Marcin Damek
Inżynieria obliczeniowa, gr 1
Nr. albumu 285952

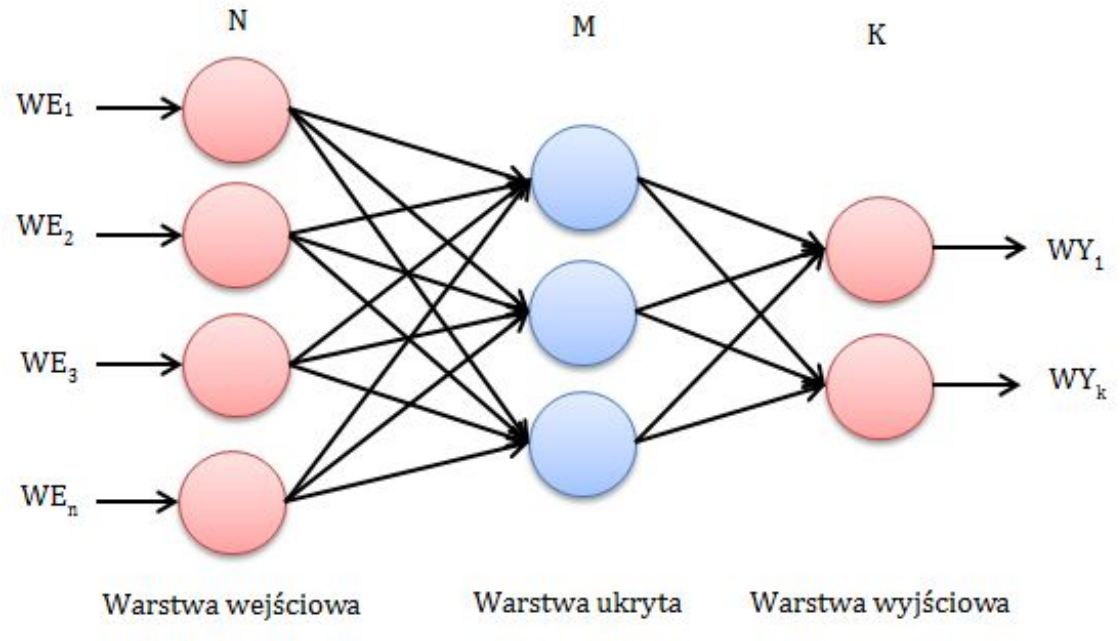
Sprawozdanie numer 3

Budowa i działanie sieci wielowarstwowej typu feedforward

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

Część teoretyczna

Sieci wielowarstwowe to odpowiednio połączone warstwy neuronów zwykle o nieliniowych funkcjach aktywacji (np. neurony sigmoidalne, radialne), aczkolwiek czasami w warstwie wyjściowej pojawiają się neurony liniowe. Sieci wielowarstwowe muszą posiadać minimalnie dwie warstwy neuronów: warstwę wejściową i warstwę wyjściową pomiędzy którymi może być jedna lub więcej warstw ukrytych. W każdej warstwie może występować różna ilość neuronów: W warstwie wejściowej odpowiada ona zwykle ilości parametrów opisujących dane wejściowe, w warstwie wyjściowej np. ilości klas, natomiast ilość neuronów w warstwach ukrytych odpowiada za możliwości modelu. Neurony łączą się tylko pomiędzy (zwykle) sąsiednimi warstwami, zaś wewnątrz warstw nie występują połączenia pomiędzy neuronami. Neurony zwykle łączymy pomiędzy sąsiednimi warstwami na zasadzie każdy z każdym.



Sposób nauki

Sieci wielowarstwowe uczymy zwykle metodami nadzorowanymi (tzn. z nauczycielem):
1. Podajemy na wejście sygnał wejściowy (zwykle w postaci wektora lub macierzy danych)
2. Obliczamy wartości wyjściowe neuronów w 1. warstwie i poprzez ich połączenia z neuronami w 2. warstwie podajemy te wartości jako sygnały wejściowe dla neuronów w 2. warstwie.
3. Obliczamy wartości wyjściowe w kolejnych warstwach tym samym sposobem.
4. Wartości wyznaczone w ostatniej warstwie stanowią zarazem odpowiedź sieci na podany sygnał wejściowy.
5. Sygnał ten porównujemy z wzorcowym (określonym przez nauczyciela) i wyznaczamy błąd.
6. Korzystając metod gradientowych propagujemy błąd wstecz przez sieć i dokonujemy korekty wartości wag połączeń synaptycznych.

