Wykonał:** | Mateusz Mazur, gr 2 |

1. **Cel ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działanie sieci Kohonena przy wykorzystaniu reguły WTM dla sieci jednowarstwowej na przykładzie grupowania liter alfabetu.

1. **Budowa sieci i wykorzystany algorytm uczenia**

Celem budowanej sieci jest podział wzorców uczących na klasy obrazów zbliżonych do siebie i przyporządkowanie każdej klasie osobnego elementu wyjściowego. Podział na grupy odbywa się w takich sposób, by elementy w tej samej grupie były do siebie podobne a jednocześnie jak najbardziej odmienne od elementów z pozostałych grup.

Tab. 1 Przygotowany alfabet

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0110  1001  1111  1001  1001 | D:\Pobrane\Alfabet (1)\Alfabet\A.png | 1110  1001  1110  1001  1110 |
| D:\Pobrane\Alfabet (1)\Alfabet\G.png | 0110  1001  1000  1001  0110 | D:\Pobrane\Alfabet (1)\Alfabet\D.png | 1110  1001  1001  1001  1110 |
| D:\Pobrane\Alfabet (1)\Alfabet\D.png | 1111  1000  1111  1000  1111 | D:\Pobrane\Alfabet (1)\Alfabet\F.png | 1111  1000  1110  1000  1000 |
| D:\Pobrane\Alfabet (1)\Alfabet\F.png | 1111  1000  1011  1001  1111 | D:\Pobrane\Alfabet (1)\Alfabet\H.png | 1001  1001  1111  1001  1001 |
| D:\Pobrane\Alfabet (1)\Alfabet\J.png | 1111  0110  0110  0100  1111 | D:\Pobrane\Alfabet (1)\Alfabet\J.png | 1111  0001  0011  0101  1111 |
| D:\Pobrane\Alfabet (1)\Alfabet\L.png | 1001  1010  1100  1010  1001 | D:\Pobrane\Alfabet (1)\Alfabet\L.png | 1000  1000  1000  1000  1111 |
| D:\Pobrane\Alfabet (1)\Alfabet\B.png | 1001  1001  1101  1011  1001 | D:\Pobrane\Alfabet (1)\Alfabet\B.png | 0110  1001  1001  1001  0110 |
| D:\Pobrane\Alfabet (1)\Alfabet\P.png | 1111  1001  1111  1000  1000 | D:\Pobrane\Alfabet (1)\Alfabet\P.png | 1111  1001  1111  1010  1001 |
| D:\Pobrane\Alfabet (1)\Alfabet\H.png | 1111  1111  0110  0110  0110 | D:\Pobrane\Alfabet (1)\Alfabet\E.png | 1001  1001  1001  1001  1111 |
| D:\Pobrane\Alfabet (1)\Alfabet\E.png | 1001  1001  1111  0110  0110 | D:\Pobrane\Alfabet (1)\Alfabet\G.png | 1111  0001  0110  1000  1111 |

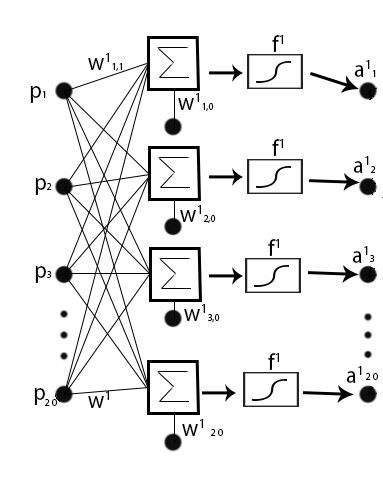
Klasyfikacja liter na poszczególne grupy odbywa się na podstawie obliczonej odległości euklidesowej między poszczególnymi literami według wzoru:

Tab 2. Zestawienie wartości odległości euklidesowych między poszczególnymi literami

