|  |  |  |
| --- | --- | --- |
| Imię i nazwisko:  Adrian Popielec | Grupa projektowa:  I | Data oddania:  23.01.2017 r. |

***Podstawy sztucznej inteligencji - sprawozdanie***

**1. Podstawowy sztuczny neuron, model MCP**

Sztuczny neuron jest to prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jedynym wyjściu. Neuron został zaimplementowany w javie. W swoim programie zastosowałem model neuronu McCullocha-Pittsa, w którym wartość na wyjściu neuronu obliczana jest w następujący sposób:

* obliczana jest suma iloczynów wartości xi podanych na wejścia i wag wi wejść:
* na wyjście podawana jest wartość funkcji aktywacji - użyłem funkcji progowej unipolarnej

Problem, który obrałem to rzut ukośny, a dokładniej rzut za lub przed określoną odległość. Do nauczenia neuronu wykorzystałem regułę delta. Wygenerowałem plik posiadający 15 000 danych, na których podstawie neuron dobierał swoje wagi. Następnie program został sprawdzony na 5000 przykładach

|  |  |  |  |
| --- | --- | --- | --- |
| **Najefektywniejsze próby:** | | | |
| **Poprawne:** | **W sumie:** | **% Skuteczność:** | **MSE:** |
| **4530** | **5000** | **90.6** | **215.5202** |
| 4480 | 5000 | 89.6 | 208.3882 |
| 4503 | 5000 | 90.06 | 209.805 |
| 4506 | 5000 | 90.12 | 216.961 |
| 4500 | 5000 | 90 | 214.5624 |
| 4026 | 5000 | 80.52 | 213.1296 |
| 4505 | 5000 | 90.1 | 219.857 |
| 4471 | 5000 | 89.42 | 215.041 |
| 4446 | 5000 | 88.92 | 211.2266 |
| **4491** | **5000** | **89.82** | **207.4464** |

Na czarno została zaznaczona próba, podczas której program osiągnął największą skuteczność, ponad 90%. Na zielono jest zaznaczona próba, podczas której wartość błędu średniokwadratowego MSE była najniższa.

Wnioski:

Zaimplementowany neuron osiągnął maksymalną wartość skuteczności ponad 90%, co pokazuje efektywność nauki. Nie jest jednak w stanie nauczyć się bezbłędnie, ponieważ problem jest dla niego zbyt złożony, składa się z dwóch wejść - jedno, jest to liczba z zakresu 1-30, drugie to liczba z zakresu 1-89. Wartości nauki są porównywane z nauczycielem, czyli wzorem matematycznym na zasięg w rzucie ukośnym:

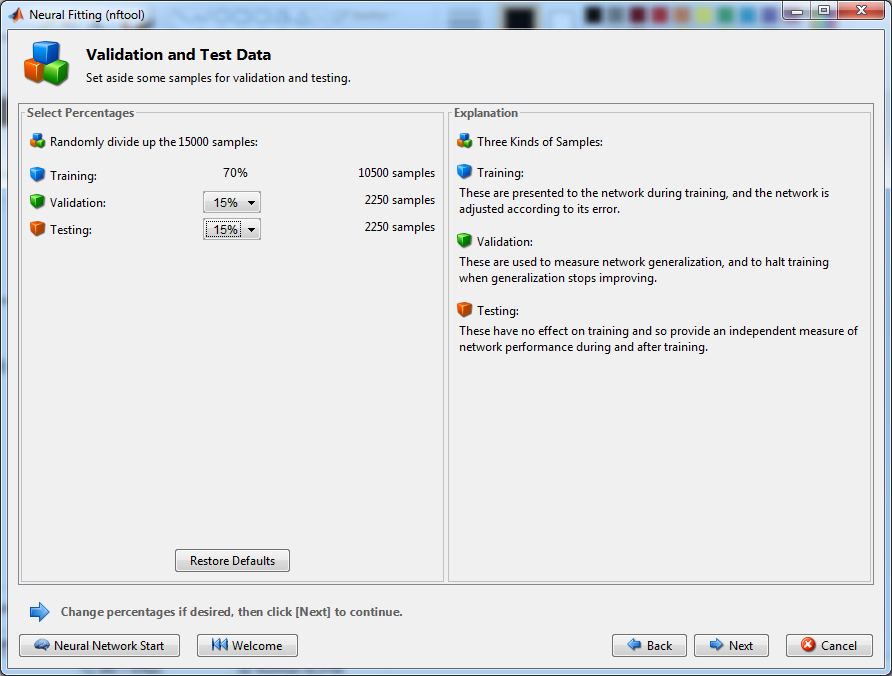
http://www.fizykon.org/wzory/wz_rzut_ukosny_zas_rz.gif

Wzór jest dosyć złożony, daje wiele możliwości, dlatego programu nie da się wyuczyć do końca.

