Zagadnienia Sztucznej Inteligencji – laboratorium

Sieci Neuronowe 2

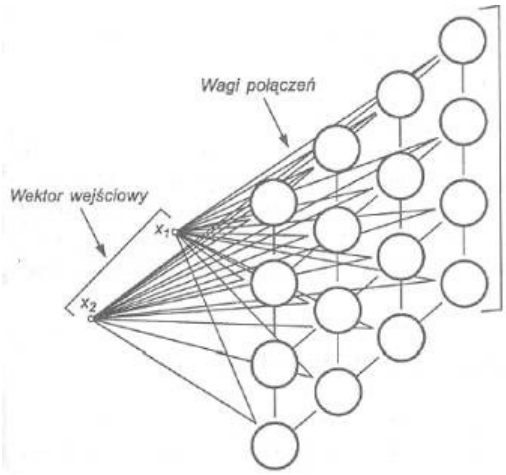
Michał Bereta

Cele laboratorium:

- zapoznanie się z nowymi rodzajami sieci neuronowych: sieciami Kohonena oraz sieciami radialnymi
- porównanie sieci Kohonena oraz sieci wielowarstwowego perceptronu w problemie klasyfikacji pisma odręcznego
- zastosowanie sieci neuronowej do aproksymacji funkcji

Samoorganizujące się sieci Kohonena

Sieci Kohonena (zwane też ***SOM*** – Self Orgenizing Maps) są przykładem sieci uczonych bez nadzoru (bez nauczyciela). Oznacza to, że sieci nie są prezentowane wzorcowe odpowiedzi a jedynie dane wejściowe. Neurony w sieci Kohonena są uporządkowane względem siebie przestrzennie. Często realizowane jest to za pomocą siatki dwuwymiarowej. Zostało to pokazane na rysunku 1:



Rysunek 1 Siec Kohonena

Można zatem mówić o sąsiedztwie danego neuronu. Sąsiedztwem tym są inne neurony leżące blisko danego neuronu w przestrzeni wyznaczonej przez zastosowane uporządkowanie neuronów.

Uczenie polega na wyznaczeniu zwycięskiego neuronu i modyfikacji jego wektora wag **w** w kierunku prezentowanego wektora wejściowego **x**. Zwycięskim neuronem jest ten neuron, którego wagi synaptyczne są najbardziej zbliżone do prezentowanego wektora danych wejściowych. Zatem wektor wag $\mathbf{w_w}$ zwycięskiego neuronu wyznacza się za pomocą:

$$d(x, w_w) = \min_{1 \le i \le n} d(x, w_i)$$

Norma **d** może by dowolną normą (normą Euklidesową, iloczynem skalarnym itd.). W trakcie uczenia zwycięzca oraz jego sąsiedztwo podlega adaptacji wag zgodnie z regułą Kohonena:

$$w_i(k+1) = w_i(k) + \eta_i(k)[x - w_i(k)]$$

Parametr **eta** krok uczenia, przeważnie malejący w trakcie kolejnych iteracji. W przypadku danych wejściowych o małych wymiarach istnieje potrzeba normalizacji wektorów wejściowych.

Algorytm WTA(Winner Takes All)

W metodzie tej tylko zwycięski neuron ma prawo adaptacji swoich wag. Aby uniknąć przypadku, w którym niektóre neurony nie mają szans wygrać z najsilniejszymi neuronami, często wprowadza się mechanizm zmęczenia. Polega on na tym, że zwycięski neuron pauzuje na pewien czas nie biorąc udziału w konkurencji, dając tym samym szanse na wygranie innym neuronom (**Algorytm CWTA – Conscience Winner Takes All)**

Algorytm WTM – Winner Takes Most

W metodzie tej oprócz wag zwycięskiego neuronu zmianie polegają również wagi jego sąsiadów według reguły:

$$w_i(k+1) = w_i(k) + \eta_i(k)G(i,x)[x - w_i(k)]$$

gdzie G(x,i) jest funkcją sąsiedztwa. W klasycznym algorytmie Kohonena funkcja G(x,i) zdefiniowana jest jako:

$$G(i,x) = \begin{cases} 1 & dla \quad d(i,w) \le \lambda \\ 0 & dla \quad d(i,w) > \lambda \end{cases}$$

gdzie d(i,w) jest odległością pomiędzy neuronami w przestrzeni sieci tzn. na siatce (a nie w przestrzeni danych wejściowych!), a lambda jest promieniem sąsiedztwa malejącym do zera w trakcie nauki. Jest to tzw. sąsiedztwo prostokątne. Innym rodzajem sąsiedztwa jest sąsiedztwo gaussowskie:

$$G(i,x) = \exp(-\frac{d^2(i,w)}{2\lambda^2})$$

W tym przypadku stopień aktywacji neuronów jest zróżnicowany w zależności od wartości funkcji Gaussa. Sąsiedzi neuronu zwycięskiego są modyfikowani w różnym stopniu.

Często stosowanym podejściem jest wyróżnienie dwóch faz uczenia sieci Kohonena:

- wstępną (ang. *ordering phase*), podczas której modyfikacji ulegają wagi neuronu zwycięzcy oraz jego sąsiedztwa malejącego z czasem
- końcową (ang. *convergence phase*), podczas której modyfikowane są wagi jedynie neuronu zwycięzcy.

Mają one na celu najpierw wstępne nauczenie sieci, a następnie jej dostrojenie.