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I | J | K | L | N | O | P | R | T | U | Y | Z |
| A | x | 2,24 | 2,65 | 2,45 | 2,45 | 2,45 | 2,45 | 2,00 | 3,74 | 3,00 | 3,16 | 3,16 | 2,45 | 2,45 | 2,00 | 2,00 | 3,74 | 2,83 | 3,46 | 2,83 |
| B | 2,24 | x | 2,00 | 1,73 | 2,24 | 2,24 | 2,24 | 2,65 | 3,32 | 2,83 | 3,32 | 2,65 | 3,16 | 2,24 | 2,24 | 2,65 | 3,00 | 2,83 | 3,00 | 2,24 |
| C | 2,65 | 2,00 | x | 1,73 | 3,00 | 3,00 | 2,65 | 3,32 | 3,87 | 3,16 | 3,61 | 2,65 | 3,32 | 1,00 | 3,00 | 3,32 | 3,16 | 2,65 | 3,32 | 3,00 |
| D | 2,45 | 1,73 | 1,73 | x | 2,45 | 2,83 | 2,00 | 2,83 | 3,74 | 2,65 | 3,46 | 2,45 | 2,83 | 1,41 | 2,45 | 2,83 | 3,46 | 2,00 | 3,16 | 2,83 |
| E | 2,45 | 2,24 | 3,00 | 2,45 | x | 2,00 | 1,41 | 2,45 | 2,83 | 2,65 | 2,83 | 2,45 | 2,83 | 2,83 | 2,00 | 2,00 | 3,16 | 2,45 | 2,83 | 2,00 |
| F | 2,45 | 2,24 | 3,00 | 2,83 | 2,00 | x | 2,45 | 2,45 | 3,16 | 3,32 | 2,45 | 2,83 | 2,83 | 3,16 | 1,41 | 2,00 | 3,16 | 3,16 | 3,16 | 2,45 |
| G | 2,45 | 2,24 | 2,65 | 2,00 | 1,41 | 2,45 | x | 2,45 | 3,16 | 2,24 | 3,16 | 2,45 | 2,83 | 2,45 | 2,45 | 2,45 | 3,46 | 2,00 | 3,16 | 2,45 |
| H | 2,00 | 2,65 | 3,32 | 2,83 | 2,45 | 2,45 | 2,45 | x | 3,74 | 3,00 | 2,45 | 2,83 | 1,41 | 3,16 | 2,00 | 2,00 | 3,74 | 2,00 | 2,83 | 2,83 |
| I | 3,74 | 3,32 | 3,87 | 3,74 | 2,83 | 3,16 | 3,16 | 3,74 | x | 2,65 | 3,16 | 3,46 | 3,74 | 4,00 | 3,46 | 3,46 | 2,00 | 3,74 | 3,16 | 2,45 |
| J | 3,00 | 2,83 | 3,16 | 2,65 | 2,65 | 3,32 | 2,24 | 3,00 | 2,65 | x | 3,87 | 3,32 | 3,32 | 3,00 | 3,00 | 3,00 | 3,00 | 2,65 | 3,00 | 2,24 |
| K | 3,16 | 3,32 | 3,61 | 3,46 | 2,83 | 2,45 | 3,16 | 2,45 | 3,16 | 3,87 | x | 2,45 | 2,00 | 3,74 | 2,83 | 2,45 | 3,46 | 2,83 | 3,16 | 3,16 |
| L | 3,16 | 2,65 | 2,65 | 2,45 | 2,45 | 2,83 | 2,45 | 2,83 | 3,46 | 3,32 | 2,45 | x | 2,83 | 2,83 | 3,16 | 3,16 | 3,74 | 2,00 | 3,16 | 2,83 |
| N | 2,45 | 3,16 | 3,32 | 2,83 | 2,83 | 2,83 | 2,83 | 1,41 | 3,74 | 3,32 | 2,00 | 2,83 | x | 3,16 | 2,45 | 2,00 | 3,74 | 2,00 | 2,83 | 3,16 |
| O | 2,45 | 2,24 | 1,00 | 1,41 | 2,83 | 3,16 | 2,45 | 3,16 | 4,00 | 3,00 | 3,74 | 2,83 | 3,16 | x | 2,83 | 3,16 | 3,46 | 2,45 | 3,16 | 3,16 |
| P | 2,00 | 2,24 | 3,00 | 2,45 | 2,00 | 1,41 | 2,45 | 2,00 | 3,46 | 3,00 | 2,83 | 3,16 | 2,45 | 2,83 | x | 1,41 | 3,16 | 2,83 | 2,83 | 2,83 |
| R | 2,00 | 2,65 | 3,32 | 2,83 | 2,00 | 2,00 | 2,45 | 2,00 | 3,46 | 3,00 | 2,45 | 3,16 | 2,00 | 3,16 | 1,41 | x | 3,16 | 2,83 | 2,83 | 2,83 |
| T | 3,74 | 3,00 | 3,16 | 3,46 | 3,16 | 3,16 | 3,46 | 3,74 | 2,00 | 3,00 | 3,46 | 3,74 | 3,74 | 3,46 | 3,16 | 3,16 | x | 3,74 | 2,45 | 2,83 |
| U | 2,83 | 2,83 | 2,65 | 2,00 | 2,45 | 3,16 | 2,00 | 2,00 | 3,74 | 2,65 | 2,83 | 2,00 | 2,00 | 2,45 | 2,83 | 2,83 | 3,74 | x | 2,83 | 2,83 |
| Y | 3,46 | 3,00 | 3,32 | 3,16 | 2,83 | 3,16 | 3,16 | 2,83 | 3,16 | 3,00 | 3,16 | 3,16 | 2,83 | 3,16 | 2,83 | 2,83 | 2,45 | 2,83 | x | 3,16 |
| Z | 2,83 | 2,24 | 3,00 | 2,83 | 2,00 | 2,45 | 2,45 | 2,83 | 2,45 | 2,24 | 3,16 | 2,83 | 3,16 | 3,16 | 2,83 | 2,83 | 2,83 | 2,83 | 3,16 | x |

Analizując powyższe wyniki podzielono litery na 9 grup:

|  |  |
| --- | --- |
| Lp. grupy | Liczby zawarte w grupie |
| 1 | A |
| 2 | B C D O |
| 3 | E G Z |
| 4 | F P R |
| 5 | I T |
| 6 | J |
| 7 | H N K |
| 8 | L U |
| 9 | Y |



Rys 1. Schemat sieci neuronowej

Proces uczenia odbywa się przy pomocy uczenia rywalizującego, które jest metodą uczenia sieci samoorganizujących. Podczas uczenia neurony uczą się rozpoznawać dane i zbliżają się odpowiednio do obszarów zajmowanych przez te dane. Po wejściu każdego wektora uczącego wybierany jest tylko jeden neuron, najbliższy prezentowanemu wzorcowi. Neurony rywalizują między sobą i zwycięża ten neuron, którego wartość jest największa. Zwycięski neuron przyjmuje na wyjściu wartość 1, a pozostałe neurony 0. Jest to uczenie bez nauczyciela.

Wielokrotne testy różnych konfiguracji wykazały, że dla danego problemu najlepszymi wynikami cechuje się sieć zbudowana z 20 neuronów, która w dodatku biorąc pod uwagę ilość neuronów charakteryzuje się małym skomplikowaniem.

1. Wybór za zakresu 0 do 1   
2. Normalizacja wektorów danych wejściowych  
3. Wybór początkowych wartości wag jako niewielkich liczb losowych z zakresu -1 do 1  
4. Dla pojedynczego neuronu obliczana jest odległość euklidesowa sygnałów wejściowych wektora uczącego i wag   
5. Wybierany jest neuron, dla którego obliczona odległość euklidesowa jest najmniejsza i zaktualizowanie jego wag według wzoru:

gdzie:

η to współczynnik uczenia wybierany z zakresu od 0 do 1

to funkcja sąsiedztwa obliczana według wzoru

gdzie:

d – dystans od zwycięskiego neuronu obliczany ze wzoru:

i – wektor wejściowy

w – waga neuronu

σ(t) – promień sąsiedztwa obliczany ze wzoru

t – obecna iteracja

λ – stała czasowa obliczana ze wzoru

– promień mapy sieci

6. Znormalizowanie wartości nowego wektora wag  
7. Zwycięski neuron daje odpowiedź na swoim wyjściu 1, a pozostałe 0  
8. Wczytanie kolejnego wektora uczącego

1. **Otrzymane wyniki**

Otrzymane przeze mnie wyniki różnią się mocno dla różnych wartości współczynnika uczenia. Podczas procesu uczenia zbadano ilość zmian odpowiedzi zwycięskiego neuronu pomiędzy pierwszą, a ostatnią epoką uczenia. Z kolei podczas procesu testowania przeprowadzono cztery próby, a następnie uzyskane rezultaty uśredniono co pozwala dokładniej ocenić efektywność i poprawność odpowiedzi sieci przy różnych wartościach współczynnika uczenia. Dla każdej z prób zebrano dane ukazujące alfabet oraz to do jakich grup zostały przyporządkowane litery go tworzące.

