**Sieci Kohonena** – SOM (z ang. Self-organizing Maps, czyli samoorganizujące się mapy). Swój sukces zawdzięczają one temu, że pozwalają stworzyć mapy przestrzeni bardzo wysokich wymiarów, a uczenie takich sieci nie wymaga nadzoru.

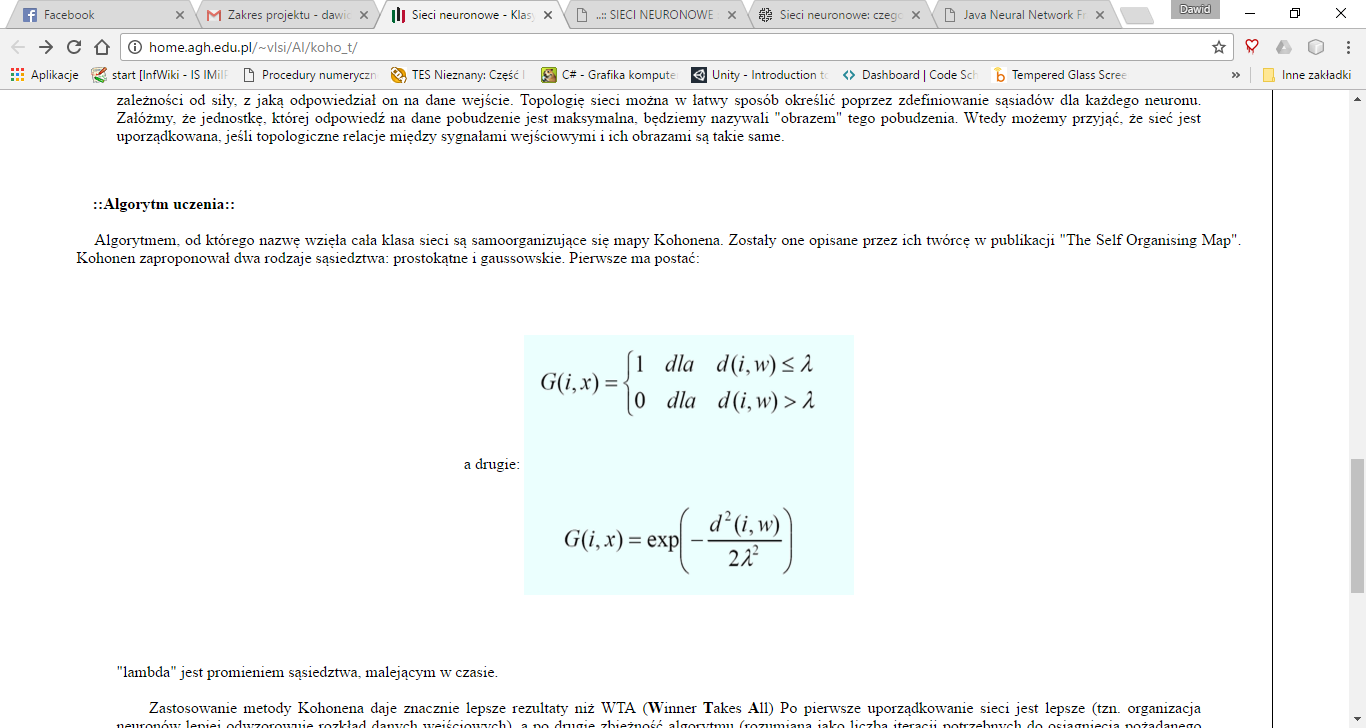
Są smoorganizujące się. Dzięki temu mogą adaptować się do wcześniej nieznanych danych wejściowych. Uczenie odbywa się metodą samoorganizującą typu konkurencyjnego. Polega ona na podawaniu na wejścia sieci sygnałów, a następnie wybraniu w drodze konkurencji zwycięskiego neuronu, który najlepiej odpowiada wektorowi wejściowemu.

 Funkcjonowanie samoorganizujących się sieci neuronowych odbywa się w trzech etapach:

* konstrukcja
* uczenie
* rozpoznawanie

System, który miałby realizować funkcjonowanie sieci samoorganizującej powinien składać się z kilku podstawowych elementów. Pierwszym z nich jest macierz neuronów pobudzanych przez sygnały wejściowe. 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. Obliczenia zaczynamy dla wag równych małym liczbom losowym, przy czym ważne jest, aby nie zachodziła żadna symetria. W trakcie uczenia wagi te są modyfikowane w taki sposób, aby najlepiej odzwierciedlać wewnętrzną strukturę danych wejściowych. reszcie 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.

Algorytmem, od którego nazwę wzięła cała klasa sieci są samoorganizujące się mapy Kohonena. Kohonen zaproponował dwa rodzaje sąsiedztwa: prostokątne i gaussowskie. Pierwsze ma postać:

, lambda" jest promieniem sąsiedztwa, malejącym w czasie.