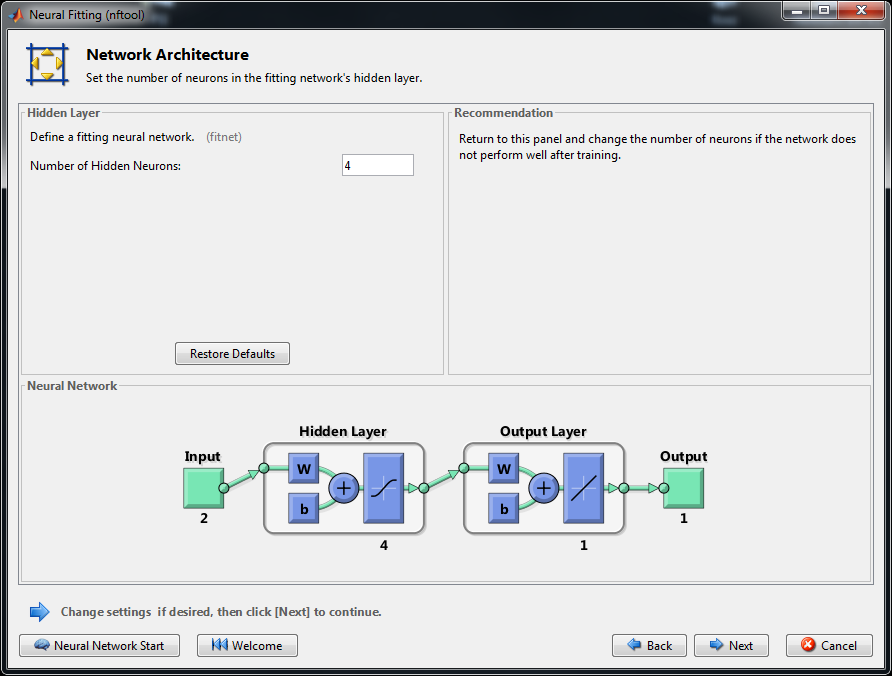
**2. Sieć wielowarstwowa**

Wielowarstwowa sieć neuronowa jest w stanie przybliżać dowolnie złożone i skomplikowane odwzorowanie. Przy czym użytkownik nie musi znać lub zakładać z góry żadnej formy występujących w poszukiwanym modelu zależności, nie musi nawet zadawać sobie pytania czy jakiekolwiek możliwe do matematycznego modelowania zależności w ogóle występują. Cecha ta w połączeniu z samodzielną metoda uczenia się sieci neuronowej, czyni z niej niezwykle użyteczne i wygodne narzędzie do wszelkiego rodzaju zastosowań związanych z prognozowaniem, klasyfikacją lub automatycznym sterowaniem. Dodatkową zaletą jest duża tolerancja systemu neuronowego na uszkodzenia poszczególnych jej elementów, co podnosi niezawodność w sposób praktycznie niemożliwy do osiągnięcia w przypadku tradycyjnego oprogramowania, np. klasycznych systemów ekspertowych. Na tym nie kończy się lista zalet, równoległy model przetwarzania sygnałów wejściowych i stosunkowo prosta struktura nauczonej sieci neuronowej pozwala osiągnąć bardzo szybki czas reakcji, pozwalający na wykorzystywanie jej w zadaniach wymagających reagowania w czasie rzeczywistym.