Tab 3. Ilość zmian odpowiedzi neuronu pomiędzy pierwszą, a ostatnią epoką uczenia podczas procesu uczenia

|  |  |
| --- | --- |
| Wartość współczynnika uczenia | Ilość zmian odpowiedzi neuronu pomiędzy pierwszą, a ostatnią epoką |
|
|
| 0,0001 | 1 |
| 0,001 | 4 |
| 0,005 | 11 |
| 0,01 | 13 |
| 0,09 | 15 |
| 0,1 | 19 |

Tab 4. Zestawienie wyników dla procesu testowania oraz grupowanie poszczególnych liter alfabetu dla czterech prób przy różnych wartościach współczynnika uczenia

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wartość współczynnika uczenia | Podział liter na grupy dla poszczególnych prób | | | |
|
| Próba 1 | Próba 2 | Próba 3 | Próba 4 |
| 0,0001 | Grupa 1: L | Grupa 1: A, D, O, P | Grupa 1: I | Grupa 1: G, U |
| Grupa 2: H, N, U | Grupa 2: T | Grupa 2: H, N, R | Grupa 2: E, I, Z |
| Grupa 3: C | Grupa 3: Y | Grupa 3: B, C, D, O, U | Grupa 3: H, L |
| Grupa 4: B | Grupa 4: I | Grupa 4: J | Grupa 4: B, J |
| Grupa 5: K | Grupa 5: C | Grupa 5: Y | Grupa 5: Y |
| Grupa 6: R | Grupa 6: N | Grupa 6: T | Grupa 6: A, C, D, O |
| Grupa 7: A, E, F, G | Grupa 7: F, K, R | Grupa 7: E, F, G, K, L, P, Z | Grupa 7: F, P, R |
| Grupa 8: Y | Grupa 8: U | Grupa 8: A | Grupa 8: K, N |
| Grupa 9: J | Grupa 9: E, G, H, L, Z |  | Grupa 9: T |
| Grupa 10: I, T, Z | Grupa 10: J |  |  |
| Grupa 11: D, O, P | Grupa 11: B |  |  |
| 0,001 | Grupa 1: T | Grupa 1: T | Grupa 1: K | Grupa 1: I |
| Grupa 2: K, N, P, R | Grupa 2: I | Grupa 2: F, G | Grupa 2: A, B, C, D, H, N |
| Grupa 3: D | Grupa 3: D, O | Grupa 3: I ,T, Y | Grupa 3: L, U, Y |
| Grupa 4: B | Grupa 4: B | Grupa 4: E, Z | Grupa 4: G, O |
| Grupa 5: Y | Grupa 5: C, L | Grupa 5: C | Grupa 5: J |
| Grupa 6: E, J | Grupa 6: G | Grupa 6: B, L, U | Grupa 6: T |
| Grupa 7: A, G, H, U | Grupa 7: A, H, N, R, U | Grupa 7: J, P | Grupa 7: R |
| Grupa 8: L | Grupa 8: E, F, P | Grupa 8: N, R | Grupa 8: E, F, K, Z |
| Grupa 9: C, O | Grupa 9: J, Z | Grupa 9: A, H | Grupa 9: P |
| Grupa 10: I, Z | Grupa 10: K | Grupa 10: D ,O |  |
| Grupa 11: F | Grupa 11: Y |  |  |
| 0,005 | Grupa 1: A | Grupa 1: I | Grupa 1: A, H, P, R | Grupa 1: B, C, D, O |
| Grupa 2: R | Grupa 2: T | Grupa 2: J | Grupa 2: A |
| Grupa 3: P | Grupa 3: C, D, O | Grupa 3: K, N | Grupa 3: T |
| Grupa 4: G, Z | Grupa 4: Y | Grupa 4: Y | Grupa 4: J, Y |
| Grupa 5: B, C, D, O | Grupa 5: G, U | Grupa 5: B | Grupa 5: E, R, Z |
| Grupa 6: F | Grupa 6: L | Grupa 6: F, T | Grupa 6: H, N, U |
| Grupa 7: E | Grupa 7: A | Grupa 7: Z | Grupa 7: G |
| Grupa 8: H, K, N, U | Grupa 8: B | Grupa 8: E, G, L | Grupa 8: L |
| Grupa 9: L | Grupa 9: J | Grupa 9: U | Grupa 9: P |
| Grupa 10: T, Y | Grupa 10: H, P, R | Grupa 10: I | Grupa 10: F |
| Grupa 11: I, J | Grupa 11: K, N | Grupa 11: O | Grupa 11: K |
|  | Grupa 12: E, F, Z | Grupa 12: C | Grupa 12: I |
|  |  | Grupa 13: D |  |
| 0,01 | Grupa 1: F, H, N, P, Y | Grupa 1: C, D, O | Grupa 1: B, C, D, O | Grupa 1: B, C, D, O |
| Grupa 2: R | Grupa 2: B, J | Grupa 2: G, L, U | Grupa 2: J |
| Grupa 3: A | Grupa 3: G | Grupa 3: H, N | Grupa 3: G, L, U |
| Grupa 4: B | Grupa 4: A | Grupa 4: A, P, R | Grupa 4: A |
| Grupa 5: C, D, O | Grupa 5: E | Grupa 5: F | Grupa 5: H, N |
| Grupa 6: U | Grupa 6: F, P, R | Grupa 6: E | Grupa 6: P |
| Grupa 7: L | Grupa 7: H | Grupa 7: T, Y | Grupa 7: F, R |
| Grupa 8: E | Grupa 8: N | Grupa 8: J | Grupa 8: E |
| Grupa 9: K | Grupa 9: U | Grupa 9: I, Y, Z | Grupa 9: K |
| Grupa 10: G | Grupa 10: Z |  | Grupa 10: I |
| Grupa 11: Z | Grupa 11: K, L |  | Grupa 11: Y |
| Grupa 12: J, T | Grupa 12: Y |  | Grupa 12: T |
| Grupa 13: I | Grupa 13: I, T |  | Grupa 13: Z |
| 0,9 | Grupa 1: B, C, D, G, L, O, U | Grupa 1: C, D, L, O, U | Grupa 1: B, C, D, G, L, O | Grupa 1: B, C, D, G, L, O |
| Grupa 2: E, G | Grupa 2: N, U | Grupa 2: A, U |
| Grupa 2: H | Grupa 3: A, B | Grupa 3: H | Grupa 3: H, N |
| Grupa 3: A, R | Grupa 4: P | Grupa 4: A | Grupa 4: K |
| Grupa 4: P | Grupa 5: F, R | Grupa 5: P, R | Grupa 5: R |
| Grupa 5: F | Grupa 6: K | Grupa 6: F | Grupa 6: P |
| Grupa 6: E | Grupa 7: N | Grupa 7: K | Grupa 7: F |
| Grupa 7: Z | Grupa 8: H | Grupa 8: E | Grupa 8: E |
| Grupa 8: J | Grupa 9: J | Grupa 9: Y | Grupa 9: Z |
| Grupa 9: I | Grupa 10: Z | Grupa 10: T | Grupa 10: J |
| Grupa 10: T | Grupa 11: I | Grupa 11: I | Grupa 11: I |
| Grupa 11: Y | Grupa 12: T | Grupa 12: J | Grupa 12: T |
| Grupa 12: K | Grupa 13: Y | Grupa 13: Z | Grupa 13: Y |
| Grupa 13: N |  |  |  |
| 0,1 | Grupa 1: B, C, D ,O | Grupa 1: B, C, D, L, O | Grupa 1: B, C, D, L, O | Grupa 1: B, C, D, O |
| Grupa 2: L, U | Grupa 2: U | Grupa 2: H, U | Grupa 2: L, U |
| Grupa 3: K, N | Grupa 3: K, N | Grupa 3: N | Grupa 3: K |
| Grupa 4: H | Grupa 4: H | Grupa 4: K | Grupa 4: H, N |
| Grupa 5: A, R | Grupa 5: A, R | Grupa 5: R | Grupa 5: A |
| Grupa 6: F, P | Grupa 6: P | Grupa 6: A | Grupa 6: R |
| Grupa 7: E | Grupa 7: F | Grupa 7: F, P | Grupa 7: F, P |
| Grupa 8: G | Grupa 8: E, G | Grupa 8: E, G | Grupa 8: E, G |
| Grupa 9: J, Z | Grupa 9: Z | Grupa 9: Z | Grupa 9: Z |
| Grupa 10: I | Grupa 10: J | Grupa 10: J | Grupa 10: J |
| Grupa 11: Y | Grupa 11: I | Grupa 11: Y | Grupa 11: I |
| Grupa 12: T | Grupa 12: T | Grupa 12: T | Grupa 12: T |
|  | Grupa 13: Y | Grupa 13: I | Grupa 13: Y |

