Michał Ligęza, gr lab 4, IS

Temat​ ​ ćwiczenia: **Budowa​ ​ i​ ​ działanie​ ​ perceptronu**

Implementacja perceptronu i nauczenie funkcji AND

1. **Opis budowy perceptronu (pseudokod)**utworzyć Perceptron (ilośćWejść=2, współczynnikNauki)  
   przygotować zestawy uczące //funkcja AND  
   dla zadanej ilości powtórzeń  
    dla każdego zestawu  
    ucz Perceptron   
   utwórz zestawy testujące  
   zbadaj prawidłowość działania

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Wsp** | **N1** | **N2** | **N3** | **N4** | **Średnio** |
| 0.1 | 58 | 58 | 61 | 56 | 58,25 |
| 0.3 | 19 | 19 | 18 | 20 | 19 |
| 0.5 | 11 | 12 | 11 | 11 | 11,25 |
| 0.7 | 8 | 8 | 8 | 8 | 8 |
| 0.9 | 6 | 6 | 6 | 6 | 6 |

1. **Wyniki**Program przetestowano dla 4 różnych wartości współczynnikNauki, każdy przetestowano na 4 różnych zadanych ilości powtórzeń, po 4 razy dla uzyskania wyniku statystycznego
2. **Analiza**Przed rozpoczęciem analizy, warto zauważyć iż podane wyniki to liczba powtórzeń nauki zestawów uczenia funkcji AND – w sumie czterech możliwości - dlatego by otrzymać rzeczywistą ilość nauczanych zestawów, należałoby otrzymane wyniki przemnożyć przez 4. Zmiana wartości współczynnika nauki z 0.1 na 0.3 (trzykrotne podniesienie wartości) pozwoliło trzykrotnie zmniejszyć czas uczenia sieci. Kolejne trzykrotne zwiększenie współczynnika z 0.3 na 0.9 pozwoliło na kolejne trzykrotne zmniejszenie czasu uczenia, jedna porównując wartości (poprzednia różnica 0.2 wywołała zmianę o ok 40 iteracji, dalsza zmiana o 0.6 wywołała zmianę o 13 iteracji). Wykres przypomina także wyglądem interpolację funkcji geometrycznej.
3. **Wnioski**Nawet nieznaczne zmniejszenie wartości współczynnika nauki blisko 0.0 wywoła duże zwiększenie czasu potrzebnego na naukę. Zwiększanie współczynnika blisko 1.0 poprawia czas nauki nieznacznie.
4. **Listing kodu**class Perceptron {

public:

double learningRate; // współczynnikNauki

vector<double> weights; //wagi, ustawione losowo

Perceptron(int numWeights, double rate); //konstruktor (ilość wejść i wsp nauki)

double guess(vector<double> inputs); //oblicza wartośc wyjścia dla obecnych wag

void train(vector<double> inputs, double target); //uczy się wyniku

};  
  
  
int main() {

srand(time(NULL)); //ustawienie losowości

Perceptron \*p = new Perceptron(2, (double)0.5); //stworzenie Perceptronu

int ile = 20; //ilość powtórzeń nauki zestawów

vector<double> labels = {0,0,0,1};//wyniki do testów

vector<vector<double>> inputs = { {0,0},{0,1},{1,0},{1,1} };//dane

for(int j=0;j<ile;j++){

for (int i = 0; i < labels.size(); i++) {

p->train(inputs[i], labels[i]);//nauczanie perceptrony

}

}

vector<double> n1 = { 0,0 };

vector<double> n2 = { 1,1 };

cout << "n1: " << p->guess(n1) << endl << "n2:" << p->guess(n2) << endl;

//badanie poprawności nauczania

system("pause");

}