# Dominik Kędzior

## Inżynieria Obliczeniowa

Nr 293094

# Podstawy Sztucznej Inteligencji

## Sprawozdanie nr 3. Budowa i działanie sieci wielowarstwowej typu feedforward

# Część teoretyczna

Celem ćwiczenia było zapoznanie się z budową oraz działaniem wielowarstwowych sieci neuronowych poprzez uczenie kształtu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

Sieć wielowarstwowa jest siecią połączonych ze sobą neuronów, w której neurony przesyłają sygnały wszystkim neuronom kolejnej warstwy (na zasadzie „każdy z każdym”). Pozwala ona na tworzenie sieci niemal o dowolnej charakterystyce. Działanie takiej sieci polega na liczeniu odpowiedzi neuronów w kolejnych warstwach – najpierw w pierwszej, do której trafiają

sygnały z wejść sieci, potem korzystając z wyników poprzedniej warstwy, liczymy odpowiedzi kolejnych warstw neuronów. Odpowiedzi znajdujące się w ostatniej warstwie są wynikami sieci.



Warstwy (minimum jedna), które znajdują się się między warstwą wejściową a warstwą wyjściową nazywamy warstwami ukrytymi.

W sieciach wielowarstwowych nie stosuje się nieliniowych funkcji aktywacji (np.: funkcje sigmoidalne, czy radialne). Często jednak można znaleźć funkcję liniową w warstwie wyjściowej. Najczęściej sieci wielowarstwowe korzystają z uczenia nadzorowanego. Jedną z zalet sieci neutronowej jest fakt, że nie musimy wyszukiwać osobno dobierać wag. Można tak wytrenować wagi tak, by znaleźć optymalny zestaw za pomocą algorytmu wstecznej propagacji błędu. Działanie tego algorytmu oparte jest na regule delta. Algorytm uczenia sieci wielowarstwowych metodą propagacji błędu:

1. Podajemy dane wejściowe.

2. Inicjujemy losowo, odpowiednio małe wagi.

3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po Warstwie).

4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią y oraz poprawną odpowiedzią t. Błędy propagowane są do wcześniejszych warstw.

5. Każdy neuron modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.

6. Dla kolejnych wektorów uczących powtarzamy operacje od kroku 3. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.

7. Jeśli średni błąd na danych treningowych przestanie maleć, przerywamy działanie algorytmu.

Funkcja obliczająca błąd:



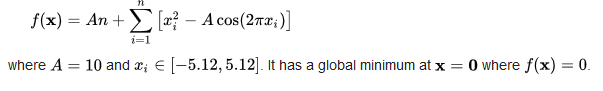
# Część praktyczna

Do wykonania tego zadania wykożystałem bibliotekę neurolab dołączoną do języka Python. Pozwala ono nie tylko na utworzenie prostych sieci jednowarstwowych, ale również tych bardziej złożonych, korzystających z modelu sieci wielowarstwowych. Można również, za pomocą parametrów wykorzystać algorytm propagacji błędów.

Zadanie polegało na wygenerowaniudanych uczących i testujących dla funkcji

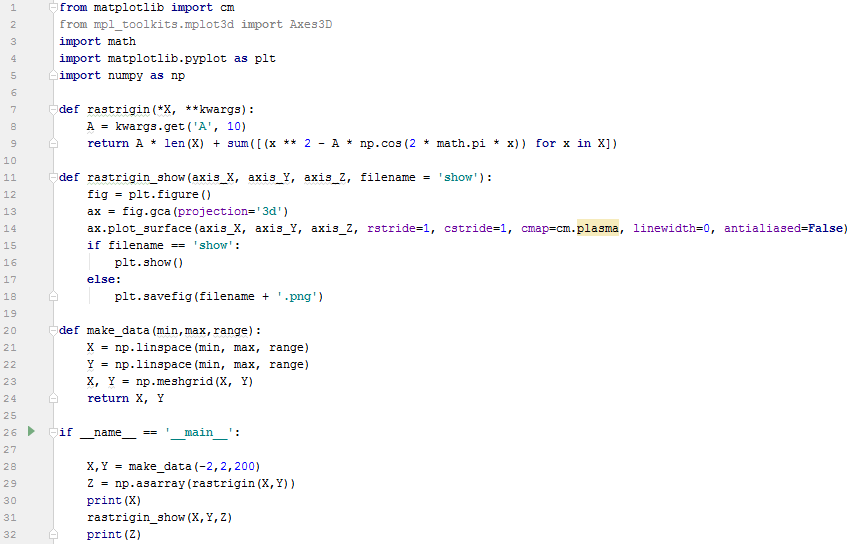
Rastrign 3D dla danych wejściowych z przedziału [-2; 2].