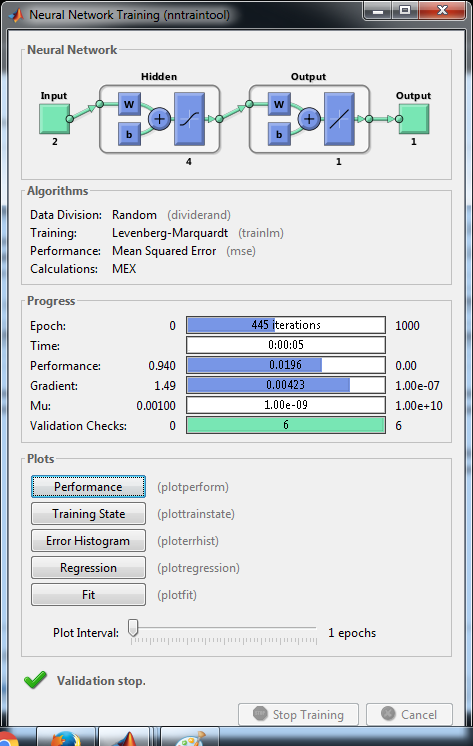
Sieć wielowarstwową utworzyłem posługując się programem Matlab, jako problem został obrany rzut ukośny. Matlab umożliwia gotowe narzędzia do tworzenia sieci neuronowych. Dane wejściowe zostały podzielone na dane uczące, walidujące i testujące. Jest to 15 000 rekordów, dwa wejścia.



Ustawienia sieci: dwa wejścia, cztery neurony ukryte, jedno wyjście

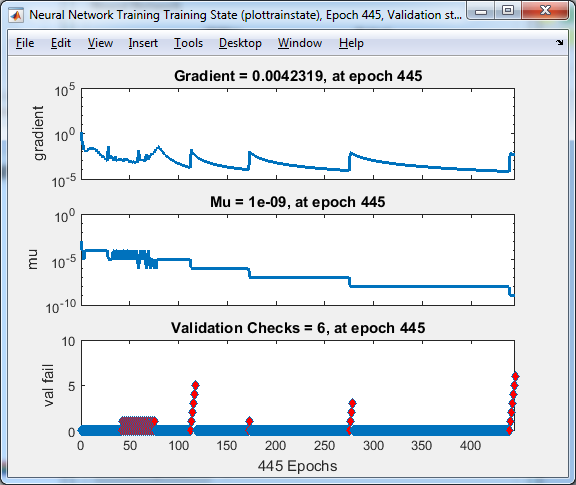
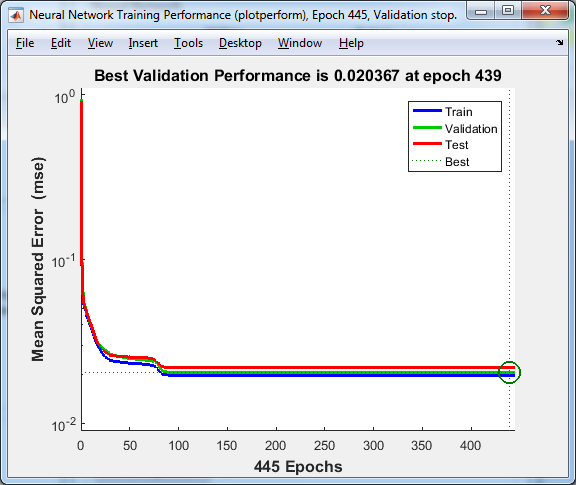


Algorytm według którego sieć będzie się uczyć nazywa się algorytmem Levenberga-Marquardta – jest to iteracyjny algorytm optymalizacji nieliniowej, łączący w sobie cechy metody największego spadku i metody Gaussa-Newtona.

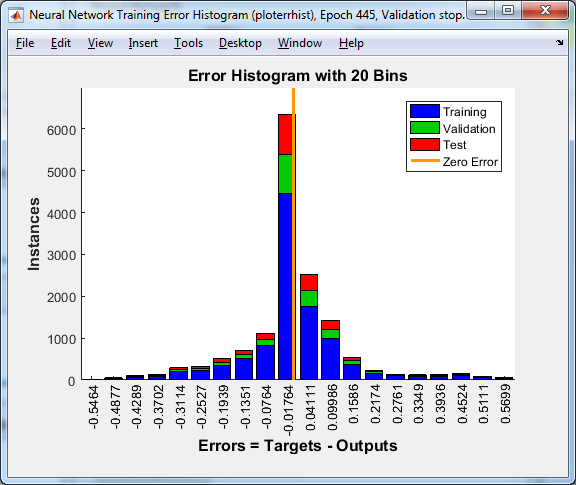


Program osiągnął swoją najniższą wartość MSE po 439 iteracjach

Wykres MSE w kolejnych iteracjach, gradientu oraz przyrostu treningowego (MU)



Histogram przedstawia wartości błędów z uwzględnieniem podziału danych na uczące, walidujące, testujące.



