Najprostsze sieci neuronowe
Perceptony

Marcin Bocheński
Piotr Weszka
Marek Gruchała

24 Styczeń 2017

Spis treści

1 Wstęp teoretyczny - opis zagadnienia. 3
1.1 Percepton 3
1.2 Sieci neuronowe 3

2 Kod programu 5
2.1 Wyniki działania programu 6

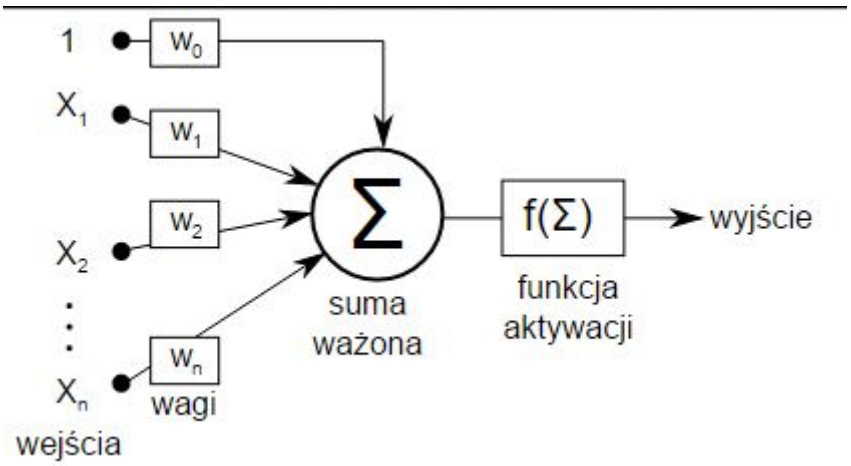
3 Dyskusja wyników - wnioski 7

1 Wstęp teoretyczny - opis zagadnienia.

1.1 Percepton

Percepton jest swoistego rodzaju algorytmem, wykorzystywanym w "machine learning". Algorytm ma charakter liniowego, do przewidywania wyniku, korzystając z operacji liniowych na wektorach z zadanymi wagami. Powstanie algorytmu datuje się na lata 50, a jego pierwszą implementacja miała miejsce w pierwszej sieci neuronowej.

Współcześnie można powiedzieć, że percepton jest alogrytmem implementującym nadzorowaną naukę klasyfikatorów binarnych. Do realizacji tego celu wykorzystywane są tzw. Neurony McCullocha-Pittsa. Poniżej prezentujemy ideowy schemat wspomnianego neuronu:



Rysunek 1: Schemat neuronu McCllocha-Pittsa

Jak widać na załączonym schemacie neuron McCllocha-Pittsa posiada wiele wejść i jedno wyjście. Każde z wejść posiada przypisaną liczbę rzczywistą - wagę wejścia. Wartość na wyjściu liczona jest w następujący sposób:

- Obliczana jest suma iloczynów wartośći x_i podanych na wejścia i wag w_i wejść według poniższego wzoru:
$$s = w_0 + \sum_{i=1}^{n} x_i w_i$$
- na wyjście podawana jest wartość funkcji aktywacji f(s) dla obliczonej sumy.

obliczana jest suma iloczynów wartości xi podanych na wejścia i wag wi wejść:

1.2 Sieci neuronowe

Sieci neuronowe to komputerowe podejście na symulacji biologicznej aktywności procesów myślowych, oparte o współdziałanie wielu elementarnych "sztucznych neuronów", modelujących zachowanie mózgu ludzkiego. Każdy sztuczny neuron połączony jest z wieloma innymi. Każde połączenie międzyneuronowe może być obarczone pewną wartością funkcji, powyżej której operacje logiczne będą realizowane, w celu usprawnienia procesu. Warto wspomnieć, że systemy sieci neuronowych są raczej systemami samouczącymi się aniżeli zaprogramowanymi explicite.

Na dzień dzisiejszy wyszczególnić można następujące typy sieci neuronowych:

- sieci jednokierunkowe - *to sieci neuronowe, w których nie występuje sprzężenie zwrotne, czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu. Najprostszą siecią neuronową jest pojedynczy perceptron progowy, opracowany przez McCullocha i Pittsa w roku 1943. W bardziej zaawansowanych rozwiązaniach stosuje się funkcje przejścia. Najpopularniejszą klasę funkcji stosowanych w sieciach neuronowych stanowią funkcje sigmoidalne, np. tangens hiperboliczny. Sieć zbudowana z neuronów wyposażonych w nieliniową funkcję przejścia ma zdolność nieliniowej separacji wzorców wejściowych. Jest więc uniwersalnym klasyfikatorem. Sieci jednokierunkowe dzielą się na jednowarstwowe, dwuwarstwowe i wielowarstwowe. Sieci jednowarstwowe mogą rozwiązać jedynie wąską klasę problemów. Sieci dwu i wielowarstwowe mogą rozwiązać znacznie szerszą klasę i są pod tym względem równoważne, jednak stosuje się do nich inne algorytmy uczenia (dla wielowarstwowych są one prostsze).*