Funkcja Rastrign ma następującą postać:

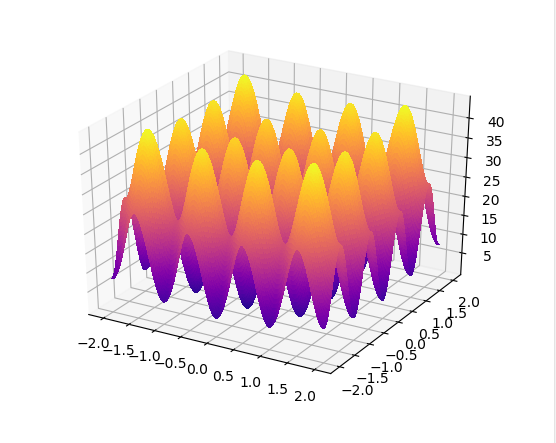


Funkcję tą zaimplementowałem w taki sposób aby było możliwe jej wykorzystanie w

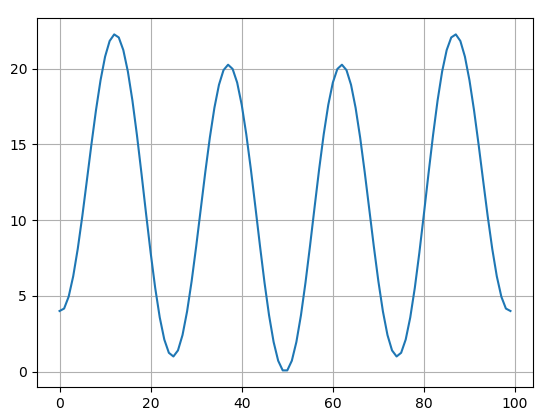
n-wymiarowej przestrzeni



Pzrykładowy wynik dla danych 3D:



Wykres funkcji wykożystanej jako dane uczące wynikowe:



Dane wejściowe stanowi tablica wypełniona liczbami zmieniającymi się co pewien krok z przedziału [-2;2] i ilością elementów oznaczaną w programie jako xrange, do stworzenia tej tablicy wykożystałem funkcję linspace biblioteki numpy. Tablica wynikowa to wynik działania funkcji Rastrign3D dla danych wejściowych.

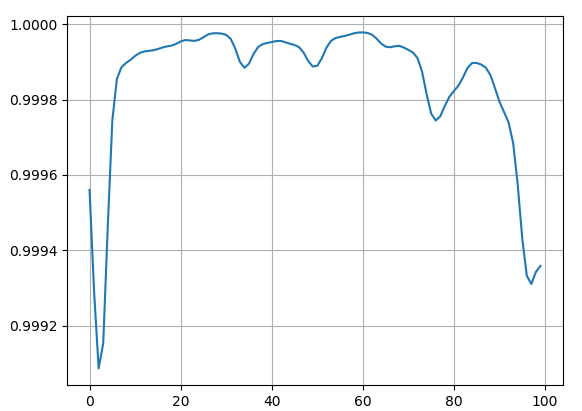
Do stworzenia sieci wykożystałem funkcję newff() przyjmującą jako argumenty dane startowe zawierające minimalna i maksymalna wartość oraz schemat budowy naszej sieci (ilość warstw i perceptonów w warstwie). Funkcja ta tworzy sieć

wielowarstwową zawierającą n warstw ukrytych. Użyliśmy dwóch, ponieważ przeważnie buduje się sieci o maksymalnie 2 warstwach ukrytych, gdyż istnieje ryzyko przeuczenia sieci. Nasz problem nie jest na tyle złożony, by korzystać z dużej ilości warstw ukrytych.

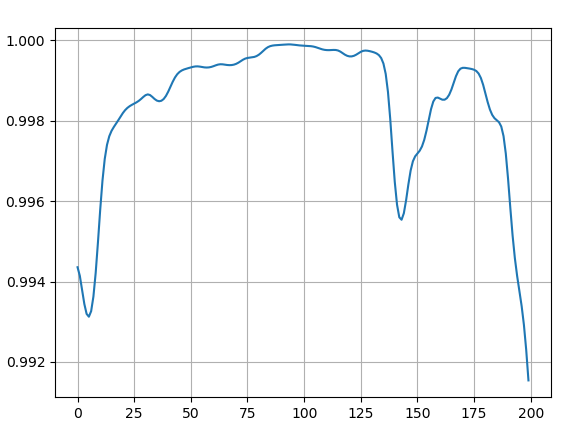
Przed treningiem wybrałem sposób trenowania na *train\_gdm* czyli algorytm wstecznej propagacji z możliwością manipulowana współczynnikiem bezwładności.

Następnie trenowałem sieć z różnymi wartościami współczynników.

