Piotr Smuga gr2

Podstawy sztucznej inteligencji

Sprawozdanie nr 2

Cel: Poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

Sieci jednokierunkowe których typowym przykładem jest perceptron jednowarstwowy (rys. perceptronu) składają się z neuronów ułożonych w warstwach o jednym kierunku przepływu sygnałów i połączeniach między warstwowych jedynie między kolejnymi warstwami. Sieć tego typu posiada warstwę wejściową, wyjściową i warstwy ukryte. Z funkcjonalnego punktu widzenia układ taki można traktować jako układ aproksymacji funkcji nieliniowej wielu zmiennych y = f(u).

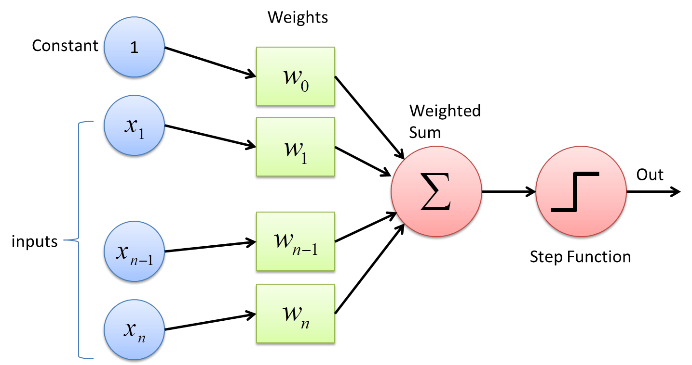


Figure 1 Perceptron

W moim projekcie użyłem sieci jednokierunkowej jednowarstwowej, zgodnie z założeniami.

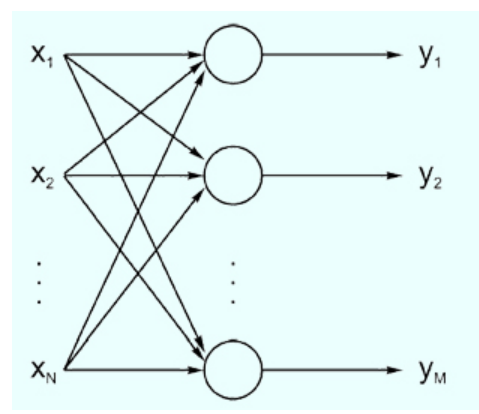


Figure 2 Przykład ilustrujący jednowarstwową jednokierunkową sieć neuronową

Sztuczny neuron to prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jedynym wyjściu. To podstawowy element sieci neuronowych, która jest jedną z metod sztucznej inteligencji. Perceptron to prosty element obliczeniowy. Sumuje ważone sygnały wejściowe oraz porównuje ją z progiem aktywacji. Zależnie od wyniku perceptron może być wzbudzony albo nie. Do uczenia perceptronu można użyć algorytmu uczenia perceptronu, czyli automatycznego doboru wag na podstawie kolejnych przykładów.

Do uczenia perceptronu użyłem algorytmu RPROP (od ang. Resilient backPROPagation). Jest to algorytm przeznaczony dla pełnego (wsadowego) trybu korekcji parametrów (wag). Oznacza to, że jedno skorygowanie parametrów (wag) następuje dopiero po przeglądnięciu przez sieć całego zbioru uczącego i obliczeniu sumarycznego a tym samym dokładnego gradientu. Kluczowymi elementami algorytmu RPROP są: wykorzystywanie jedynie samego znaku każdej składowej gradientu (natomiast wartości są pomijane), a także modyfikowanie współczynnika (współczynników) uczenia w każdym kroku. Współczynnik uczenia jest zwiększany, gdy znaki kolejnych gradientów pozostają zgodne, natomiast zmniejszany (a dokładnie połowiony), gdy są różne.