Wnioski:

Ograniczenia pojedynczych perceptronów można zminimalizować stosując sieci neuronowe. Ich prosta architektura polegajaca na tym, że tworzymy kilka (zwykle 3) warstw neuronów połączonych w ten sposób, że wyjścia neuronów należących do warstwy niższej połączone są z wejściami neuronów należących do warstwy wyższej (każdy z każdym), pozwala na tworzenie sieci o niemal dowolnej charakterystyce. Działanie takiej sieci polega na liczeniu odpowiedzi neuronów w kolejnych warstwach - najpierw w pierwszej, do której trafiają sygnały z wejść sieci, potem (na podstawie wyników pierwszej warstwy) liczymy odpowiedzi drugiej warstwy neuronów itd., przy czym odpowiedzi ostatniej warstwy traktowane są jako wyjścia z sieci. Funkcja aktywacji neuronów nie powinna być skokowa. W praktyce wykorzystuje się kilka rodzajów funkcji aktywacji, dbając jedynie, by była ona monotoniczna i różniczkowalna. Porównując wyniki z powyższej sieci neuronowej oraz pojedyńczym neuronem z poprzedniego zadania, wyniki są podobne, jednak w sieci neuronowej efekt ten jest otrzymywany zdecydowanie szybciej.

**3. Sieć wielowarstwowa, alternatywna wersja**

Dla tej samej sieci neuronowej przeprowadziłem uczenia korzystając z trzech różnych sposobów nauki

*Pierwszy zestaw danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Metoda nauki | **Bayesian Regularization** | **Levenberg-Marquardt** | **Scaled Conjugate Gradient** |
| Ilość iteracji | 62 | 8 | 18 |
| MSE | 0.103 | 0.103 | 0.106 |
| Gradient | 8.55e-05 | 0.000932 | 0.00837 |

*Drugi zestaw danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Metoda nauki | **Bayesian Regularization** | **Levenberg-Marquardt** | **Scaled Conjugate Gradient** |
| Ilość iteracji | 353 | 8 | 16 |
| MSE | 0.103 | 0.102 | 0.103 |
| Gradient | 6.16e-05 | 0.000526 | 0.000356 |

*Trzeci zestaw danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Metoda nauki | **Bayesian Regularization** | **Levenberg-Marquardt** | **Scaled Conjugate Gradient** |
| Ilość iteracji | 401 | 8 | 40 |
| MSE | 0.101 | 0.101 | 0.103 |
| Gradient | 6.90e-05 | 0.00177 | 0.00126 |

Na podstawie otrzymanych wyników zdecydowanie możemy podzielić techniki nauki. Na ostatnim miejscu, ze względu na dużą ilość iteracji uporządkuje metodę Bayesian'a. Resztę wyników ma na podobnym poziomie, natomiast w ilości iteracji, a co za tym idzie czasu, odstaje od reszty. Na drugim miejscu metoda wstecznej propagacji, dobre wyniki, ale również większa ilość iteracji. Najbardziej efektywna i najszybsza metoda nauki wg powyższych danych to algorytm Levenberg-Marquardt.

**4. Reguła Hebba, reguła Oji**

Program został zaimplementowany w javie. Główna różnica między nauczaniem zwykłego neuronu oraz powyższych reguł odnosi się do wzoru modyfikacji wag, oraz do faktu, że powyższe reguły nie uczą się od nauczyciela - same modyfikują swoje wagi.

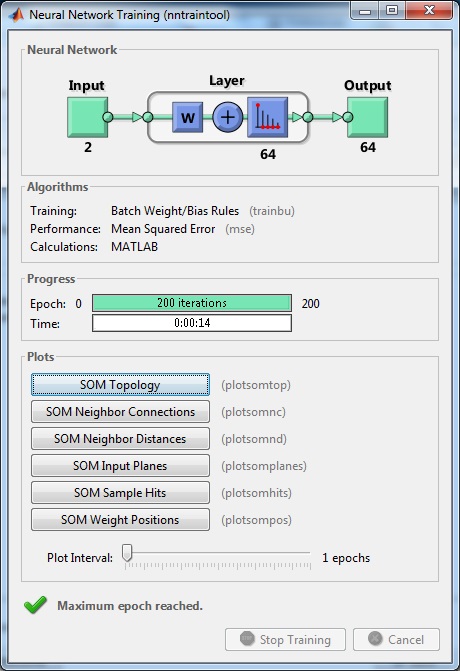
Problem postawiony w tym zadaniu to rzut ukośny, a dokładniej rzut za daleko lub za blisko określonego celu. Program nie działa do końca poprawnie, są dwie możliwości nauki:

pierwsza - program osiąga wartość MSE = 15000 (czyli maksymalną), efektywność nauki to 11%. Druga - program osiąga wartość MSE = 0 (najniższą), efektywność nauki to 89%.

