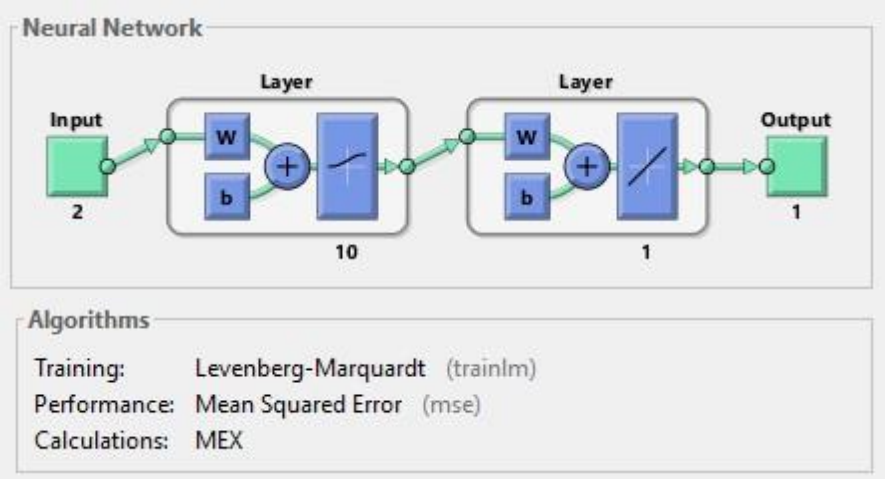
Fabiola Dąbroś , IO gr.1

Sprawozdanie z zagadnienia nr 1.

Temat ćwiczenia:
Budowa i działanie perceptronu.
Cel ćwiczenia:
Celem ćwiczenia jest poznanie budowy i działanie perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

1. Opis budowy perceptronu oraz wykorzystanego algorytmu uczenia.

Perceptron zbudowany jest jedynie z warstwy wejściowej oraz warstwy wyjściowej. Jego działanie polega na klasyfikowaniu danych pojawiających się na wejściu i ustawianiu stosownie do tego wartości wyjścia. Przed używaniem perceptron należy wytrenować podając mu przykładowe dane na wejście i modyfikując w odpowiedni sposób wagi wejść i połączeń między warstwami neuronów, tak aby wartość na wyjściu przybierała pożądane wartości.



Do treningu sieci jednokierunkowej wykorzystałam funkcję trainlm, która aktualizuje wagi i wartości błędu zgodnie z metodą Levenberga-Marquardta. Algorytm ten jest bardzo popularną metodą uczenia sieci. Łączy w sobie cechy metody największego spadku i metody Gaussa-Newtona.

Polega ona wyłącznie na tym, że w tej drugiej metodzie do hesjanu dodawana jest dodatkowo macierz jednostkowa przemnożona przez pewien dodatni współczynnik μ, którego wartość modyfikowana jest w trakcie uczenia.

Wzór opisujący ten algorytm wygląda następująco:

$$\mathbf{w}^{t} = \mathbf{w}^{t-1} - (\mathbf{H} + \mu\mathbf{I})^{-1}\nabla E(\mathbf{w}^{t-1})$$

Jeśli μ zbliża się do zera, to wektor wag uzyskany przy pomocy metody Levenberga-Marquardta będzie bardzo zbliżony do wartości wyznaczonych przy zastosowaniu prostej

formuły Newtona. Zwiększaniu wartości μ towarzyszy wzrost znaczenia kierunku poprawy wyznaczonego w oparciu o gradient funkcji błędu. Metoda Newtona jest szybsza i dokładniejsza niż minimalne minimum, więc celem jest jak najszybsze przejście do metody Newtona. Zatem μ po każdym pomyślnym kroku (obniżenie funkcji wydajności) zostaje zmniejszony i wzrasta tylko wtedy, gdy wstępny krok zwiększy wydajność. W ten sposób funkcja performance zawsze jest redukowana przy każdej iteracji algorytmu.

2. Zestawienie otrzymanych wyników.

Jako funkcję logiczną wybrałam funkcję OR, która polega na tym , że alternatywa jest prawdziwa, jeżeli co najmniej jeden z jej składników jest prawdziwy. W przeciwnym razie alternatywa zdań jest fałszywa.