Tab 5. Zestawienie poprawnych odpowiedzi sieci w procesie testowania dla różnych wartości współczynnika uczenia

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Wartość współczynnika uczenia | Procent poprawnych odpowiedzi | | | | |
|
| Próba 1 | Próba 2 | Próba 3 | Próba 4 | Średnia |
| 0,0001 | 69,64% | 64,64% | 76,87% | 70,12% | 70,32% |
| 0,001 | 68,21% | 65,39% | 69,32% | 80,89% | 70,95% |
| 0,005 | 74,80% | 84,21% | 76,71% | 65,64% | 75,34% |
| 0,01 | 68,69% | 79,98% | 76,64% | 75,64% | 75,24% |
| 0,09 | 75,50% | 82,39% | 78,09% | 67,32% | 75,83% |
| 0,1 | 79,64% | 78,98% | 88,14% | 92,96% | 84,93% |

1. **Analiza wyników**

Na podstawie tabeli numer 3 widzimy, iż przy niższej wartości współczynnika uczenia różnice pomiędzy neuronem wygrywającym w pierwszej epoce, a neuronem wygrywającym w ostatniej epoce są niewielkie. Zazwyczaj tylko jeden neuron przestaje być neuronem wygrywającym dla danego zestawu uczącego. Na podstawie tego można stwierdzić, iż przy tym współczynniku sieć potrzebuje o wiele więcej epok, aby nauczyć się lepiej grupować litery oraz to że w przypadku takiej sieci jej zdolność do grupowania liter jest tak naprawdę losowa, ponieważ jest ściśle powiązana z początkowym doborem wag, który jest losowy. W miarę zwiększania się współczynnika uczenia zmienia się także większa ilość neuronów wygrywających, co pozwala stwierdzić, iż w tych przypadkach grupowanie liter jest mniej losowe i jest bardziej zależne od faktycznych zdolności sieci, wag neuronów i wcześniejszych danych, na których sieć ta się uczy. Pozwala to także zaoszczędzić wiele czasu oraz mocy obliczeniowej, ponieważ proces uczenia sieci z odpowiednio wyższym współczynnikiem uczenia przebiega szybciej – składa się z mniejszej ilości epok, a nawet wydłużenie uczenia sieci o niższym współczynniku uczenia nie gwarantuje uzyskania efektywności sieci z wyższym współczynnikiem, czasem może prowadzić nawet do przeuczenia i pogorszenia poprawności odpowiedzi sieci.

