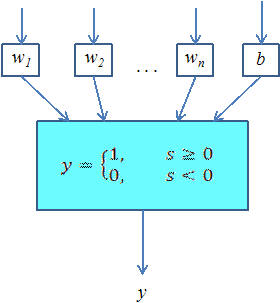
**PSI**

**Sprawozdanie z zadania nr 1.**

**Celem ćwiczenia** było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptrony realizującego wybraną funkcję logiczną dwóch zmiennych.

**Wstęp do zagadnienia:**

Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuje tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).



**Algorytm uczenia perceptronu** - tzn. automatycznego doboru wag na podstawie napływajacych przykładów (uczących). W uproszczonym przypadku dwuwymiarowym algorytm wygląda następująco:

1. Inicjujemy wagi losowo.

2. Dla każdego przykładu uczącego obliczamy odpowiedź perceptronu.

3. Jeśli odpowiedź perceptronu jest nieprawidłowa, to modyfikujemy wagi:

w1 += n \* (d-y) \* x1

w2 += n \* (d-y) \* x2

b += n \* (d-y)

gdzie b to bias, n jest niewielkim współczynnikiem uczenia (1>n > 0), d - oczekiwana odpowiedź a y - odpowiedź neuronu. Po wyczerpaniu przykładów zaczynamy proces uczenia od początku, dopóki następują jakiekolwiek zmiany wag połączeń.

**Opis wykonanych kroków w toku realizacji zadania**:

1. Zaimplementowano neuron według instrukcji podanej na wykładzie ( wejścia, wagi dla wejść, blok sumujący, blok aktywacji itd.)
2. Wygenerowano dane uczące dla bramki logicznej OR (kod java) oraz dla pozostałych bramek (kod cpp).
3. Wykonano etap uczenia perceptronu dla różnej ilości danych uczących oraz współczynników uczenia „n”.
4. Przeprowadzono testy działania .

**Zestawienie rezultatów dla przykładowej ilości danych uczących (na przykładzie bramki OR):**

Gdy podawano niepełny zestaw danych uczących to odpowiedź perceptronu nie zawsze była prawidłowa( losowo działała)

|  |  |  |
| --- | --- | --- |
| Przykładowe dane uczące | | |
| A | B | Out |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

Dla takiego zestawu danych uczących odpowiedz perceptronu dla kombinacji zmiennych{0,1} {1,0}  
często była przypadkowa.   
Z tego powodu kolejne dane uczące jakie dostarczałem były już pełniejsze.

Np.

|  |  |  |
| --- | --- | --- |
| Przykładowe dane uczące | | |
| A | B | Out |
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

Lub dla kodu w cpp

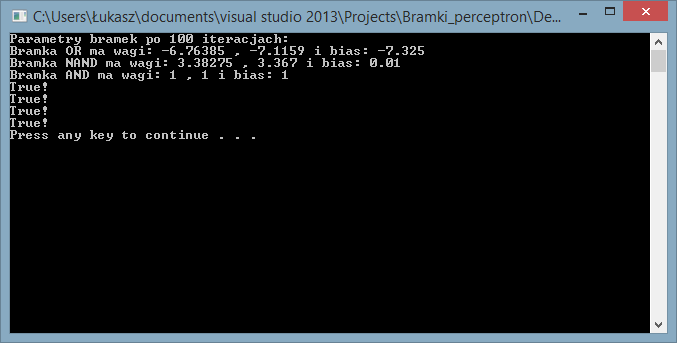
|  |  |  |
| --- | --- | --- |
| Przykładowe dane uczące | Przykładowe dane uczące | Przykładowe dane uczące |
| A | A | A |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 0.22 | 0.12 | 0 |
| 0.0 | 1.1 | 1 |
| 1.1 | 1.1 | 1 |
| 1.8 | 0.1 | 1 |

Na tym etapie można już sformułować pierwszy wniosek , że im więcej danych uczących dostarczymy, tym odpowiedź perceptronu będzie pewniejsza gdyż jednostka zna więcej przypadków i na ich podstawie wie jak je interpretować.

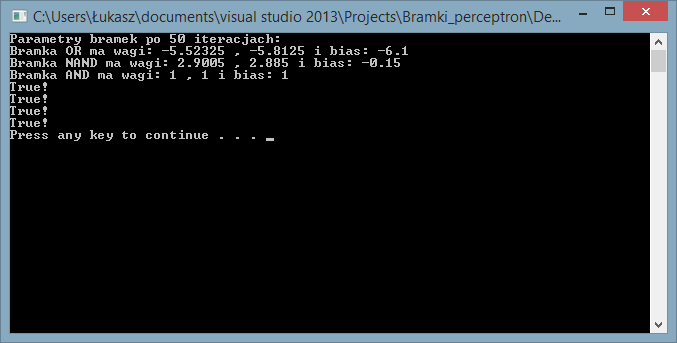
**Zestawienie rezultatów w przypadku zmiany współczynnika uczenia:**

Zbyt mała wartość tego współczynnika powoduje zwiększenie się liczby potrzebnych iteracji i tym samym czasu wymaganego na zakończenie treningu. Algorytm ma ponadto w takim przypadku tendencję to wygasania w minimach lokalnych funkcji celu. Ustalenie z kolei zbyt dużej wartości kroku grozi wystąpieniem oscylacji wokół poszukiwanego minimum, prowadzącej ostatecznie do zatrzymania procesu uczenia bez osiągniecie wymaganej wartości funkcji celu.  
Mówiąc prościej wartość współczynnika uczenia się ma wpływ zarówno na czas potrzebny(ilość iteracji) by osiągnąć wartość optymalną oraz na poprawność i dokładność wyniku. Jego nieprawidłowa wartość może spowodować to że nie odnajdziemy lub „przeskoczymy” nad wartością pożądaną.

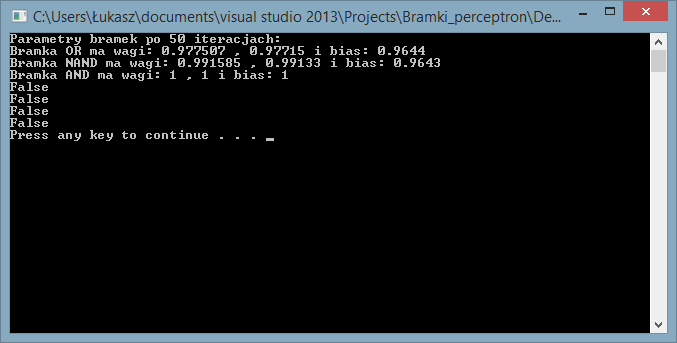
Dla 100 iteracji i n=0,015



Dla 50 iteracji i n=0,025



Dla 50 iteracji i n=0,0001



Poniższe zrzuty pokazują, że wartość współczynnika uczenia się oraz ilość iteracji mają wpływ na poprawność odpowiedzi.

**Wnioski:**

Jak już wcześniej wspomniałem ilość danych uczących oraz wartość współczynnika uczenia się perceptronu ( a tym samym czas czyli ilość iteracji potrzebnych do osiągnięcia poprawnego wyniku, który może się skrócić lub wydłużyć ) mają ogromny wpływ na poprawne działanie perceptronu.