Zastosowanie sieci Kohonena:

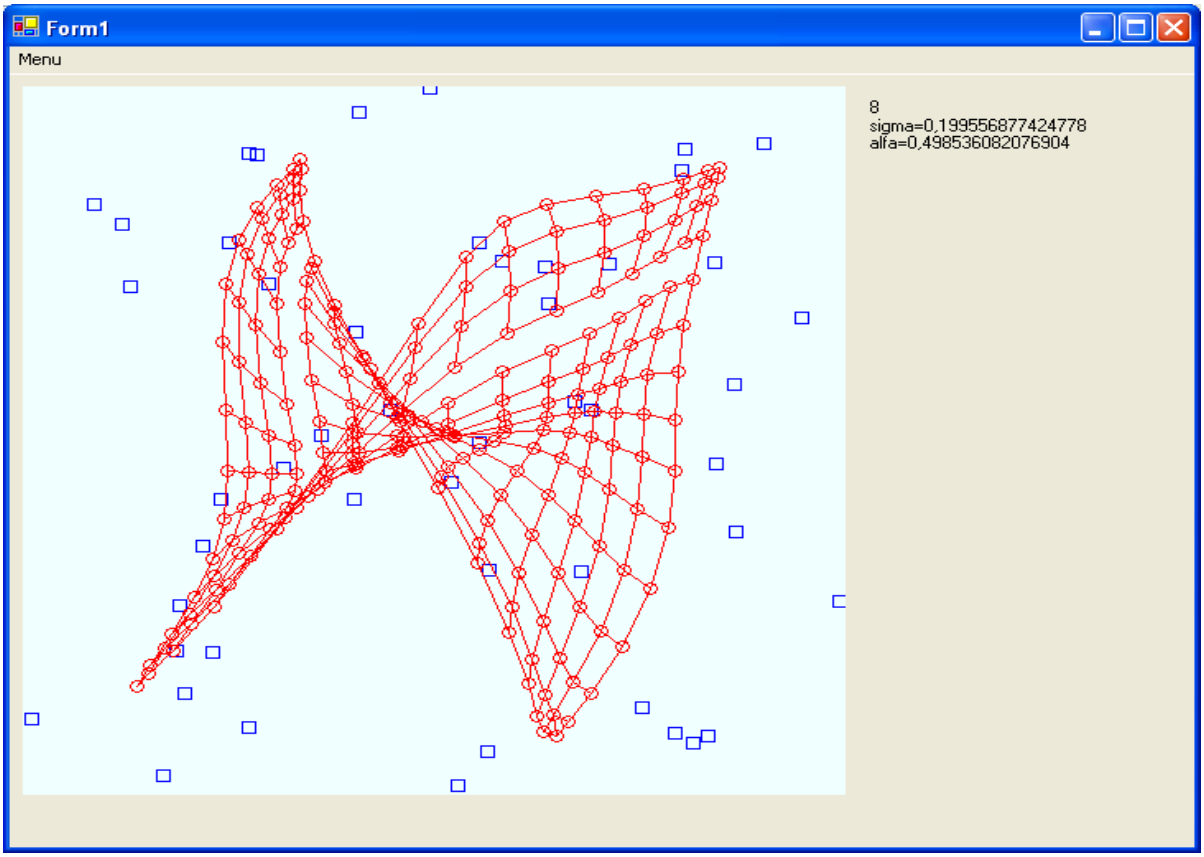
- wykrywanie związków w zbiorze wektorów
- uczenie się rozkładu danych
- kompresja obrazów
- wykrywanie nieprawidłowości

Przykład : Wizualizacja sieci Kohonena na płaszczyźnie

Pobierz ze strony program ***Kohonen.exe.*** Prezentuje on proces nauki sieci Kohonena w przypadku danych uczących dwuwymiarowych. Również neurony tej sieci są uporządkowane na dwuwymiarowej siatce. Nie musi to wcale być regułą. Należy zapamiętać, ze wymiarowość danych wejściowych nie musi być równa wymiarowości siatki, na której są uporządkowane neurony Kohonena. W przykładzie tym przyjęto tak, gdyż ułatwia to wizualizację działania algorytmu. Można stosować różne topologie, niekoniecznie siatkę dwuwymiarową, jednak jest ona jedną z najpopularniejszych.
Przed przystąpieniem do analizy demonstracji, warto podkreslić, co jest prezentowane na ekranie. Niebieskie kwadraty to dane uczące. Jak widać, są to tylko dane wejściowe, nie ma odpowiedzi wzorcowych, uczenie

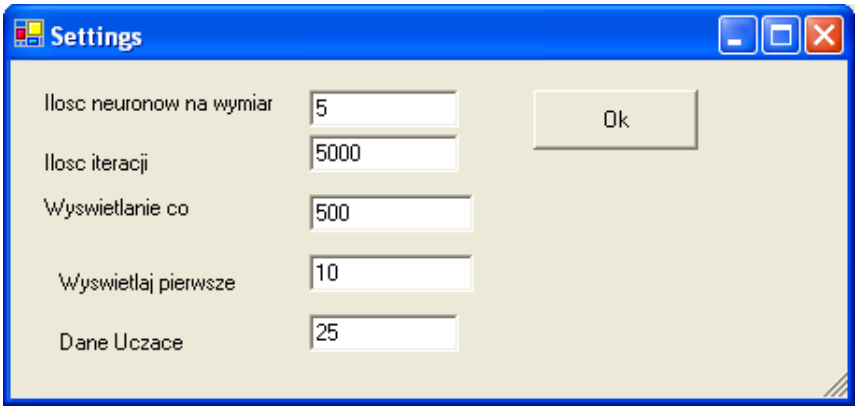
zatem jest nienadzorowane. Czerwone kółka reprezentują neurony. Połączenia między nimi pokazują, które neurony są sąsiadami. Jednak pozycja każdego neuronu na ekranie to nie jego współrzędne w przestrzeni tej sieci (na siatce), lecz raczej wartości jego wag. Pozwala nam to zaobserwowac, jak bardzo dany neuron zdołał zaadaptować swe wagi w kierunku konkretnej danej uczącej. Widzimy również, jak jego sąsiedzi są przez niego „ciągnięci". Siła tego „ciągnięcia" maleje w trakcie kolejnych iteracji.