**Algorytm** (mniej-więcej):

1. Dane treningowe – wektory w n wymiarach = każdy węzeł zawiera podobny wektor wag w n wymiarach.
2. Wagi węzłów zostają zainicjalizowane (losowo)
3. Każdy węzeł jest brany pod uwagę, czy to nie jego wagi są najbardziej zbliżone do wektora wejściowego. Ten, który wygrał – BMU (Best Matching Unit):

Robimy to poprzez iterowanie po wszystkich węzłach i liczenie odległości pomiędzy wektorem wagi i wektorem wejściowym. Węzeł z tą odległością najmniejszą – BMU.

  
for (int i=0; i<m\_dWeights.size(); ++i)

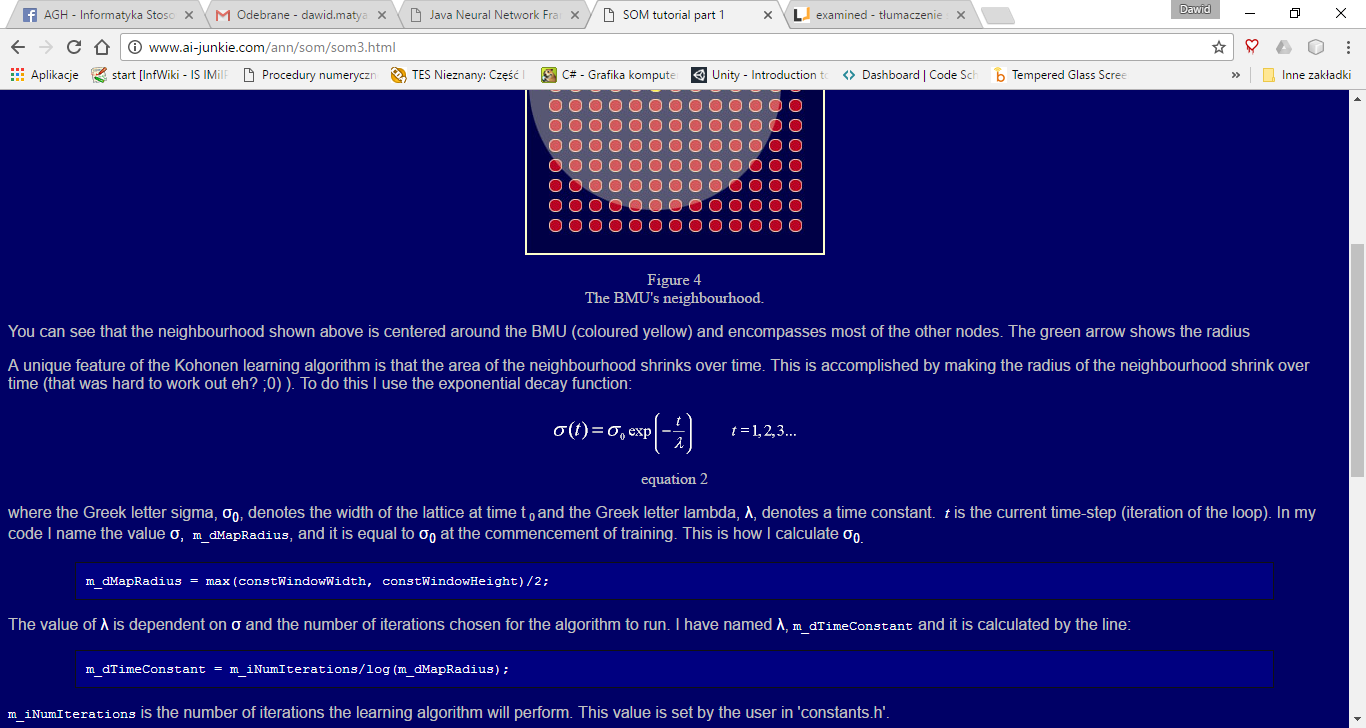
  {

    distance += (InputVector[i] - m\_dWeights[i]) \* (InputVector[i] - m\_dWeights[i]);

  }

  return sqrt(distance);

1. Obliczanie lokalnego sąsiedztwa dla BMU:  
   obszar sąsiedztwa zmienia się, waha się nawet do rozmiaru pojedynczego węzła BMU



gdzie:  
sigma0: m\_dMapRadius = max(constWindowWidth, constWindowHeight)/2;  
lambda: m\_dTimeConstant = m\_iNumIterations/log(m\_dMapRadius);  
końcowy promień zależny od czasu:   
m\_dNeighbourhoodRadius = m\_dMapRadius \* exp(-(double)m\_iIterationCount/m\_dTimeConstant);

1. Dostosowywanie wag z sąsiedztwa:  
   http://www.ai-junkie.com/ann/som/images/equation4.gif  
   gdzie: t – kroki czasowe (jak poprzednio), L – współczynnik nauki (zmniejsza się wraz z upływem czasu).

http://www.ai-junkie.com/ann/som/images/equation3.gif

1. Powtarzanie kroków od 2. dla wszystkich iteracji