Wyniki przestawione w tabeli numer 4 pokazują na jakie grupy sieć neuronowa dzieliła przygotowany przeze mnie alfabet w poszczególnych próbach dla różnych wartości współczynnika uczenia. Zebrane wyniki obrazują prace programu, jego wyniki oraz zdolność mojej sieci do grupowania liter. Możemy zauważyć, iż w przypadku niższego współczynnika grupy bardziej odbiegają od tych przedstawionych przeze mnie w tabeli numer 2, mimo to jednak cechują się one odwzorowaniem na dostatecznym poziomie, czasami jedynie niewiele odbiegając od prób z wyższym współczynnikiem uczenia. Co jednak odróżnia próby przy niższym współczynniku uczenia od tych z wyższym jest fakt, iż sieć z wyższym współczynnikiem uczenia lepiej radzi sobie z literami, które są jedynymi w swojej grupie. Pozwala to stwierdzić, iż taka sieć jest lepiej przystosowana do wyszukiwania cech charakterystycznych, wyróżniających litery spośród innych. Sieć z wyższym współczynnikiem uczenia lepiej radzi sobie także z przypadkami, gdy dana grupa złożona jest z większej ilości potrafiąc wskazać wszystkich jej reprezentantów jednocześnie nie dołączając do niej żadnej innej litery.

Wyniki przestawione w tabeli numer 5 możemy zobaczyć, jak zmieniała się ilość poprawnych odpowiedzi dla każdej z prób z osobna oraz jak przedstawia się wartość średnia dla danego współczynnika uczenia. Widzimy tutaj, iż procent poprawnych odpowiedzi może się wahać nawet w granicach 10% dla poszczególnych współczynników uczenia co jest wartością bardzo dużą. Jedną z głównych przyczyn takiego stanu rzeczy są dobierane w sposób losowy pierwsze wagi dla neuronów. Szczególnie duże znaczenie ma to w przypadku sieci z niskim współczynnikiem uczenia, gdzie to wagi neuronów zmieniają się co epokę o małe wartości, w wyniku czego neurony które wygrywały przez pierwszych epokach wygrywają dalej nawet już w mocno zaawansowanym czasowo procesie uczenia. Patrząc na uśrednione wartości poprawnych odpowiedzi widzimy, iż utworzyły się tutaj 3 grupy obejmujące współczynniki: grupa pierwsza ze współczynnikami: 0,0001; 0,001; 0,005, grupa druga ze współczynnikami: 0,01; 0,09 oraz grupa trzecia ze współczynnikiem 0,1. W przypadku pierwszej z nich mówimy tutaj o poprawności odpowiedzi oscylującej na poziomie 70% co jest rezultatem zadowalającym, pozwalającym jednak stwierdzić, iż tak przygotowana sieć jest w stanie poradzić sobie z postawionym jej zadaniem. Zmiana współczynnika 10- krotnie nie przyniosła jednak dużych zmian efektywności sieci, które w tym przypadku wynoszą jedynie dziesiąte części procenta. W przypadku drugiej grupy mówimy o 5% wzroście poprawności odpowiedzi w porównaniu z grupą pierwszą. Współczynniki tutaj zostały powiększone od 5 do nawet ponad 10 krotnie w porównaniu z grupą pierwszą co dało większe rezultaty. Dalsze zwiększanie tego współczynnika nie przyniosło jednak oczekiwanych rezultatów, gdyż widzimy, iż nie przekłada się to na lepszą efektywność, która to utrzymuje się na stałej wartości 75% i w dodatku potrafi lekko spaść podczas testowania sieci z wyższym współczynnikiem. Dopiero kolejne podwyższenie współczynnika uczenia do poziomu 0,1 zaowocowało o wiele większą skutecznością sieci, największa spośród wszystkich oscylującą w tym przypadku na poziomie 85%. Pomimo małej zmianie wielkości współczynnika w porównaniu z wartością wcześniejszą, udało się tutaj uzyskać wzrost efektywności o niemal 10% co jest wartością bardzo dużą. Pozwala to stwierdzić, iż czasem nawet minimalna zmiana współczynnika uczenia pozwala uzyskać często niespodziewanie wręcz lepsze rezultaty, co oznacza, iż projektując sieć należy poświęcić wiele czasu na testy i odpowiedni dobór tego parametru.

Wykres 1. Wartości odległości euklidesowej podczas 1 próby procesu testowania i współczynnika uczenia równego 0,0001

Wykres 2. Wartości odległości euklidesowej podczas 2 próby procesu testowania i współczynnika uczenia równego 0,0001

Wykres 3. Wartości odległości euklidesowej podczas 3 próby procesu testowania i współczynnika uczenia równego 0,0001

Wykres 4. Wartości odległości euklidesowej podczas 4 próby procesu testowania i współczynnika uczenia równego 0,0001

Powyższe cztery wykresy odnoszą się do poszczególnych prób testowania sieci dla współczynnika uczenia równego 0,0001. Zbliżone wartości odległości euklidesowych pozwalają rozpoznać punkty które zostały przez sieć zakwalifikowane do tej samej grupy.

Wykres 5. Wartości odległości euklidesowej podczas 1 próby procesu testowania i współczynnika uczenia równego 0,001

Wykres 6. Wartości odległości euklidesowej podczas 2 próby procesu testowania i współczynnika uczenia równego 0,001

Wykres 7. Wartości odległości euklidesowej podczas 3 próby procesu testowania i współczynnika uczenia równego 0,001

Wykres 8. Wartości odległości euklidesowej podczas 4 próby procesu testowania i współczynnika uczenia równego 0,001

Powyższe cztery wykresy odnoszą się do poszczególnych prób testowania sieci dla współczynnika uczenia równego 0,001. Zbliżone wartości odległości euklidesowych pozwalają rozpoznać punkty które zostały przez sieć zakwalifikowane do tej samej grupy.