Wizualizacja taka jest możliwa dlatego, że wymiarowość siatki jest taka sama jak wymiarowość danych uczących. Siatka często jest dwuwymiarowa, natomiast dane mogą być w praktyce wysokowymiarowe. W takich przypadkach dwuwymiarowe sieci Kohonena stosuje się do wizualizacji rozkładu danych wysokowymiarowych poprzez prezentację nauczonej za pomocą takich danych dwuwymiarowej sieci Kohonena.



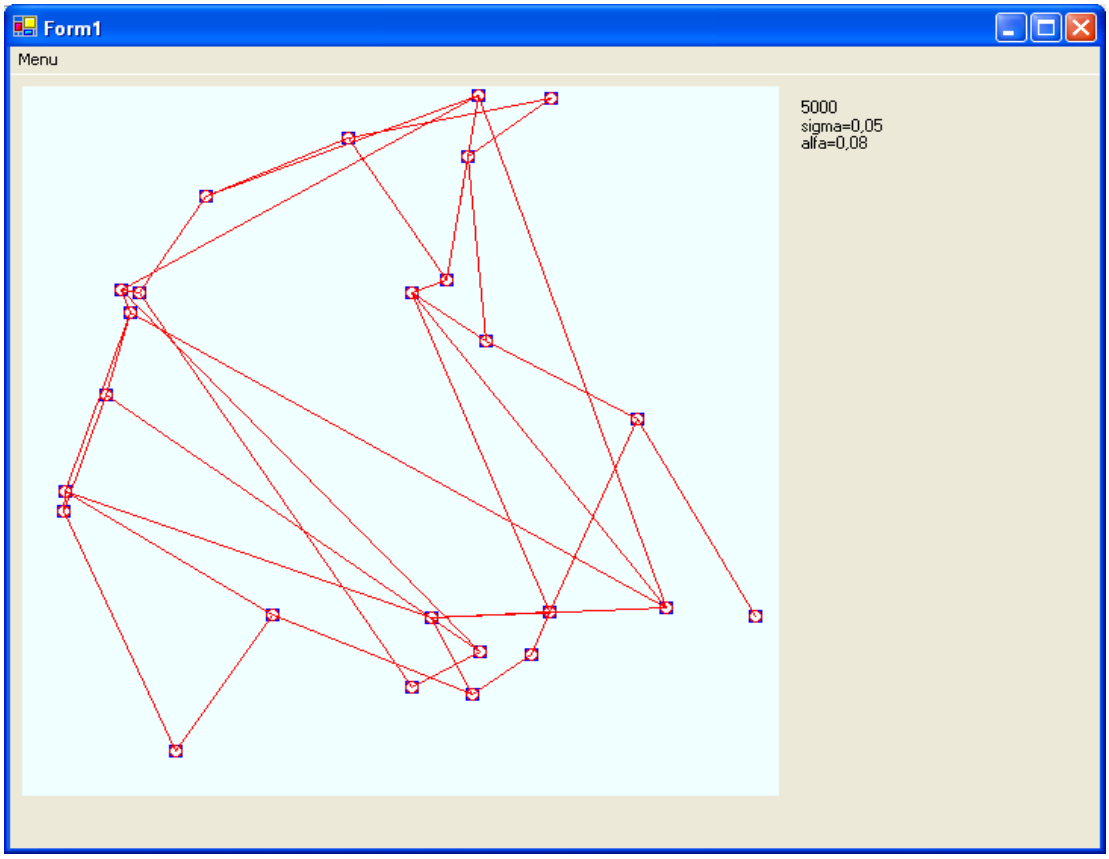
Rysunek 2: Program demonstrujący uczenie sieci Kohonena

Zadanie: Przyjrzyj się działaniu sieci dla różnych ustawień parametów. Przykładowo, nastepujące ustawienia:



Rysunek 3: Parametry algorytmu Kohonena

spowodują, że neuronów będzie tyle samo, co danych uczacych (25), zatem każdy neuron powinien znaleźć się w pobliżu „swojej” danej uczącej. Przykład pokazano na rysunku 4.



Rysunek 4: Przykład nauczonej sieci Kohonena

Można również zastosować algorytm Kohonena przy założeniu, że sąsiedztwo neuronów liczone jest nie w przestrzeni sieci, lecz w przestrzeni wag (a więc tej samej co danych wejściowych). Taka zmiana (rezygnacja z ułożenia neuronów w sztywną sieć) pozwala mówić raczej o zbiorze „luźnych” neuronów niż o sieci. Cały mechanizm adaptacji wag zwycięzcy oraz jego sąsiadów pozostaje natomiast niezmieniony – zmienia się tylko przestrzeń, w której szukamy tych sąsiadów.