Dane:

```
P=[1 0 0 1  ; 1 0 1 0];
T=[1 0 1 1];
```

Perceptron przed treningiem wskazał następujące wartości:

```
przed_treningiem =

    0.2064    0.5317    0.4199    0.8768
```

Natomiast po treningu wyniki prezentują się następująco:

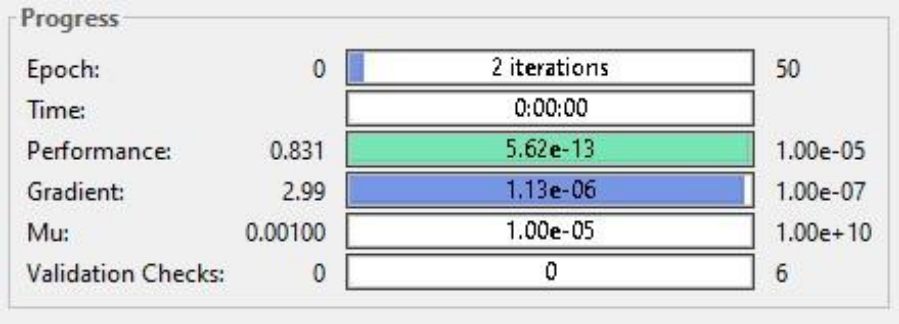
```
po_treningu =

    1.0000    0.0000    1.0000    1.0000
```

Widzimy zatem, że nauka przyniosła oczekiwane rezultaty. Na początku perceptron nie odbył jeszcze szkolenia dlatego wyniki nie ukazują prawidłowego działania funkcji OR. Natomiast po treningu pokrywają się one z wynikami zestawu uczącego.

3. Analiza błędów uczenia i testowania opracowanego perceptronu

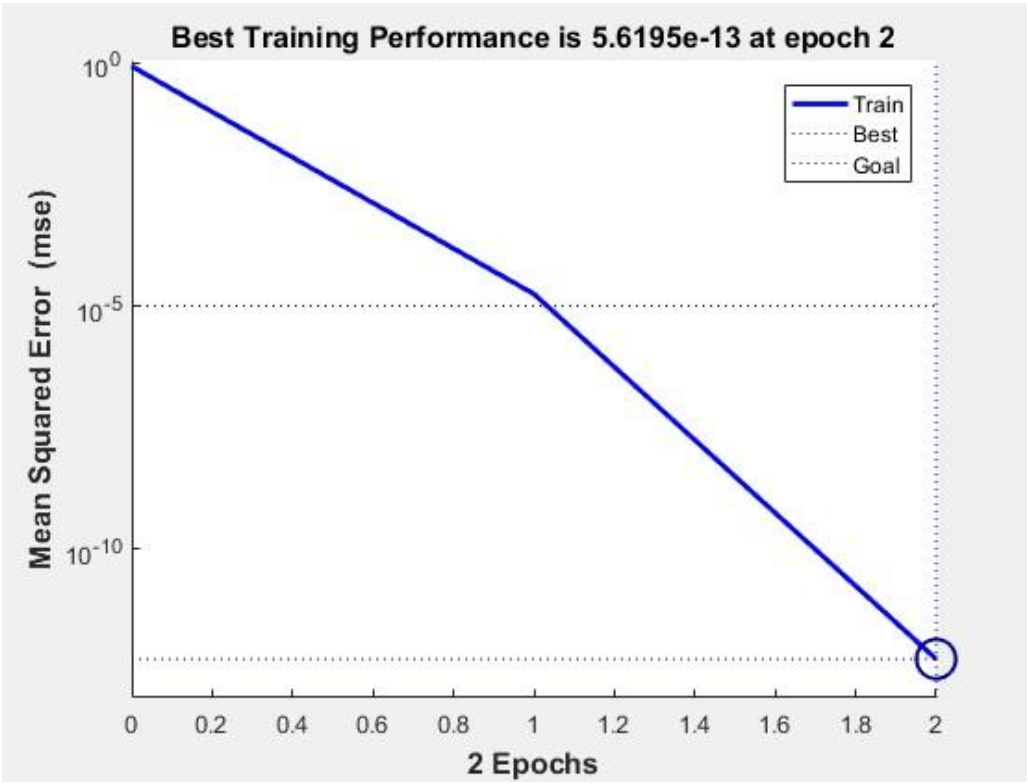
W celu analizy nauki perceptronu poddawałam zmianie wartość współczynnika uczenia oraz liczbę danych uczących. Wartości te mają ogromny wpływ na naukę perceptronu. Błąd uczenia jak i szybkość nauki zmieniały się wraz ze zmianą współczynników. Na przykład dla współczynnika uczenia wynoszącego 0.01 szybkość nauki trwała znacznie dłużej niż dla współczynnika wynoszącego 0.2, przy którym była konieczna mniejsza ilość wywołań trenowania. Zarówno dla danych uczących składających się z szesnastoliczbowego zbioru czas nauki był krótszy niż dla zbioru czteroliczbowego.