Wykres 9. Wartości odległości euklidesowej podczas 1 próby procesu testowania i współczynnika uczenia równego 0,005

Wykres 10. Wartości odległości euklidesowej podczas 2 próby procesu testowania i współczynnika uczenia równego 0,005

Wykres 11. Wartości odległości euklidesowej podczas 3 próby procesu testowania i współczynnika uczenia równego 0,005

Wykres 12. Wartości odległości euklidesowej podczas 4 próby procesu testowania i współczynnika uczenia równego 0,005

Powyższe cztery wykresy odnoszą się do poszczególnych prób testowania sieci dla współczynnika uczenia równego 0,005. Zbliżone wartości odległości euklidesowych pozwalają rozpoznać punkty które zostały przez sieć zakwalifikowane do tej samej grupy.

Wykres 13. Wartości odległości euklidesowej podczas 1 próby procesu testowania i współczynnika uczenia równego 0,01

Wykres 14. Wartości odległości euklidesowej podczas 2 próby procesu testowania i współczynnika uczenia równego 0,01

Wykres 15. Wartości odległości euklidesowej podczas 3 próby procesu testowania i współczynnika uczenia równego 0,01

Wykres 16. Wartości odległości euklidesowej podczas 4 próby procesu testowania i współczynnika uczenia równego 0,01

Powyższe cztery wykresy odnoszą się do poszczególnych prób testowania sieci dla współczynnika uczenia równego 0,01. Zbliżone wartości odległości euklidesowych pozwalają rozpoznać punkty które zostały przez sieć zakwalifikowane do tej samej grupy.

Wykres 17. Wartości odległości euklidesowej podczas 1 próby procesu testowania i współczynnika uczenia równego 0,09

Wykres 18. Wartości odległości euklidesowej podczas 2 próby procesu testowania i współczynnika uczenia równego 0,09

Wykres 19. Wartości odległości euklidesowej podczas 3 próby procesu testowania i współczynnika uczenia równego 0,09

Wykres 20. Wartości odległości euklidesowej podczas 4 próby procesu testowania i współczynnika uczenia równego 0,09

Powyższe cztery wykresy odnoszą się do poszczególnych prób testowania sieci dla współczynnika uczenia równego 0,09. Zbliżone wartości odległości euklidesowych pozwalają rozpoznać punkty które zostały przez sieć zakwalifikowane do tej samej grupy.

Wykres 21. Wartości odległości euklidesowej podczas 1 próby procesu testowania i współczynnika uczenia równego 0,1

Wykres 22. Wartości odległości euklidesowej podczas 2 próby procesu testowania i współczynnika uczenia równego 0,1

Wykres 23. Wartości odległości euklidesowej podczas 3 próby procesu testowania i współczynnika uczenia równego 0,1

Wykres 24. Wartości odległości euklidesowej podczas 4 próby procesu testowania i współczynnika uczenia równego 0,1

Powyższe cztery wykresy odnoszą się do poszczególnych prób testowania sieci dla współczynnika uczenia równego 0,1. Zbliżone wartości odległości euklidesowych pozwalają rozpoznać punkty które zostały przez sieć zakwalifikowane do tej samej grupy.

1. **Wnioski**

* Efektywność sieci Kohonena opartej na regule WTM, jej zdolność do wskazywania poprawnych odpowiedzi jest powiązana z jej budową. Po wielokrotnych testach różnych konfiguracji stwierdzono, że dla powyższego problemu najlepszą skuteczność przy jednoczesnym małym rozbudowaniu sieci uzyskano przy ilości neuronów równej 20.
* Efektywność procesu uczenia dla sieci neuronowej jest silnie powiązana z wartością współczynnika uczenia sieci. Należy na podstawie wielokrotnych testów dobrać takie wartości współczynników, aby uniknąć bardzo powolnego procesu uczenia, ale też, należy uważać, żeby nasza sieć się nie przeuczyła, a bardzo szybka modyfikacja wag nie prowadziła do błędnych odpowiedzi sieci.
* Sieć Kohonena oparta o regułę WTM w odróżnieniu od tej opartej o regułę WTA sprawia, iż pomimo dużej liczby neuronów wiele z nich pozostaje aktywnych i bierze aktywny udział w rozwiązywaniu danego zadania w przeciwieństwie do metody WTA gdzie tylko jeden neuron modyfikował swoje wagi. Neurony, które nie zwyciężyły nie modyfikują swoich wag w tak dużym stopniu jak neuron zwycięski, jednak mimo to są one w stanie realnie uczestniczyć w procesie uczenia czy testowania danych i nie pozostają wyłączone jak przy metodzie WTA.
* Wraz ze zwiększająca się wartością współczynnika uczenia możemy zaobserwować, iż sieć nabiera umiejętności grupowania poszczególnych liter do grup jednoelementowych, dzięki czemu sieć staje się bardziej wrażliwa na nawet małe różnice pomiędzy literami. Należy jednak uważać, gdyż przy zbyt wysokim współczynniku uczenia sieć traci tę zdolność i zaczyna większość liter umieszczać w tej samej grupie. Dlatego też wartość współczynnika uczenia jak i liczbę epok uczących należy dobierać indywidualnie w zależności od rozwiązywanego problemu.

1. **Kod programu**

**Layer.h:**

#pragma once

#include<vector>

#include"Neuron.h"

using namespace std;

class Layer

{

public:

Layer(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double numberOfIterations);

vector<Neuron> neurons;

int getNumberOfNeurons() { return \_numberOfNeurons; }

void changeNeuronWeight(double currentIteration, bool testing);

double getScalarProduct(int index) { return \_scalarProducts[index]; }

void gatherScalarProducts();

int getWinnerNeuronIndex() { return \_winnerNeuronIndex; }

private:

void foundMaxScalarProduct();

int \_numberOfNeurons;

vector<double> \_scalarProducts;

int \_winnerNeuronIndex;

double \_mapRadius;

double \_timeConstant;

};

**Layer.cpp:**

#include"Layer.h"

Layer::Layer(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double numberOfIterations)

{

\_numberOfNeurons = numberOfNeurons;

neurons.resize(numberOfNeurons);

\_mapRadius = (double)numberOfNeurons;

\_timeConstant = numberOfIterations / \_mapRadius;

for (int i = 0; i < numberOfNeurons; i++)

neurons[i].Neuron::Neuron(amountOfDendrites, amountOfOutputs, learningCoefficient);