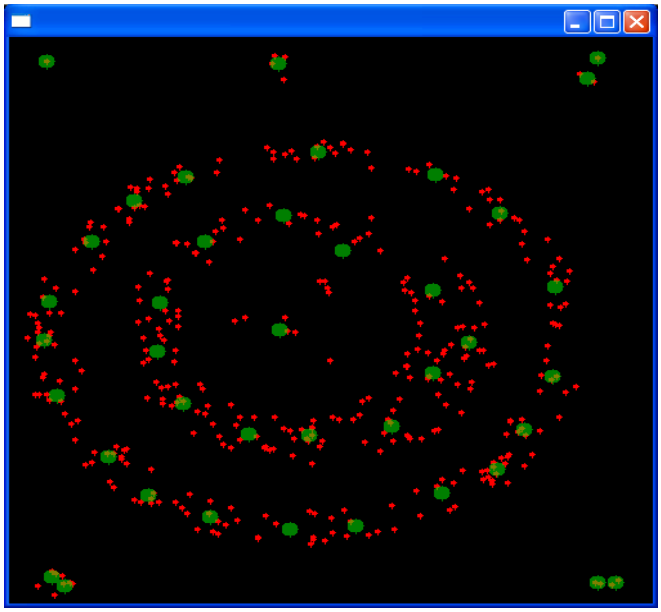
Taka odmiana uczenia nienadzorowanego w postaci algorytmu Kohonena służyć może do znalezienia rozkładu danych uczacych, ich podziału na grupy. Podział ten polega na tym, że wszystkie dane, dla których dany neuron jest zwycięzcą, stanowią jedną grupę (jedną klasę). Cały zbiór neuronów uczy się zatem sam, jak podzielić wszystkie dane wejściowe na rozłączne klasy. Sieć tak nauczoną można później wykorzystać do klasyfikacji danych wejściowych, które nie uczestniczyły w uczeniu sieci, poprzez wyznaczenie neuronu zwycięskiego i zaklasyfikowanie nieznanej danej do klasy reprezentowanej przez ten neuron. Jeśli liczba klas jest z góry znana, można przyjąć liczbę neuronów równą liczbie klas. Nic jednak nie stoi na przeszkodzie, by jedna klasa była reprezentowana przez więcej niż jeden neuron.

Innym algorytmem uczenia nienadzorowanego, który może być wykorzystany w tym celu, jest algorytm ***gazu neuronowego.*** Nazwa pochodzi od własności tego algorytmu: zbiór neuronów swoim zachowaniem przypomina zachowanie się cząsteczek gazu. Jego dokladne omówienie wykracza poza zakres tego kursu, jednak w poniższym przykładzie, zobaczymy jak on działa w porównaniu z algorytmem Kohonena.

Przykład : Uczenie rozkładu danych dwuwymiarowych za pomocą algorytmu Kohonena oraz algorytmu gazu neuronowego.

Ze strony pobierz plik z demonstracyjnymi programami ***Kohonen_Grupowanie.exe*** oraz ***Gaz_Neuronowy.exe***. Wykorzytaj pliki ***bat*** w celu zaobserwowania działania algorytmu na różnych danych

wejściowych.



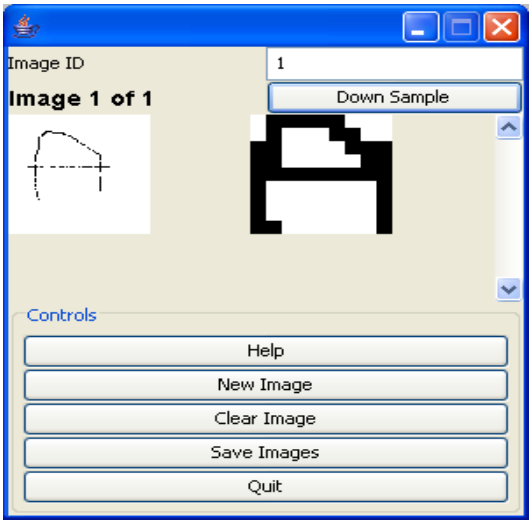
Rysunek 5: Neurony nauczone algorytmem gazu neuronowego.

Pytania: Jakie róznice w działaniu obu algorytmów jesteś w stanie zauważyć? Który z nich wydaje się być lepszy?

Zadanie: Porównanie sieci Kohonena i sieci MLP (wielowarstowy percpetron) w zadaniu klasyfikacji pisma ręcznego .

Aby rozwiązać zadanie, wykorzystamy ponownie środowisko ***JOONE.*** Umożliwia nam ono przygotowanie odpowiednich danych uczacych i testowych. W tym celu:

wybierz ***Help->Examples->SOM Image Tester*** (rysunek 6).

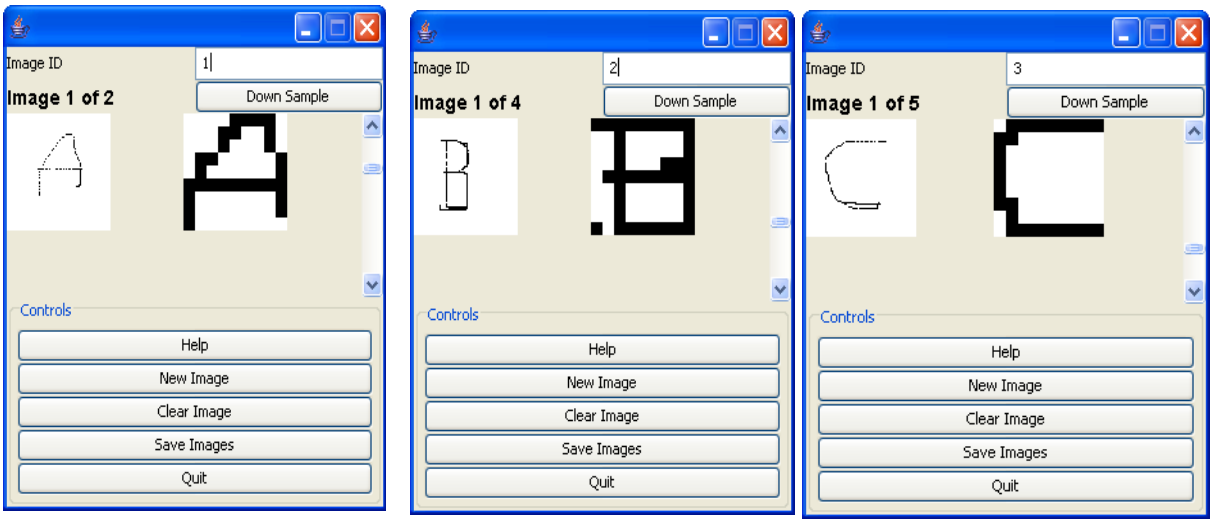


Rysunek 6: Edytor do przygotowania danych uczących i testujących.

