Marcin Damek

Inżynieria obliczeniowa, gr 1

Nr. albumu 285952

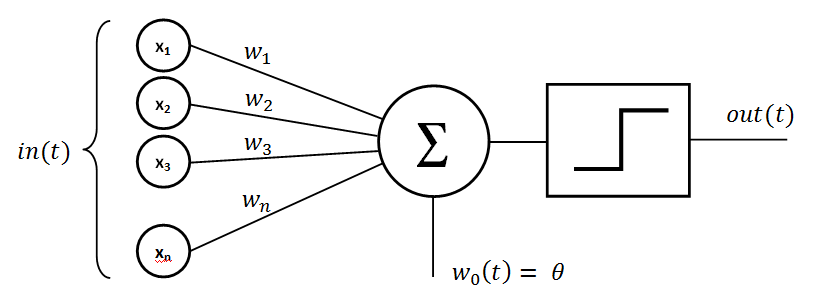
**Sprawozdanie numer 1**

Budowa i działanie perceptronu

Celem​ ​ ćwiczenia​ ​ jest​ ​ poznanie​ ​ budowy​ ​ i​ ​ działania​ ​ perceptronu​ ​ poprzez​ ​ implementację​ ​ oraz uczenie​ ​ perceptronu​ ​ realizującego​ ​ wybraną​ ​ funkcję​ ​ logiczną​ ​ dwóch​ ​ zmiennych.

#### **Perceptron prosty**

#### Najprostszą siecią jednokierunkową jest perceptron prosty. Zbudowany jest jedynie z warstwy wejściowej i warstwy wyjściowej. Ponieważ nie istnieją połączenia pomiędzy elementami warstwy wyjściowej, każdy z tych elementów może być traktowany niezależnie jako osobna sieć o m+1 wejściach i jednym wyjściu. Perceptron jest funkcją, która potrafi określić przynależność parametrów wejściowych do jednej z dwóch klas.



#### 

#### 

#### 

#### Stworzyłem perceptron realizujący funkcję logiczną AND. Poniżej zestawienie oczekiwanych wyników.

|  |  |  |
| --- | --- | --- |
| **p** | **q** | **p ^ q** |
| **1** | **1** | **1** |
| **1** | **0** | **0** |
| **0** | **0** | **0** |
| **0** | **1** | **0** |

#### W celu implementacji perceptronu skorzystałem z informacji z wykładu oraz ze strony: <https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/>

Dane do algorytmu zostały zapisane w zmiennej input, są to pary 1 lub 0, natomiast oczekiwane dane czyli 1 lub 0 w zmiennej outputs. Będę wykorzystywał te dane do uczenia perceptronu. Sterując licznikiem pętli będę decydował na ilu parach liczb algorytm będzie się uczył, żeby przetestować różne wartości. Wagi generowane są losowo. Dane które będę wykorzystywał do testów zapisałem w zmiennej dataToTest,domyślnie zawiera ona 4 pary liczb, ale w celu testów będę je również zwiększał. W zmiennej dataToCompare przechowuję rezultaty które będę porównywał. Zastosowałem miarę błędu nazwaną totalError będąca całkowitą sumą błędów lokalnych. Zakończenie epoki występuje wtedy gdy błąd ten będzie równy 0.

**Zestawienie danych uczących i wyników**

Każdy z wyników został obliczony na podstawie potrójnego uruchomienie programu i obliczenia z nich średniej.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Learning rate | 0.25 | | | | | |
| Ilość danych do uczenia | 0 | 1 | 2 | 3 | 4 | 8 |
| Ilość iteracji | 0 | 1 | 4 | 7 | 4 | 2 |
| Pary do nauki |  | (1,0) | (1,0),(1,1) | (1,0),(1,1),(0,1) | (1,0),(1,1),(0,1),(0,0) | x2 |
| Test | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} |  |
| Wyniki | 4 Błędy | 2 Błąd | 1 Błąd | 0 Błędów | 0 Błędów | 0 Błędów |