}

void Layer::changeNeuronWeight(double currentIteration, bool testing)

{

gatherScalarProducts();

foundMaxScalarProduct();

neurons[\_winnerNeuronIndex].processOutput();

if (testing == false)

{

neurons[\_winnerNeuronIndex].countNeighbourhoodRadius(\_mapRadius, currentIteration, \_timeConstant);

int radius = neurons[\_winnerNeuronIndex].getNeighbourhoodRadius();

int leftBorderNeuronIndex = 0;

int rightBorderNeuronIndex = 0;

if (\_winnerNeuronIndex - radius < 0)

leftBorderNeuronIndex = 0;

else

leftBorderNeuronIndex = \_winnerNeuronIndex - radius;

if (\_winnerNeuronIndex + radius >= \_numberOfNeurons)

rightBorderNeuronIndex = \_numberOfNeurons - 1;

else

rightBorderNeuronIndex = \_winnerNeuronIndex + radius;

for (int i = leftBorderNeuronIndex; i < rightBorderNeuronIndex; i++)

{

neurons[i].setDistance((i < \_winnerNeuronIndex) ? (\_winnerNeuronIndex - i) : (i - \_winnerNeuronIndex));

neurons[i].setNeighbourhoodRadius(neurons[\_winnerNeuronIndex].getNeighbourhoodRadius());

neurons[i].countNewWeights();

}

}

}

void Layer::gatherScalarProducts()

{

\_scalarProducts.clear();

for (int i = 0; i < \_numberOfNeurons; i++)

\_scalarProducts.push\_back(neurons[i].countScalarProduct());

}

void Layer::foundMaxScalarProduct()

{

double tmp = \_scalarProducts[0];

\_winnerNeuronIndex = 0;

for (int i = 1; i < \_scalarProducts.size(); i++)

{

if (tmp > \_scalarProducts[i])

{

\_winnerNeuronIndex = i;

tmp = \_scalarProducts[i];

}

}

}

**Neuron.h:**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Neuron

{

public:

void createInputs(int amountOfDendrites, int amountOfSynapses);

void createInput() { \_inputs.push\_back(0); }

void createWeight(int index) { \_weights.push\_back(0); }

int getInputsAmount() { return \_inputs.size(); }

int getWeightsAmount() { return \_weights.size(); }

double getInput(int index) { return \_inputs[index]; }

void setInput(int index, double value) { \_inputs[index] = value; }

double getSynapse(int index) { return \_weights[index]; }

void setSynapse(int index, double value) { \_weights[index] = value; }

double getSumOfAllInputs() { return \_sumOfAllInputs; }

double getOutputValue() { return \_outputValue; }

double processSingleInput(int index) { return \_inputs[index] \* \_weights[index]; }

void processOutput();

void countNewWeights();

double countScalarProduct();

void countNeighbourhoodRadius(double mapRadius, double currentIteration, double timeConstant);

double getNeighbourhoodRadius() { return \_neighbourhoodRadius; }

void setNeighbourhoodRadius(double neighbourhoodRadius) {\_neighbourhoodRadius = neighbourhoodRadius; }

void setDistance(double distance) { \_distance = distance; }

Neuron();

Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient);

private:

double countFirstWeight();

void countNeighbourhoodFunction();

void normalizeWeights();

vector<double> \_inputs;

vector<double> \_weights;

double \_sumOfAllInputs;

double \_outputValue;

double \_learningCoefficient;

double \_neighbourhoodFunctionValue;

double \_distance;

double \_neighbourhoodRadius;

};

**Neuron.cpp:**

#include "Neuron.h"

#include <time.h>

#include <math.h>

Neuron::Neuron()

{

\_inputs.resize(0);

\_weights.resize(0);

\_sumOfAllInputs = 0.0;

\_outputValue = 0.0;

\_learningCoefficient = 0.0;

}

Neuron::Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient)

{

createInputs(amountOfDendrites, amountOfOutputs);

normalizeWeights();

\_learningCoefficient = learningCoefficient;

\_sumOfAllInputs = 0.0;

\_outputValue = 0.0;

}

void Neuron::createInputs(int amountOfDendrites, int amountOfOutputs)

{

for (int j = 0; j < amountOfDendrites; j++)

{

\_inputs.push\_back(0);

\_weights.push\_back(countFirstWeight());

}

}

double Neuron::countScalarProduct()

{

\_sumOfAllInputs = 0.0;

for (int i = 0; i < getInputsAmount(); i++)

\_sumOfAllInputs += pow(\_inputs[i] - \_weights[i],2);

\_sumOfAllInputs = sqrt(\_sumOfAllInputs);

return \_sumOfAllInputs;

}

void Neuron::processOutput()

{

double beta = 1.0;

\_outputValue = (1.0 / (1.0 + (exp(-beta \* \_sumOfAllInputs))));

}

void Neuron::countNewWeights()

{

countNeighbourhoodFunction();

for (int i = 0; i < getWeightsAmount(); i++)

\_weights[i] = \_weights[i] + (\_learningCoefficient \* \_neighbourhoodFunctionValue \* (\_inputs[i] - \_weights[i]));

normalizeWeights();

}

void Neuron::countNeighbourhoodRadius(double mapRadius, double currentIteration, double timeConstant)

{

\_neighbourhoodRadius = mapRadius \* exp(-currentIteration / timeConstant);

}

//private methods

void Neuron::countNeighbourhoodFunction()

{

\_neighbourhoodFunctionValue = exp(pow(-\_distance,2) / (2 \* pow(\_neighbourhoodRadius,2)));

}

double Neuron::countFirstWeight()

{

double max = 1.0;

double min = 0.0;

double weight = ((double(rand()) / double(RAND\_MAX)) \* (max - min)) + min;

return weight;

}

void Neuron::normalizeWeights()

{

double vectorLength = 0.0;

for (int i = 0; i < getWeightsAmount(); i++)

vectorLength += pow(\_weights[i], 2);

vectorLength = sqrt(vectorLength);

for (int i = 0; i < getWeightsAmount(); i++)