Edytor ten pozwala na przygotowanie potrzebnych danych. Narysuj kształt litery ***A*** i nacisnij ***DownSample***. W polu ***ImageID*** należy wpisać identyfikator klasy, do której dany znak należy. Utworzony obraz będzie

przekształcony automatycznie ma obraz 9x9 pikseli binarnych (czanych lub białych). Klikając ***NewImage*** dodajemy kolejne przykłady znaków w analogiczny sposób. Po utworzeniu wszystkich znaków i nadania im etykiet klas (***ImageID***), klikamy ***Save Images,*** co spowoduje zapis danych do pliku w formacie, który możemy następnie wykorzystać w środowisku ***JOONE*** do uczenia sieci.

Przykład: Stworzymy plik z danymi uczącymi ***abc_train.txt*** oraz plik z danymi testowymi ***abc_test.txt***. Do uczenia sieci wykorzytamy pierwszy z nich, natomiast drugi wykorzystamy do przetestowania skuteczności sieci po zakończeniu nauki. W tym przykładzie oba pliki zawierają po dwa przykłady liter ***A***, ***B*** oraz ***C*** z etykietami kolejno 1, 2, 3.



Zawartość pliku ***abc_train.txt***

0.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1

0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1

1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;1.0;1.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;2

1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;2

0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;3

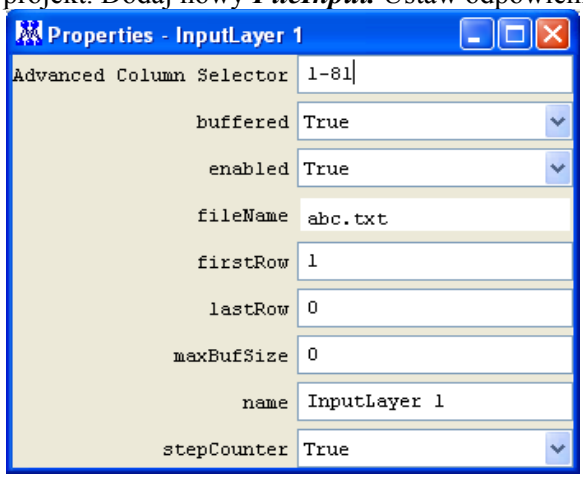
0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;3

Jak widać, jest sześć danych uczących, każda z nich to wektor skladający sie z 81 (9x9) wartości pikseli (1.0-czarny piksel, 0.0-biały piksel). Każdy wektor jest zakończony etykietą klasy, którą reprezentuje.

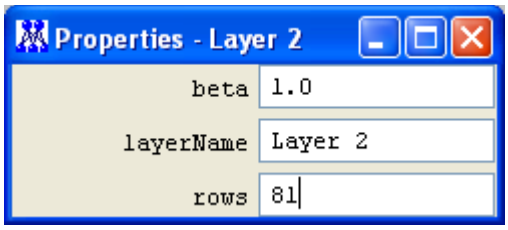
W przypadku drugiego pliku, sytuacja jest analogiczna.

Stworzymy teraz w środowisku ***JOONE*** sieć z neuronami Kohonena, która rozwiąże problem klasyfikacji tak przygotowanych danych opisujących ręcznie pisane znaki.

Otwórz ***JOONE*** i stórz nowy projekt. Dodaj nowy ***FileInput.*** Ustaw odpowienie parametry jak poniżej:

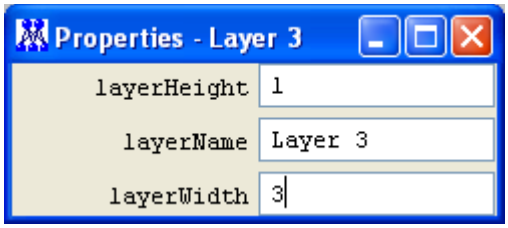


Dodaj ***LinearLayer***, ustaw liczbe neuronów wejściowych na 81.

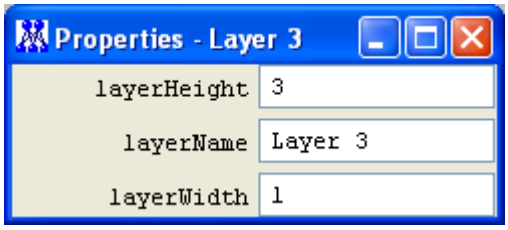


Połącz ***FileInput*** z ***LinearLayer***.

Dodaj nową ***WinnerTakeAllLayer***. Warstwa ta będzie miała 3 neurony, jako, że mamy trzy klasy (trzy różne znaki). Zatem:

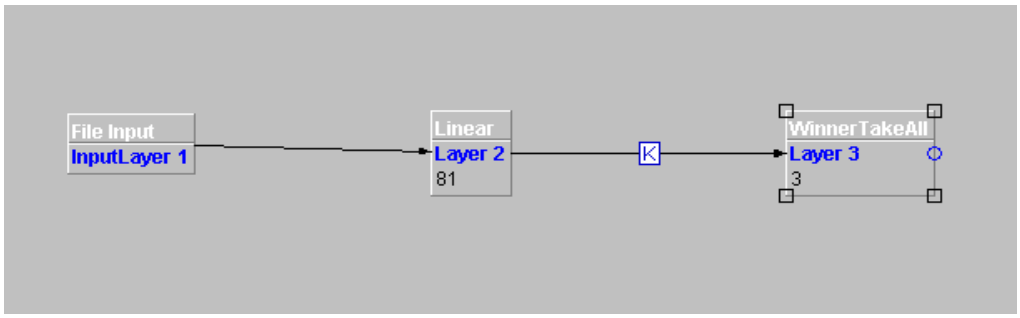


lub

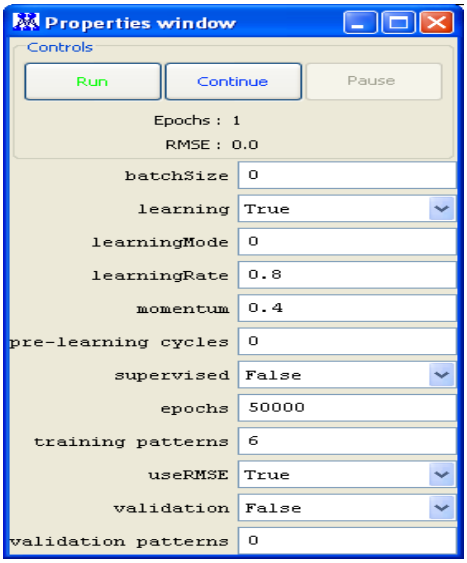


Połącz ***LinearLayer*** z ***WinnerTakeAllLayer*** za pomocą ***KohonenSynapse***.

Efekt:

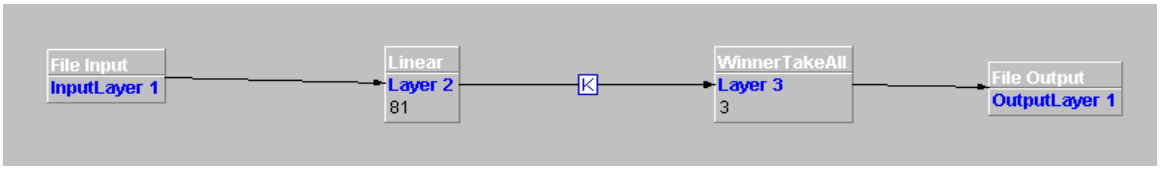


Otwórz ***Tools->ControlPanel*** i ustaw parametry :

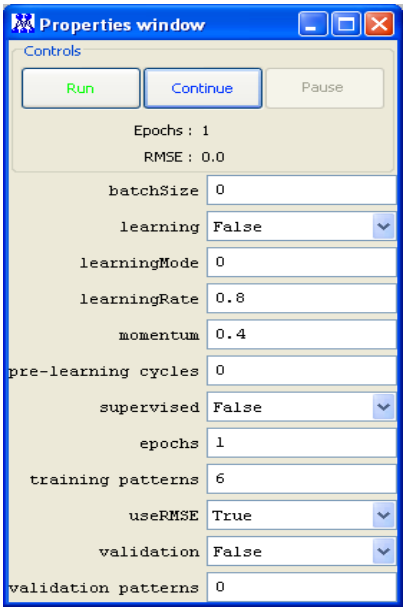


Zwróć uwagę, że ***supervised*** jest teraz ustawione na ***false.*** Również błąd RMSE nie występuje w uczeniu nienadzorowanym. Nie ma również nauczyciela.

Po zakończeniu nauki dodaj ***FileOutput*** i ustaw odpowiednią nazwę pliku wyjściowego.



W ***ControlPanel*** ustaw ***learning*** na ***false***, ***epochs*** na ***1*** :



Przykładowa zawartość pliku wynikowego może wyglądać tak:

1.0;0.0;0.0

1.0;0.0;0.0

0.0;0.0;1.0

0.0;0.0;1.0

0.0;1.0;0.0

0.0;1.0;0.0

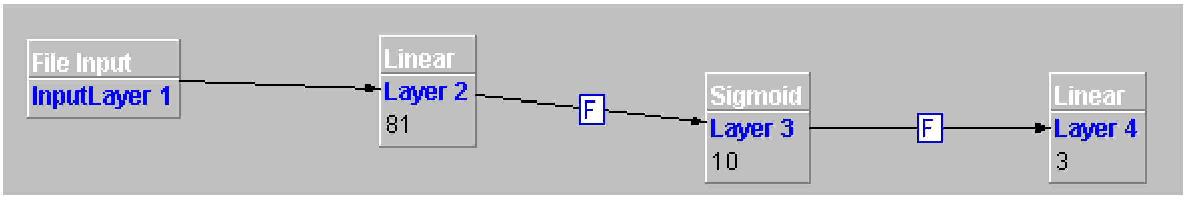
Oznacza to, że neuron pierwszy odpalił (zwyciężył) dla pierwszych dwóch danych (dwa znaki ***A***), drugi neuron odpalił dla piątego i szóstego znaku (dwa znaki ***C***) a ostatni neuron odpalił dla trzeciego i czwartego znaku (dwa znaki **B**). Zatem każdy neuron nauczył się reprezentować jedną klasę.

Nie zmieniając ustawień ***ControlPanel***, podepnij pod ***FileInput*** plik z danymi testowymi. ***FileOutput*** ustaw na nowy plik i wygeneruj odpowiedź sieci na nowe dane. Sprawdź, czy poprawnie klasyfikuje dane, tzn. czy odpowiedni neuron odpala dla każdego znaku.

Klasyfikacja znaków za pomocą sieci MLP (Multi Layer Perceptron)

Aby rozwiązać zadanie klasyfikacji znaków wykorzystamy teraz sieć MLP, podobną jak w przypadku problemu XOR. Teraz sieć nasza będzie miała warstwę wejsciową liniową doprowadzającą sygnały do sieci, warstwę ukrytą z pewną liczbą neuronów ukrytych, oraz warstwe wyjściową z liczbą neuronów równą liczbie klas, a więc w tym przypadku 3.