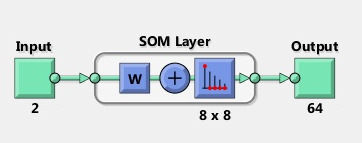
**5. Sieci samoorganizujące**

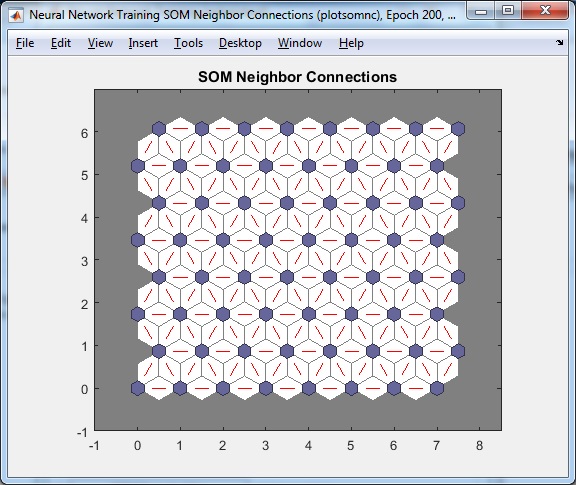
System, który miałby realizować funkcjonowanie sieci samoorganizującej powinien składać się z kilku podstawowych elementów. Pierwszym z nich jest macierz neuronów pobudzanych przez sygnały wejściowe. Sygnały te powinny opisywać pewne charakterystyczne cechy zjawisk zachodzących w otoczeniu, tak, aby na ich podstawie sieć była w stanie je pogrupować. Informacja o zdarzeniach jest przekładana na bodźce pobudzające neurony. Kolejną częścią składową sieci jest mechanizm, który dla każdego neuronu określa stopień podobieństwa jego wag do danego sygnału wejściowego oraz wyznacza jednostkę z największym dopasowaniem - zwycięzcę. W trakcie uczenia wagi te są modyfikowane w taki sposób, aby najlepiej odzwierciedlać wewnętrzną strukturę danych wejściowych. Wreszcie konieczne do przeprowadzenia samoorganizacji jest, aby sieć była wyposażona w zdolność do adaptacji wartości wag neuronu zwycięzcy i jego sąsiadów w zależności od siły, z jaką odpowiedział on na dane wejście. Topologię sieci można w łatwy sposób określić poprzez zdefiniowanie sąsiadów dla każdego neuronu.

Jako przykład własny analizowałem nadal wpływ rzutu ukośnego, z dwoma wejściami oraz odległość rzutu. Korzystałem z programu Matlab, który posiada wygodne rozwiązania do tworzenia sieci SOM. Utworzyłem sieć o dwóch wejściach oraz 8x8 warstwie neuronów, czyli 64. Dane wejściowe to 15 000 rekordów, które zostały podzielone na dane uczące, walidujące i testujące.