\_weights[i] /= vectorLength;

}

**main.cpp:**

#include <iostream>

#include <vector>

#include "Layer.h"

#include <time.h>

#include <fstream>

using namespace std;

void setInputValue(Neuron& neuron, vector< vector<double> > inputData, int numberOfEntrances, int inputDataRow);

void learn(Layer& layer, vector< vector<double> > inputData);

void test(Layer& layer, vector< vector<double> > inputData);

void readLearningData(vector< vector<double> > &learningInputData, int numberOfEntrances, int amountOfOutputs);

void readTestingData(vector< vector<double> > &testingInputData, int numberOfEntrances, int amountOfOutputs);

fstream OUTPUT\_LEARNING\_FILE;

fstream OUTPUT\_TESTING\_FILE;

fstream OUTPUT\_TESTING\_NEURON;

fstream LEARNING\_DATA;

fstream TESTING\_DATA;

int main()

{

srand(time(NULL));

vector< vector<double> > learningInputData;

vector< vector<double> > testingInputData;

int numberOfNeurons = 20;

int numberOfEntrances = 20;

int numberOfOutputs = 1;

double learningCoefficient = 0.1;

int numberOfEpochs = 200;

Layer kohonenNetwork(numberOfNeurons, numberOfEntrances, numberOfOutputs, learningCoefficient, (double)numberOfEpochs);

readLearningData(learningInputData, numberOfEntrances, numberOfOutputs);

readTestingData(testingInputData, numberOfEntrances, numberOfOutputs);

do

{

cout << "1. Learn" << endl;

cout << "2. Test" << endl;

cout << "3. Exit" << endl;

int choice;

cin >> choice;

switch (choice)

{

case 1:

OUTPUT\_LEARNING\_FILE.open("output\_learning\_data.txt", ios::out);

for (int epochNumber = 1, i = 0; i < numberOfEpochs; i++, epochNumber++)

{

learn(kohonenNetwork, learningInputData);

if (epochNumber == 1 || epochNumber == 200)

OUTPUT\_LEARNING\_FILE << "EPOCH NUMBER: " << epochNumber << endl;

}

break;

case 2:

OUTPUT\_TESTING\_FILE.open("output\_testing\_data.txt", ios::out);

OUTPUT\_TESTING\_NEURON.open("output\_testing\_neuron.txt", ios::out);

test(kohonenNetwork, testingInputData);

break;

case 3:

OUTPUT\_LEARNING\_FILE.close();

OUTPUT\_TESTING\_FILE.close();

return 0;

}

} while (true);

return 0;

}

void setInputValue(Neuron& neuron, vector< vector<double> > inputData, int numberOfEntrances, int row)

{

for (int i = 0; i < numberOfEntrances; i++)

neuron.setInput(i, inputData[row][i]);

}

void learn(Layer& layer, vector< vector<double> > inputData)

{

static int currentIteration = 1;

for (int inputDataRow = 0; inputDataRow < inputData.size(); inputDataRow++)

{

for (int i = 0; i < layer.getNumberOfNeurons(); i++)

{

setInputValue(layer.neurons[i], inputData, layer.neurons[i].getInputsAmount(), inputDataRow);

layer.neurons[i].countScalarProduct();

}

layer.changeNeuronWeight(currentIteration, false);

if (currentIteration == 1 || currentIteration == 200)

OUTPUT\_LEARNING\_FILE << inputDataRow << ": " << layer.getWinnerNeuronIndex() << endl;

}

currentIteration++;

}

void test(Layer& layer, vector< vector<double> > inputData)

{

for (int inputDataRow = 0, myDictionary = 0; inputDataRow < inputData.size(); inputDataRow++)

{

for (int i = 0; i < layer.getNumberOfNeurons(); i++)

{

setInputValue(layer.neurons[i], inputData, layer.neurons[i].getInputsAmount(), inputDataRow);

layer.neurons[i].countScalarProduct();

}

if (inputDataRow == 12 || inputDataRow == 15 || inputDataRow == 16)

myDictionary++;

if (inputDataRow == 18)

myDictionary += 3;

char letter = 'A';

layer.changeNeuronWeight(0, true);

OUTPUT\_TESTING\_FILE << layer.neurons[layer.getWinnerNeuronIndex()].getSumOfAllInputs() << endl;

OUTPUT\_TESTING\_NEURON << static\_cast<char>(letter + inputDataRow + myDictionary) << " " << layer.getWinnerNeuronIndex() << endl;

}

}

void readLearningData(vector< vector<double> > &learningInputData, int numberOfEntrances, int amountOfOutputs)

{

LEARNING\_DATA.open("learning\_data.txt", ios::in);

vector<double> row;

do

{

row.clear();

for (int i = 0; i < numberOfEntrances; i++)

{

double inputTmp = 0.0;

LEARNING\_DATA >> (double)inputTmp;

row.push\_back(inputTmp);

}

double vectorLength = 0.0;

for (int i = 0; i < numberOfEntrances; i++)

vectorLength += pow(row[i], 2);

vectorLength = sqrt(vectorLength);

for (int i = 0; i < numberOfEntrances; i++)

row[i] /= vectorLength;

learningInputData.push\_back(row);

} while (!LEARNING\_DATA.eof());

LEARNING\_DATA.close();

}

void readTestingData(vector< vector<double> > &testingInputData, int numberOfEntrances, int amountOfOutputs)

{

TESTING\_DATA.open("testing\_data.txt", ios::in);

vector<double> row;

while (!TESTING\_DATA.eof())

{

row.clear();

for (int i = 0; i < numberOfEntrances; i++)

{

double inputTmp = 0.0;

TESTING\_DATA >> (double)inputTmp;

row.push\_back(inputTmp);

}

double vectorLength = 0.0;

for (int i = 0; i < numberOfEntrances; i++)

vectorLength += pow(row[i], 2);

vectorLength = sqrt(vectorLength);

for (int i = 0; i < numberOfEntrances; i++)

row[i] /= vectorLength;

testingInputData.push\_back(row);

}

TESTING\_DATA.close();

}