Skonstruuj następującą sieć:



Każdy neuron w warstwie wyjściowej powinien odpowiadać 1.0 jeśli na wejściu sieci pojawi się znak należący do klasy reprezentowanej przez dany neuron, natomiast 0.0 w przeciwnym wypadku. Musimy zatem przystosować nasze dane uczące. Plik z danymi wygląda teraz tak:

0.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;1.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;1.0;0.
0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0

0.0;0.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;1.0;1.
0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;
1.0;1.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0

0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.
0;1.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;1.0;
0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;1.0;0.0

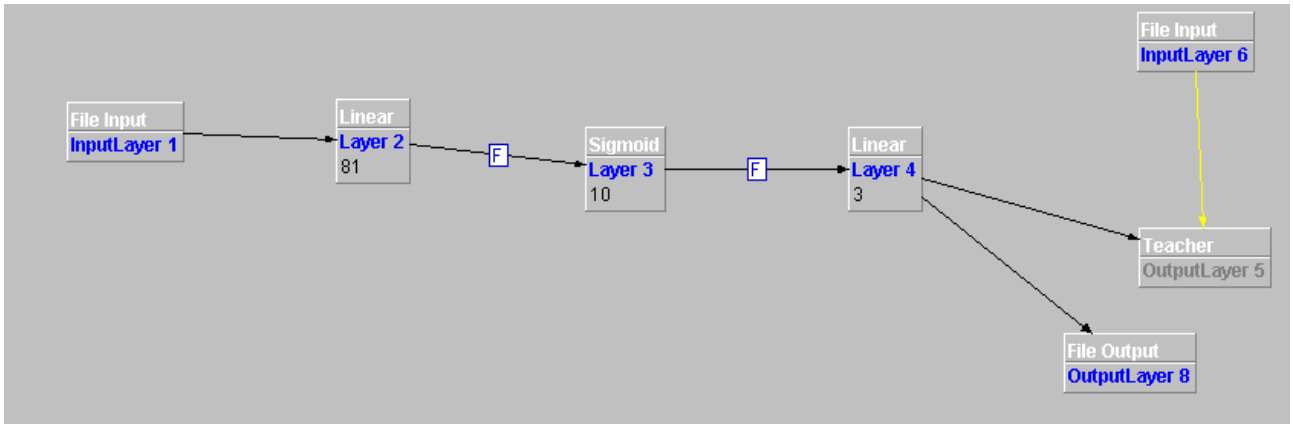
1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;1.0;1.0;0.0;0.0;0.0;1.0;0.0;0.0;0.
0;1.0;1.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;
1.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;1.0;0.0

0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.
0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;
0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0

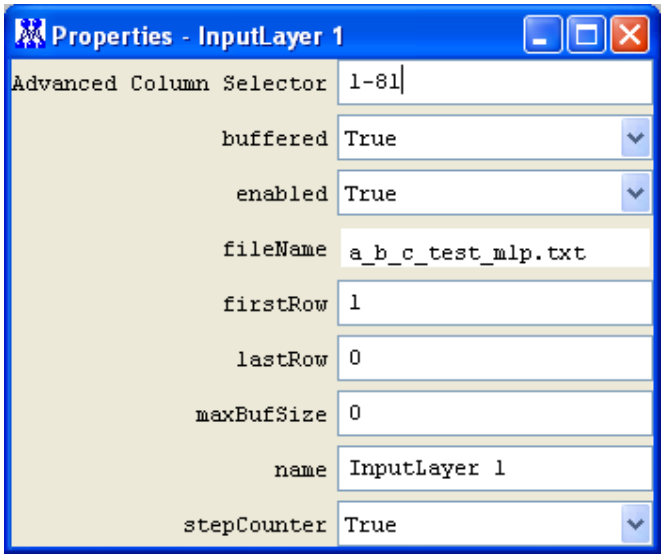
0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.
0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;1.0;0.0;0.0;1.0

Zatem nie ma juz w ostatniej kolumnie etykiety klasy, dodane natomiast zostały w trzech ostatnich kolumnach odpowiedzi wzorcowe dla trzech neuronow warstwy wyjściowej. Przenikliwości Czytelnika pozostawiam ustalenie, który neuron reprezentuje którą klasę.

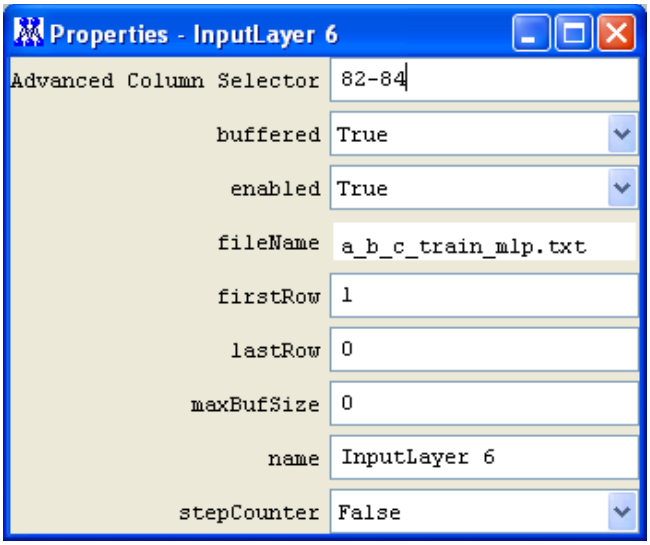
Tak przygotowane dane można podpiąć na wejśce sieci oraz do nauczyciela. Nauczyciel korzysta z trzech ostatnich kolumn, zatem:



oraz



oraz



Naucz sieć podobnie jak w przypadku problemu XOR. Następnie ustaw ***enabled*** nauczyciela na ***false*** i zmieniając ustawienia ***ControlPanel*** przekieruj odpowiedzi do pliku wyjściowego. Przykładowy wynik:

0.9999959600135174;5.524582845276753E-5;4.804248323111521E-6

1.0000001043947928;-7.272529515555109E-6;1.192532009834283E-6

4.47507659956381E-8;1.0000001847592561;-6.171580458486092E-8

1.249711879258708E-7;0.9999998379969469;-5.437855032236483E-8

-1.9369004378990643E-7;6.983926132336088E-6;0.9999995231219034

-1.1677807908638729E-7;-2.1244467943316164E-6;1.000000353797686

Zatem jak widać, sieć sie nauczyła (Dlaczego?) . Sprawdź jakie będą wyniki dla danych testujących.