Ogólny schemat procesu trenowania sieci wygląda następująco:

* Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
* Inicjujemy wagi losowo (na małe wartości).
* Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
* Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią y oraz poprawną odpowiedzią t.
* Błędy propagowane są do wcześniejszych warstw.
* Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkosci przetwarzanych w tym kroku sygnałów.
* Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
* Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

- W programie użyłem funkcję sigmoidalną jako funkcję aktywacji

- każda litera reprezentowana jest jako tablica zawierająca 5x5 liczb reprezentującą piksele

- dane uczące składają się z tablicy 20 liczb reprezentujących 10 pierwszych liter alfabetu polskiego wielkich oraz takich samych 10 liter małych:

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

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

new double[25] {1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0};

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1};

new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1};

new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0};

new double[25] {1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0};

new double[25] {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

new double[25] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0};

new double[25] {1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0};

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

new double[25] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

new double[25] {0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0};

new double[25] {0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

new double[25] {0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0};

new double[25] {0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0};

new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0};

new double[25] {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0};

new double[25] {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0};

new double[25] {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0};

- dane testujące składają się z tablicy 25 liczb reprezentujących daną literkę: new double[25] {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0};

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

Wyniki:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0,1 | 0,2 | 0,3 | 0,5 | 1 |
| A | 0,000436 | 0,000277 | 0,000212 | 0,000151 | 0,000097 |
| B | 0,001312 | 0,000879 | 0,000695 | 0,000521 | 0,000358 |
| C | 0,000103 | 0,000058 | 0,000041 | 0,000027 | 0,000015 |
| D | 0,001511 | 0,001010 | 0,000805 | 0,000595 | 0,000397 |
| E | 0,002281 | 0,001562 | 0,001239 | 0,000935 | 0,000634 |
| F | 0,003325 | 0,002416 | 0,002026 | 0,001613 | 0,001152 |
| G | 0,002473 | 0,001712 | 0,001387 | 0,001062 | 0,000739 |
| H | 0,006873 | 0,005215 | 0,004485 | 0,003661 | 0,002823 |
| I | 0,000214 | 0,000128 | 0,000095 | 0,000065 | 0,000039 |
| J | 0,006545 | 0,004587 | 0,003752 | 0,002892 | 0,002015 |
| a | 0,014720 | 0,007621 | 0,006469 | 0,005202 | 0,003869 |
| b | 0,014720 | 0,011091 | 0,009494 | 0,007720 | 0,005870 |
| c | 0,000488 | 0,000294 | 0,000220 | 0,000149 | 0,000089 |
| d | 0,006478 | 0,004567 | 0,003749 | 0,002887 | 0,002069 |
| e | 0,000961 | 0,000622 | 0,000479 | 0,000353 | 0,000226 |
| f | 0,000132 | 0,000075 | 0,000055 | 0,000037 | 0,000021 |
| g | 0,004999 | 0,003472 | 0,002818 | 0,002139 | 0,001499 |
| h | 0,001736 | 0,001239 | 0,001023 | 0,000794 | 0,000578 |
| i | 0,903991 | 0,918539 | 0,925960 | 0,934350 | 0,943719 |
| j | 0,003332 | 0,027187 | 0,023933 | 0,020192 | 0,016 |

Wnioski:

* Im większy współczynnik uczenia tym dokładniejszy wynik
* wraz ze zwiększaniem wartości współczynnika uczenia zmniejsza się liczba iteracji, co ma wpływ na działanie programu i powoduje uzyskanie błędnych wyników
* dobór wag ma bezpośredni wpływ na działanie perceptronów w sieci, w zależności od ich wartości zmienia się poprawność wyników i liczba iteracji
* dane uczące wpływają na poprawność uczenia perceptronu, przy zbyt małej ilości otrzymujemy błędne wyniki
* bardzo ważny jest odpowiedni dobór współczynnika uczenia, wag oraz liczby danych uczących, gdyż mają bezpośredni wpływ na działanie sieci.

Listing kodu:

* Cały listing kodu został umieszony w repozytorium Git pod następującym adresem: https://github.com/psmuga/PSI/blob/master/Scenariusz%202/ConsoleApp1/ConsoleApp1/Program.cs