Metoda propagacji błędu

Algorytm wstecznej propagacji błędu zdecydowanie dominuje wśród metod uczenia jednokierunkowych sieci wielowarstwowych. Nazwa metody oddaje zasadę jej działania, która polega na „przenoszeniu" błędu, jaki popełniła sieć, w kierunku od warstwy wyjściowej do warstwy wejściowej (a więc wstecz w stosunku do kierunku przepływu informacji).

Cykl uczenia metodą wstecznej propagacji błędu (backpropagation) składa się z następujących etapów:
1. Wygeneruj losowo wektory wag.
2. Podaj wybrany wzorzec na wejście sieci.
3. Wyznacz odpowiedzi wszystkich neuronów wyjściowych sieci
4. Oblicz błędy wszystkich neuronów warstwy wyjściowej
5. Oblicz błędy w warstwach ukrytych
6. Zmodyfikuj wagi
7. Jeśli wartość funkcji celu jest zbyt duża wróć do punktu 2.

Algorytm wstecznej propagacji błędu (backpropagation) określa procedurę korekty wag w sieci wielowarstwowej przy wykorzystaniu gradientowych metod optymalizacji. Korekta wektora wag sieci oparta jest na minimalizacji funkcji miary błędu (funkcji celu), którą określono jako sumę kwadratów błędów na wyjściach sieci. Błąd obliczamy według wzoru:

$$d(w_1,\ldots,w_K)=\frac{1}{2}\left(f(w_1z_1+\ldots+w_Kz_K)-t\right)^2$$

Część praktyczna

Do wykonania zadania użyłem pakietu MATLAB wraz z narzędziem Neural Network Training Tool. Dzięki niemu w stosunkowo łatwy sposób możemy tworzyć i uczyć sieć oraz w intuicyjny sposób analizować wyniki.
Moim zadaniem było w pierwszej kolejności wygenerowanie danych testowych z przedziału od -2 do 2.
Funkcja Rastrigin ma następującą postać:

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$$

where $A = 10$ and $x_i \in [-5.12, 5.12]$. It has a global minimum at $\mathbf{x} = \mathbf{0}$ where $f(\mathbf{x}) = 0$.