- sieci rekurencyjne - *Mianem sieci rekurencyjnej określa się sieć, w której połączenia między neuronami stanowią graf z cyklami. Wyróżnia się dwa typy takich sieci: sieć Hopfielda (układ gęsto połączonych ze sobą neuronów (każdy z każdym, ale bez połączeń zwrotnych) realizującą dynamikę gwarantującą zbieżność do preferowanych wzorców) oraz maszyny Boltzmanna – koncepcja opracowana przez Geoffa Hintona i Terry'ego Sejnowskiego; stochastyczna modyfikacja sieci Hopfielda; modyfikacja ta pozwoliła na uczenie neuronów ukrytych i likwidację wzorców pasożytniczych kosztem zwiększenia czasu symulacji*

- samoorganizujące się may - *to sieci neuronów, z którymi są stowarzyszone współrzędne na prostej, płaszczyźnie lub w dowolnej n-wymiarowej przestrzeni. Uczenie tego rodzaju sieci polega na zmianach współrzędnych neuronów, tak, by dążyły one do wzorca zgodnego ze strukturą analizowanych danych. Sieci zatem "rozpinają się" wokół zbiorów danych, dopasowując do nich swoją strukturę. Sieci te stosowane są do klasyfikacji wzorców, np. głosek mowy ciągłej, tekstu, muzyki. Do najciekawszych zastosowań należy rozpinanie siatki wokół komputerowego modelu skanowanego obiektu.*

2 Kod programu

```
'''
Program napisany w pythonie probojacy nauczyc sie
mnozenia trzech liczb ulamkowych wykorzytujac szczegolny
przypadek sieci neuronowej - perceptron

autorzy: M. Bochenski, M. Gruchala, P. Weszka
'''

import numpy as np


# funkcja aktywacji (funkcja sigmoidalna)
def sigmoid(x,pochodna=False):
    if(pochodna==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))



# dane wejsciowe, macierz 9x3
X = np.array([  [0.5,0.2,0.2],
                [0.3,0.3,0.3],
                [0.1,0.4,0.9],
                [0.4,0.3,0.2],
                [0.3,0.6,0.4],
                [0.5,0.5,0.7],
                [0.9,0.8,0.7],
                [0.3,0.4,0.5],
                [0.6,0.7,0.5] ])

# dane wyjsciowe, wektor 9x1
y = np.array([[0.02,0.027,0.036,0.024,0.72,0.172,0.504,0.06,0.210]]).T

# inicjalizacja wag, wektor 3x1
syn0 = 2*np.random.random((3,1)) - 1

print 'Wylosowane wagi:\n'

print syn0

#xrange w python2 zwraca iterator dlatego lepszy niz range
for iter in xrange(1000000):

    l0 = X
    #wykonanie funkcji aktywacji na sumie iloczynow wag i danych wejsciowych
    l1 = sigmoid(np.dot(l0,syn0))

    # obliczanie bledu
```

```
    l1_blad = y - l1

    #obliczanie rozznicy
    l1_delta = l1_blad * sigmoid(l1,True)

    # aktualizacja wag
    syn0 += np.dot(l0.T,l1_delta)

print "\n\nDane wyjsciowe po trenowaniu:"

print l1

print "\n\nRoznica miedzy wartoscia obliczona i wyuczona: \n"

print y-l1
```

2.1 Wyniki działania programu

Poniżej prezentowane są wyniki działania powyższego programu: Wylosowane wagi:
-0,89180645
0,47918085
-0,59315833

Dane wyjsciowe po trenowaniu:
0.00987644
0.11639736
0.01642483
0.10126659
0.74284032
0.00528932
0.00382807
0.0726666
0.14359096

Roznica miedzy wartoscia obliczona i wyuczona:
0.01012356
-0.08939736
0.01957517
-0.07726659
-0.02284032
0.16671068
0.50017193
-0.0126666
0.06640904

3 Dyskusja wyników - wnioski

Wyniki, które otrzymujemy jako konsekwencja działania programu różnią się od wyników analitycznych. Niedokładność wyników jest konsekwencją małego "zbioru treningowego". Na wynik działania programu można wpływać poprzez sterowanie parametrami funkcji aktywacji, co może się przełożyć na lepsze wyniki. Bez wątpienia funkcja aktywacji nie jest jedyną bolączką, z jaką potyka się nasz program w ujęciu dokładności generowanych wyników. Poza większym zbiorem treningowym oraz doborem inneh funkcji aktywacji możnaby zastosować większą liczbę warstw. Podejście takie jednak jest wysoce obciążające zasoby naszych komputerów i w konsekwencji zdecydowaliśmy się nie zmieniać ilości warstw. Gdybyśmy mogli spróbować zrównoleglić obliczenia i wykorzystać zasoby jakiegoś klastra obliczeniowego, sprawa miałaby się zupełnie inaczej.