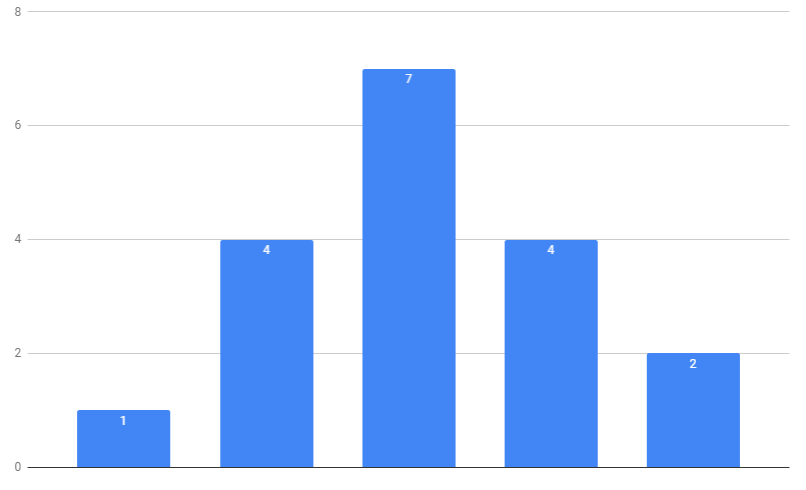
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Learning rate | 0.5 | | | | | |
| Ilość danych do uczenia | 0 | 1 | 2 | 3 | 4 | 8 |
| Ilość iteracji | 0 | 2 | 3 | 8 | 6 | 2 |
| Pary do nauki |  | (1,0) | (1,0),(1,1) | (1,0),(1,1),(0,1) | (1,0),(1,1),(0,1),(0,0) | x2 |
| Test | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} |  |
| Wyniki | 4 Błędy | 1 Błąd | 2 Błąd | 0 Błędów | 0 Błędów | 0 Błędów |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Learning rate | 0.75 | | | | | |
| Ilość danych do uczenia | 0 | 1 | 2 | 3 | 4 | 8 |
| Ilość iteracji | 0 | 1 | 3 | 7 | 6 | 4 |
| Pary do nauki |  | (1,0) | (1,0),(1,1) | (1,0),(1,1),(0,1) | (1,0),(1,1),(0,1),(0,0) | x2 |
| Test | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} |  |
| Wyniki | 4 Błędy | 3 Błędy | 1 Błędy | 0 Błędów | 0 Błędów | 0 Błędów |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Learning rate | 1 | | | | | |
| Ilość danych do uczenia | 0 | 1 | 2 | 3 | 4 | 8 |
| Ilość iteracji | 0 | 1 | 4 | 6 | 7 | 4 |
| Pary do nauki |  | (1,0) | (1,0),(1,1) | (1,0),(1,1),(0,1) | (1,0),(1,1),(0,1),(0,0) | x2 |
| Test | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} | {1, 0}, {0, 1}, {1, 1}, {0, 0} |  |
| Wyniki | 4 Błędy | 2 Błędy | 1 Błąd | 0 Błędów | 0 Błędów | 0 Błędów |

Z zestawienia wyników można odczytać, że przy współczynniku uczenia równym 0.25 i przy pełnym zakresie danych do uczenia wyniki są zadowalające, jednak kiedy dostarczymy więcej danych wyniki były najbardziej zadowalające przy współczynniku uczenia 0.25 oraz 0.5. Wystarczyły 2 epoki, aby nauczyć perceptron w wystarczającym stopniu. Na podstawie zestawienia można zauważyć, że wpływ na prawdziwość wyników ma ilość epok oraz wielkość danych zadanych do uczenia. Można powielać ilość danych do uczenia, jednak dodawanie ich bardzo dużo nie wpłynie już na korzystniejsze wyniki w zadanej sytuacji.

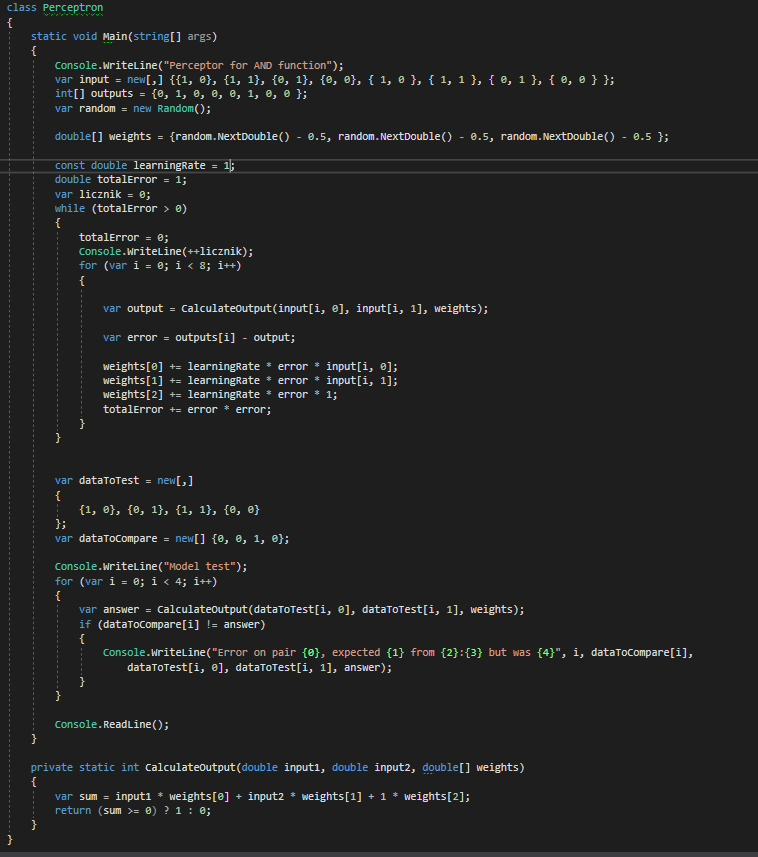
**Wykres ilości danych do uczenia i ilości epok dla learning rate = 0.25**



**Wnioski**

Głównym wyznacznikiem mającym wpływ na działanie perceptronu jest współczynnik nauki. Poprzez manipulację nim w różnych próbach otrzymywałem różne wydajności uczenia algorytmu. Na podstawie wyników oraz przeprowadzonych testów, mogę stwierdzić, że odpowiednie dobranie współczynnika nauki oraz zestawu danych uczących ma fundamentalne znaczenie dla otrzymania prawdziwych wyników i do optymalnego czasu wykonywania programu. Zwiększenie ilości danych w zadanej funkcji logicznej, tak aby były zdublowane w niektórych przypadkach sprawiało, że program działał wydajniej i potrafił szybciej znaleźć oczekiwane wartości. Nie można przesadzać z ilością danych których używamy do uczenia, ponieważ w pewnym momencie już to nie pomaga, tylko zmniejsza wydajność uczenia i może dojść do przetrenowania.

**Kod programu**

****