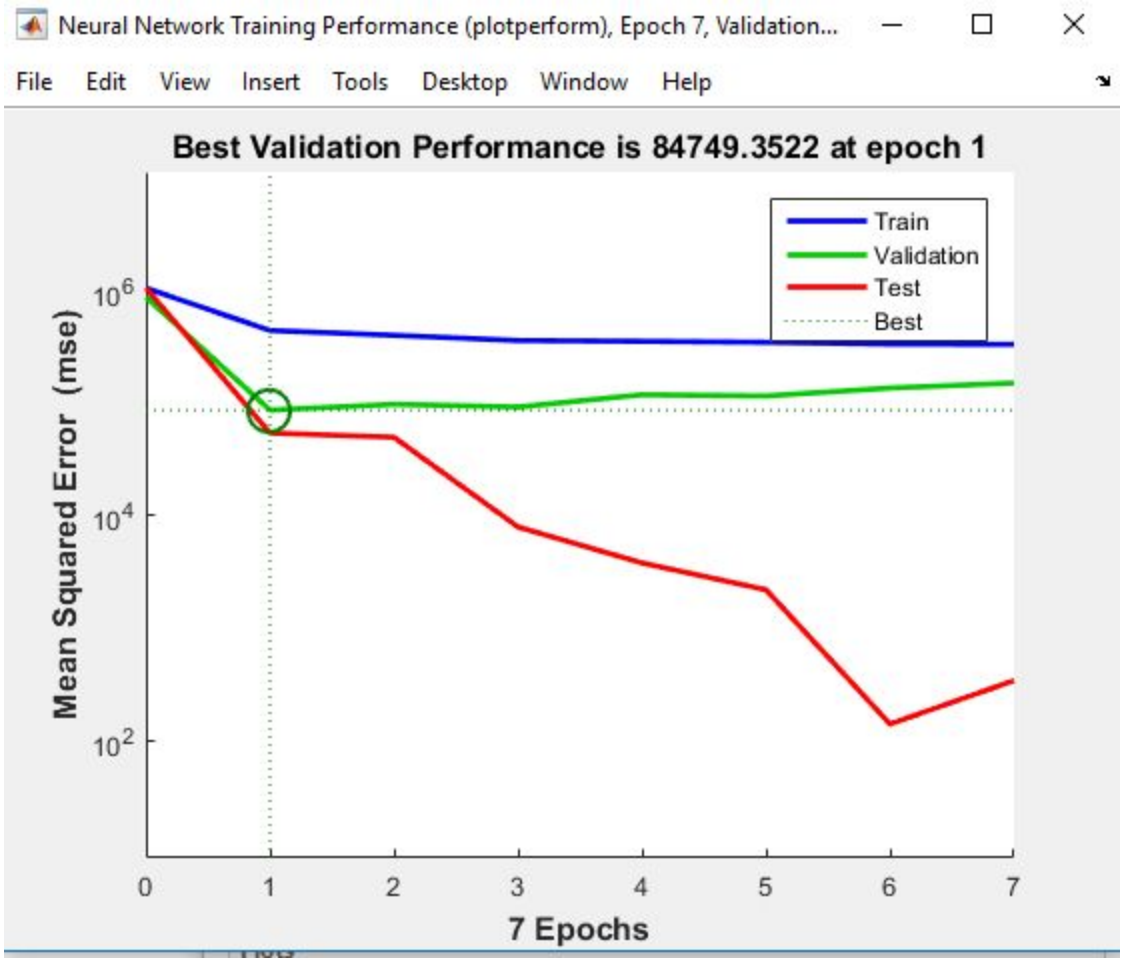
Funkcja ta w programie nazywa się Rastrigin. Napisałem ją w oparciu o implementację znalezioną w internecie. Dane testowe przyjąłem z przedziału -2 do 2 z krokiem co 0.5 aby otrzymać optymalną ilość wyników. Dodałem je do 9 elementowej tablicy dataIn, która posłuży za dane wejściowe. Danych oczekiwanych również jest 9 i znajdują się w tablicy o nazwie dataOut.
Dzięki poleceniu
network = feedforwardnet(2);
Utworzyłem sieć z dwoma warstwami ukrytymi. Uznałem, że dwie warstwy powinny wystarczyć, ponieważ problem nie jest bardzo skomplikowany, a przy okazji chciałem uniknąć przeuczenia się sieci. Wykorzystanie algorytmu wstecznej propagacji błędu zagwarantowało mi napisanie:
network.trainFcn = 'traingd';
Działanie sieci było uzależnione od tego w jaki sposób manipulowałem współczynnikiem uczenia oraz bezwładności. Wszystkie wyniki zebrałem i zestawiłem poniżej.