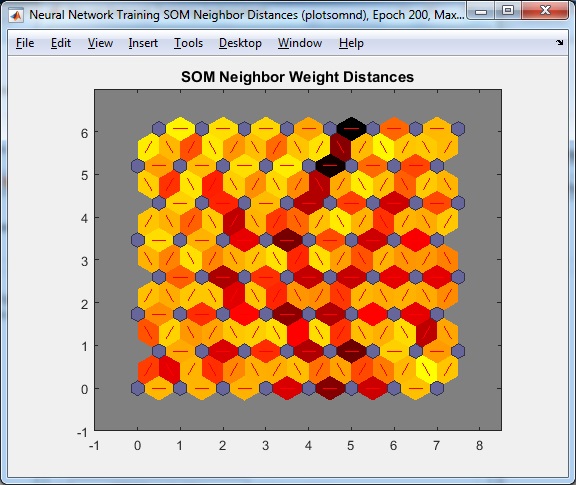


Budowa sieci SOM

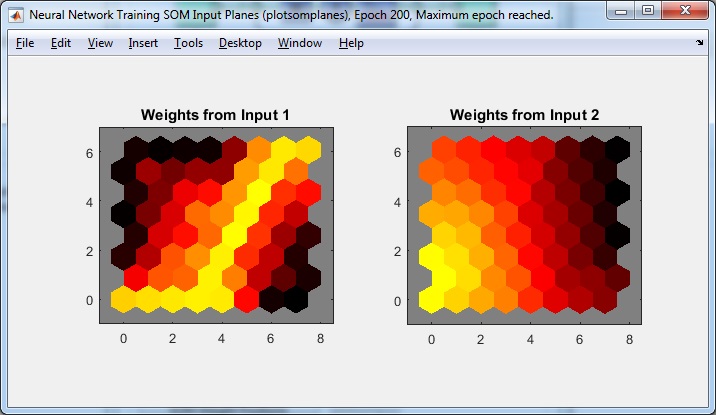




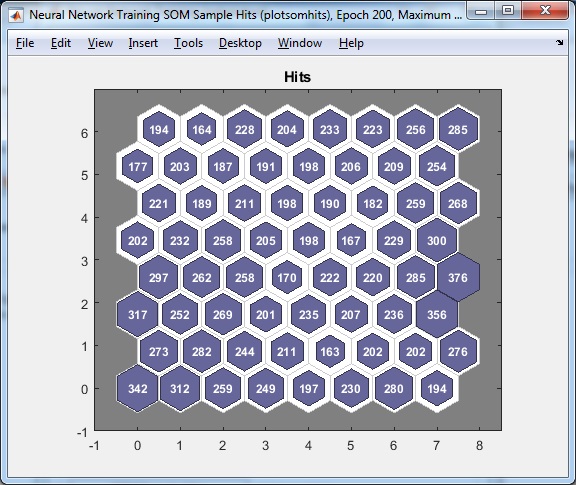
Mapa sieci przedstawiona w sposób wizualny - połączenia między sąsiadującymi neuronami



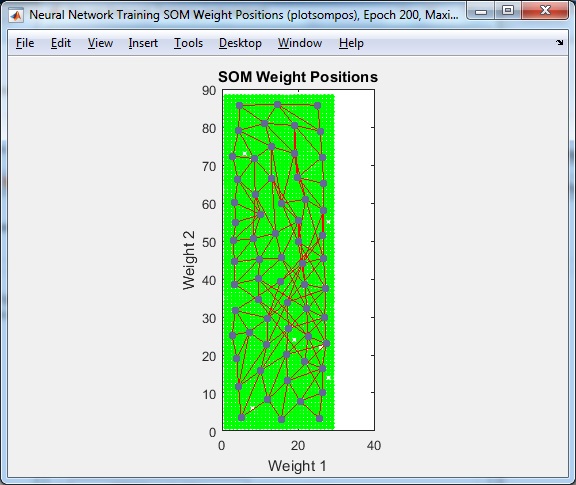
Mapa przedstawiajaca grupowanie neuronów w sieci, na podstawie podobieństwa ich wag.



Sieć wag dla każdego wejścia (dopasowanie wag neuronów do sygnału wejściowego)



Sieć pokazująca jak często występowało "trafienie" w dany neuron, czyli ile razy neurony były zwycięzcami (najbardziej zbliżone wagi do sygnałów wejściowych)



Połączenia między sąsiadującymi neuronami (neurony są uporządkowane wg podobieństwa wag)

Wnioski:

Najciekawsze efekty uzyskuje się wprowadzające sieci SOM dla sieci neuronowych dwuwymiarowych. Istotnym parametrem jest określenie ile neuronów obok ma podlegać uczeniu w przypadku zwycięstwa danego neuronu. Do określania miar odległości używa się:

* miary euklidesowej
* iloczynu skalarnego
* miary według normy L1
* miary według normy L

Najbardziej popularnym typem sieci samoorganizującej jest sieć Kohonena. W sieci tej mamy do czynienia z uczeniem konkurencyjnym. Sieć ma na celu utworzenie takiej struktury, która w najlepszy sposób będzie odwzorowywała zależności w przestrzeni wektorów wejściowych.