Rozwój naszego trenowania prezentuje się następująco:



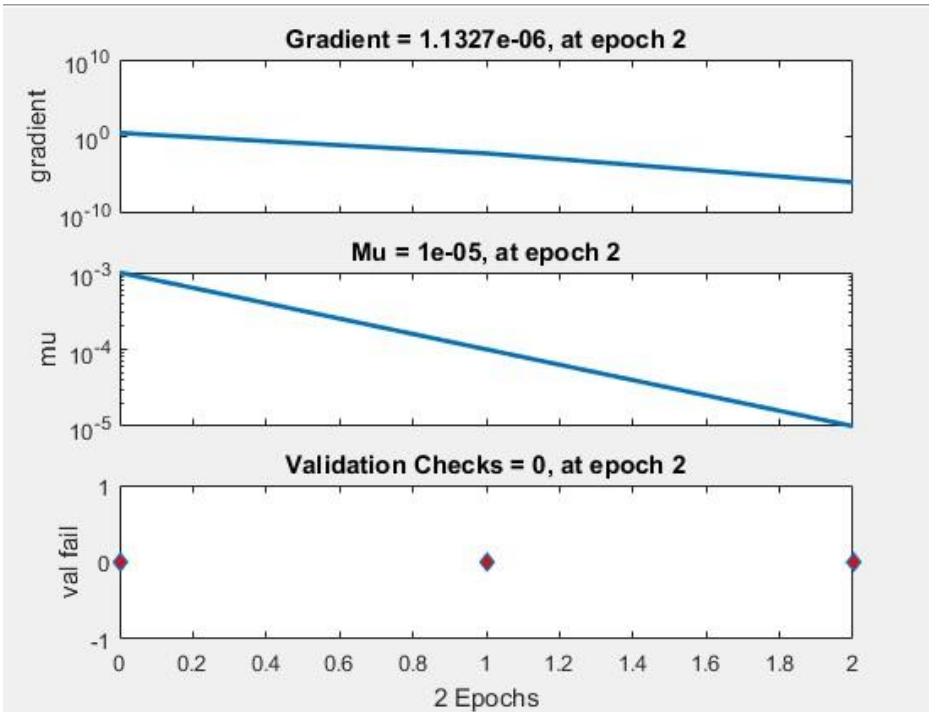
Widzimy, że czas treningu jest bardzo krótki, wykonuje się w zaledwie 2 iteracje.

Wykres wydajności prezentuje się następująco:



Mamy tutaj ukazaną iterację, dla której jest najlepsza wydajność wynosząca 5.62e-13.

Wykresy rozwoju pozostałych zmiennych prezentują się następująco:



4. Wnioski.

Po wykonaniu testów można zauważyć, że perceptron uczy się tym szybciej im współczynnik uczenia się jest większy. Liczba danych uczących również przyśpiesza naukę, ale w mniejszym stopniu. Podczas nauki zauważyłam, że początkowo wyniki są zbliżone do poprawnych, np.:

```
po_treningu =

    0.9702    0.0067    0.9941    0.9746
```

Dopiero po wykonaniu kolejnych testów zbliżają się one coraz bardziej do oczekiwanych rezultatów, aż w końcu osiągają wartości funkcji OR. Należy też uważać, aby nie przetrenować perceptronu. Gdy wyniki są już zadowalające należy przerwać naukę, ponieważ po przekroczeniu pewnej granicy wyniki zaczynają się pogarszać.

5. Listing całego kodu programu

```
close all; clc;
P=[1 0 0 1  ; 1 0 1 0];
T=[1 0 1 1];
net=newff(minmax(P),[10 1], {'logsig', 'purelin', 'trainlm'});
przed_treningiem=sim(net,P);
net.trainparam.epochs=50;
net.trainparam.goal=0.00001;
net.trainparam.lr=0.2;
[net,tr]=train(net,P,T);
po_treningu=sim(net,P);
```