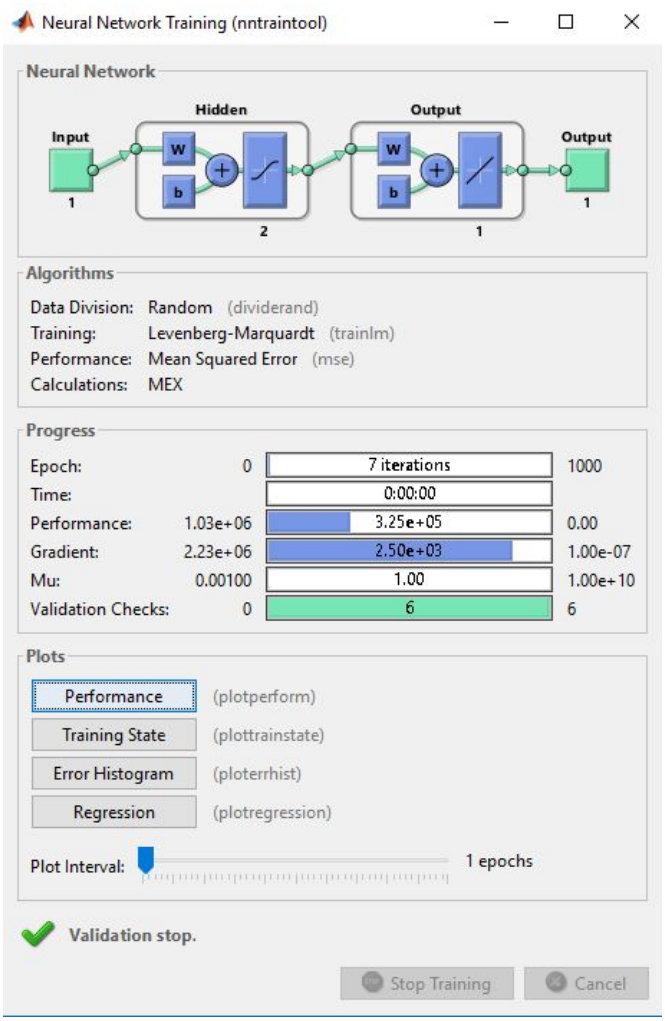
Wyniki

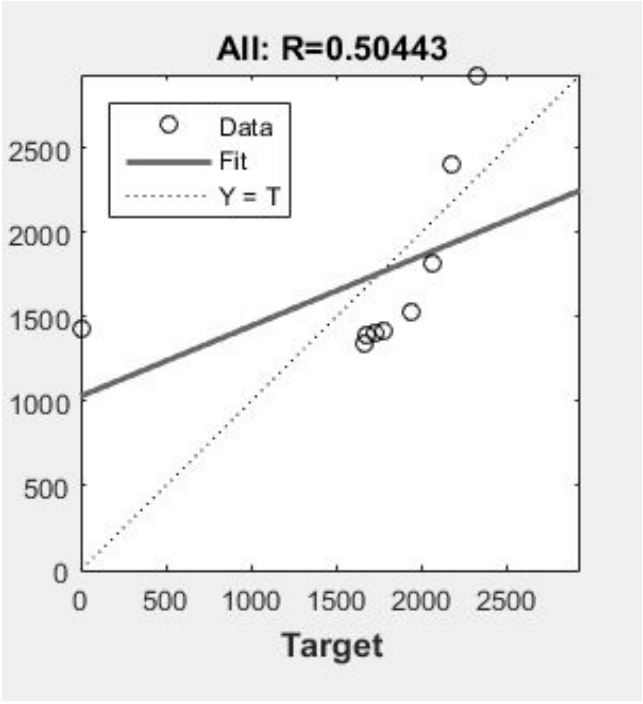
Współczynnik uczenia	Bezwładność	-2	-1.5	-1	-0.5	0	0.5	1	1.5	2	Epoki
0.5	0	1,252.70	1,770.20	2,011.00	2,085.20	2,109.40	2,135.00	2,216.00	2,480.10	3,047.50	10
	0.5	928.77	859.81	827.85	818.60	817.98	824.97	854.06	950.59	1,158.30	6
	1	1,070.00	979.86	937.92	925.18	921.70	920.19	917.37	908.69	890.11	10
0.1	0	2,322.50	2,392.90	2,425.40	2,434.20	2,431.90	2,413.10	2,338.90	2,093.20	1,564.70	10
	0.5	561.35	-33.84	-310.42	-393.63	-413.04	-408.75	-372.71	-249.58	15.91	6
	1	1,337.40	799.46	549.18	472.55	449.23	429.93	373.63	191.45	-199.81	24
0.01	0	2,799.80	2,434.10	2,263.80	2,211.40	2,193.80	2,174.10	2,110.50	1,903.10	1,457.30	11
	0.5	1,191.00	1,607.30	1,800.60	1,857.90	1,867.80	1,850.30	1,769.50	1,499.60	918.59	8
	1	1,629.20	1,571.40	1,544.70	1,537.40	1,538.70	1,551.70	1,603.60	1,775.30	2,144.60	7
Bez wstecznej propagacji przy ustawieniu 0.5/0.5		1,424.20	1,480.40	1,830.30	1,796.20	636.39	1,951.70	2,070.50	2,318.00	2,279.20	
Oczekiwane wyniki		1,663.30	1,680.00	1,732.90	1,782.00	0.00	1,948.40	2,065.50	2,178.20	2,326.20	

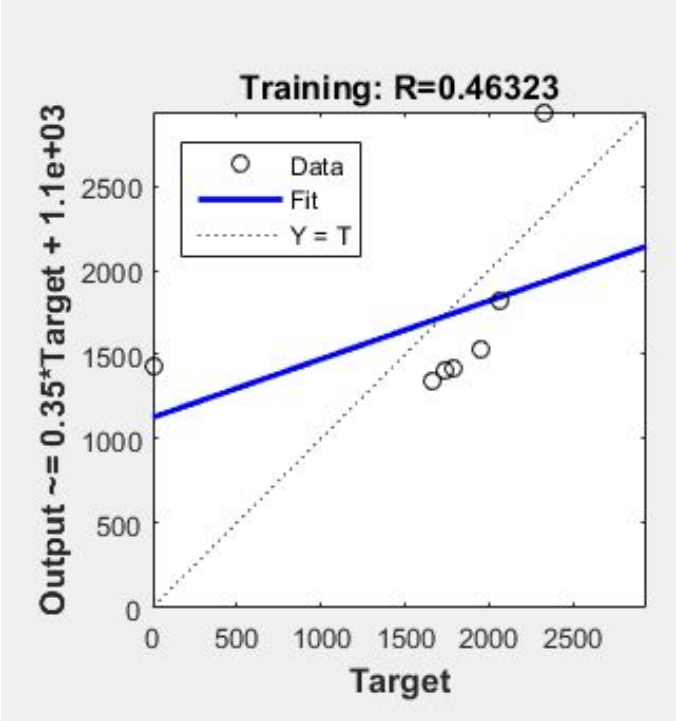
W tabeli umieściłem zestawienie wyników jakie otrzymałem przy kolejnych kombinacjach bezwładności oraz współczynnika uczenia. Na zielono zaznaczyłem wyniki których błąd był mniejszy niż +/- 300 od wartości otrzymanej.