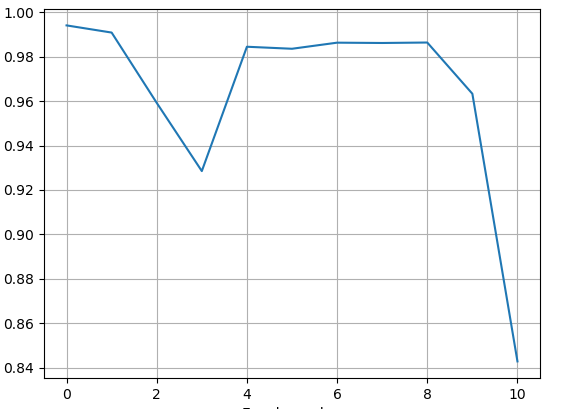
Przykładowe wywołania testowe:



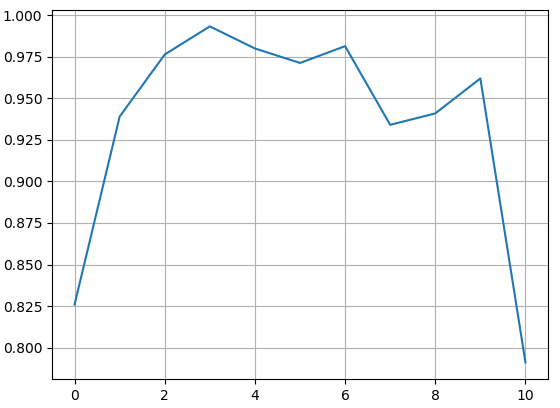
Lr = 0.0001 epochs = 2000 mc =0.1



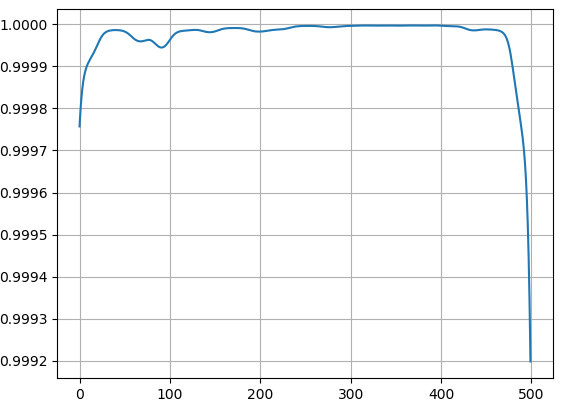
Lr =0.001 epochs=1000 mc = 0.5



Zmniejszenie ilości danych uczącyc



Lr =0.01 epochs = 10000 mc = 0.5



Sieć przetrenowana

# Wnioski:

• Sieci wielowarstwowe nie są pozbawione błędów, podobnie w stosunku do swoich

prostszych, jednowarstwowych odpowiedników.

• Podobnie jak sieciami jednowarstwowymi, możemy sterować dopasowując poszczególne

parametry uczenia, takie jak współczynnik uczenia, czy bezwładność (momentum).

Współczynniki te mają niebagatelny wpływ na wyniki działania sieci.

• Sieć wielowarstwowa bez algorytmu propagacji błędu działają lepiej niż sieci posiadające

ten algorytm mimo, że błędy, które na wejściu i w trakcie działania wydają się być

pomijalne mogą na końcu spowodować duży rozrzut między tym, co zostało zasymulowane

a tym, co powinna dać sieć w swoim wyniku.

• Jeżeli nie korzystamy z algorytmu propagacji danych należy pamiętać, by były duże różnice

między wsp. uczenia a bezwładnością. Najgorsze wyniki osiągają te sieci, których wsp.

uczenia jest równy bezwładności. Tylko wówczas nasze wyniki osiągają akceptowalne

wartości.

• Sieci wielowarstwowe posiadające algorytm propagacji danych wykazują praktycznie duże

odsetki błędów. Nie należy jednak nie zwracać uwagi na wymienione wcześniej parametry.

W tym przypadku im większy jest wsp. uczenia od bezwładności, tym częściej mogą

zdarzać się błędy.

• Algorytmy z propagacją danych przyspieszają proces uczenia nawet o połowę epok w

stosunku do sieci niezawierających tego algorytmu.

• Gradienty uczenia przy sieciach z algorytmem propagacji błędów posiadają charakterystykę niemalże liniową, dzięki czemu można łatwiej przewidzieć, jakich współczynników do nauki sieci najlepiej użyć, by otrzymane rezultaty były zadowalające.

• Najtrudniejszym zadaniem dla sieci było przyporządkowanie argumentowi 0 wartości 0.

Niestety żadna z sieci nie podołała temu zadaniu – sieci tylko modelują pewne dane na bazie pewnych danych uczący, a nie tworzą idealnych obrazów między danymi wejściowymi a danymi wyjściowymi.

• Złe dobranie współczynników może całkowicie zniszczyć sieć

# Listing właściwego programu