Poniżej prezentuję wykresy dla najlepiej dobranej pary współczynnika uczenia oraz bezwładności.




Neural Network Training (nntraintool)
Neural Network
Input
Hidden
Output
Output
W
b
1
2
1
1
Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX
Progress
Epoch: 0 7 iterations 1000
Time: 0:00:00
Performance: 1.03e+06 3.25e+05 0.00
Gradient: 2.23e+06 2.50e+03 1.00e-07
Mu: 0.00100 1.00 1.00e+10
Validation Checks: 0 6 6
Plots
Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Regression (plotregression)
Plot Interval: 1 epochs
Validation stop.
Stop Training
Cancel


All: R=0.50443
Data
Fit
Y = T
0
500
1000
1500
2000
2500
Target


Training: R=0.46323
Data
Fit
Y = T
Output ~= 0.35*Target + 1.1e+03
0
500
1000
1500
2000
2500
Target

Analiza i wnioski

W sieciach wielowarstwowych występują błędy, tak samo jak w testowanych ostatnim razem sieciach jednowarstwowych. Przy różnych kombinacjach współczynnika uczenia oraz bezwładności otrzymywałem sporo błędów. W zestawieniu widać, że najlepsza okazała się kombinacja, w której skład wchodzi współczynnik nauki równy 0.01 oraz bezwładność równa jeden. 6 wyników na 9 wykonywanych było w granicy ustalonej przeze mnie normy. Jest to stosunkowo dobry wynik, zważywszy na to, że przy niektórych kombinacjach nie było ani jednego wyniku który byłby zbliżony do oczekiwanego przeze mnie. W przypadku w którym najwięcej wyników było zbliżonych do oczekiwanych liczba epok wyniosła 7.

-Ważną rzeczą tak samo jak przy sieciach z jedną warstwą jest odpowiedni dobór współczynników. Ma on ogromny wpływ na ostateczny wynik nauczania sieci.

-W ramach testów uruchomiłem sieć kilkukrotnie bez użycia algorytmu wstecznej propagacji błędu, w wyniku czego zaobserwowałem, że potrzebowały one z reguły więcej opok aby sieć została nauczona, z czego wynika, że zastosowanie algorytmu zmniejsza ilość epok potrzebnych do nauczenia modelu

-Największy problem sieć miała jeśli chodzi o wartość 0, która dość mocno odbiegają od pozostałych wartości zadanych w modelu

-Lepsze wyniki otrzymywałem kiedy różnica między współczynnikiem nauki oraz bezwładności była duża, niż kiedy były zbliżone lub równe

-Im większy był współczynnik nauki tym więcej błędów mogłem zaobserwować

Kod programu

```
dataIn = [-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2];
dataOut = [1.6633e+03, 1.6800e+03, 1.7329e+03, 1.7820e+03, 0, 1.9484e+03, 2.0655e+03, 2.1782e+03, 2.3262e+03];

testowe = zeros(1);

network = feedforwardnet(2);
network.trainFcn = 'traingd';
network.trainParam.lr = 0.01;
network.trainParam.mc = 1;
network = train(network, dataIn, dataOut);
wyniki = zeros(size(network));

for i = 1:9
    testowe(i) = Rastrigin(dataIn(i));
    wyniki(i) = sim(network, dataIn(i));

end
```

```
function fx = Rastrigin(x)
    if x == 0
        fx = 0;
    else
        A = 10;
        n = 100;
        x1 = x;
        dx = (5.12-x)/n;
        fx = A * n;
        for i = 1:n
            x = x1 + (i * dx);
            fx = fx + (x^2) - (A * cos(2 * pi * x));
        end
    end
end
```