Zadanie do wykonania:

Powtórz poprzednie zadanie dla cyfr od 0 do 9. Przygotuj dane uczące składające się z każdej cyfry narysowanej 3 razy. Dane testowe składają się z każdej cyfry narysowanej również 3 razy. Przeprowadź obliczenia dla sieci Kohonena oraz MLP. Która sieć działała lepiej na danych uczących a która na testowych? Czy pomaga zwiększenie liczby neuronów ukrytych w sieci MLP? Wyniki obliczneń i parametry algorytmów uczenia przedstaw w tabelkach. Opisz swoje wnioski.

Aproksymacja funkcji jednowymiarowej za pomocą sieci neuronowej

Sieci neuronowe uważane są za uniwersalne aproksymatory. Przyjrzymy się problemowi aproksymacji funkcji jednej zmiennej.

Problem aproksymacji polega na oszacowaniu, jakie będą wartości funkcji w nieznanych punktach rozpatrywanej dziedziny, jeśli dysponujemy pewnym zbiorem par ***(x, d)***, gdzie ***x*** to punkt dziedziny a ***d*** to wartość funkcji w ***x.*** Trzeba zatem znaleźć przybliżenie funkcji.

Aby rozwiązać ten problem za pomocą sieci neuronowej, zbiór par ***(x, d)*** bedziemy traktować jako dane uczące: ***x*** jako dane wejsciowe a ***d*** jako wzorcowe odpowiedzi. Od nauczonej sieci bedziemy oczekiwać, ze równiez w pozostałych punktach dziedziny bedzie odpowiadała zgodnie z prawdziwym przebiegiem nieznanej funkcji (nieznanej często w praktyce, gdy mamy tylko pary ***(x, d)*** np. z pomiarów).

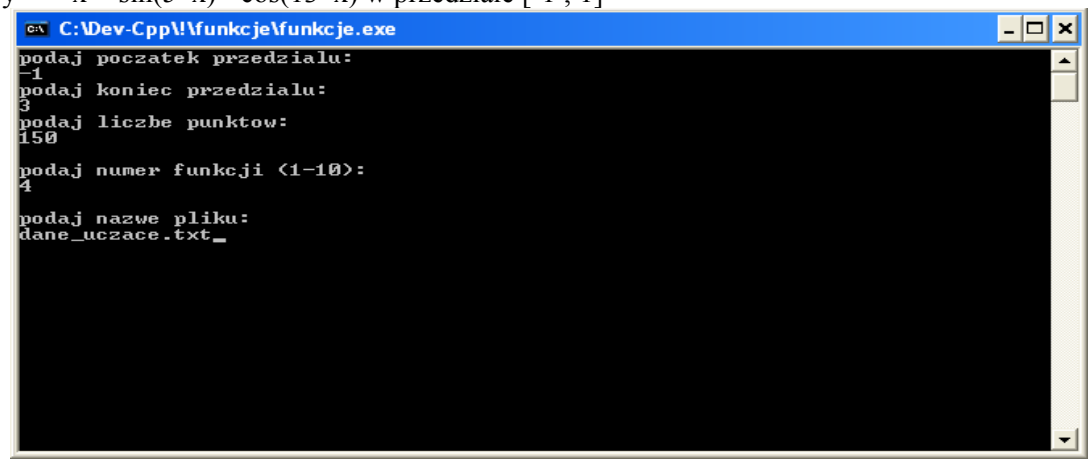
Generowanie danych uczących i testowych

Pobierz ze strony program ***funkcje.exe***. Z jego pomocą wygeneruj dane uczace i testowe (po 150 par) dla jednej z poniższych funkcji w zadanym przedziale.

!Funkcje wskazuje prowadzący!

1. $y = \sin(2*x) + \log(x^2)$ w przedziale [1 ; 15]
2. $y = \sin(2*x) + (\cos(4*x))^3$ w przedziale [-1 ; 1]
3. $y = x^2 + (\cos(4*x))^3$ w przedziale [-1.5 ; 2.5]
4. $y = \log(x^3) - \log(x^5) + (\cos(4*x))^3$ w przedziale [5 ; 12]
5. $y= x + \sin(3*\cos(5*x))$ w przedziale [0 ; 2]
6. $y= x^{1/3} + \sin(5*x)$ w przedziale [1.5 ; 5]
7. $y = 0.5*x + \sin(x^3)$ w przedziale [0 ; 4]
8. $y = x^3 – x^2 + \sin(4*x) - \cos(15*x)$ w przedziale [-1 ; 1]
9. $y = - x^2 - x + \cos(10*x) - \sin(50*x)$ w przedziale [-1 ; 1]

10. y = - x + sin(5*x) - cos(13*x) w przedziale [-1 ; 1]



Plik wygenerowany przez program jest w formacie akceptowanym przez ***Joone.***

Fragment pliku: (pierwsza kolumna ***x***, druga ***d***)

1.00922;0.656974

1.00928;0.657145

1.0123;0.665456

1.0137;0.669228

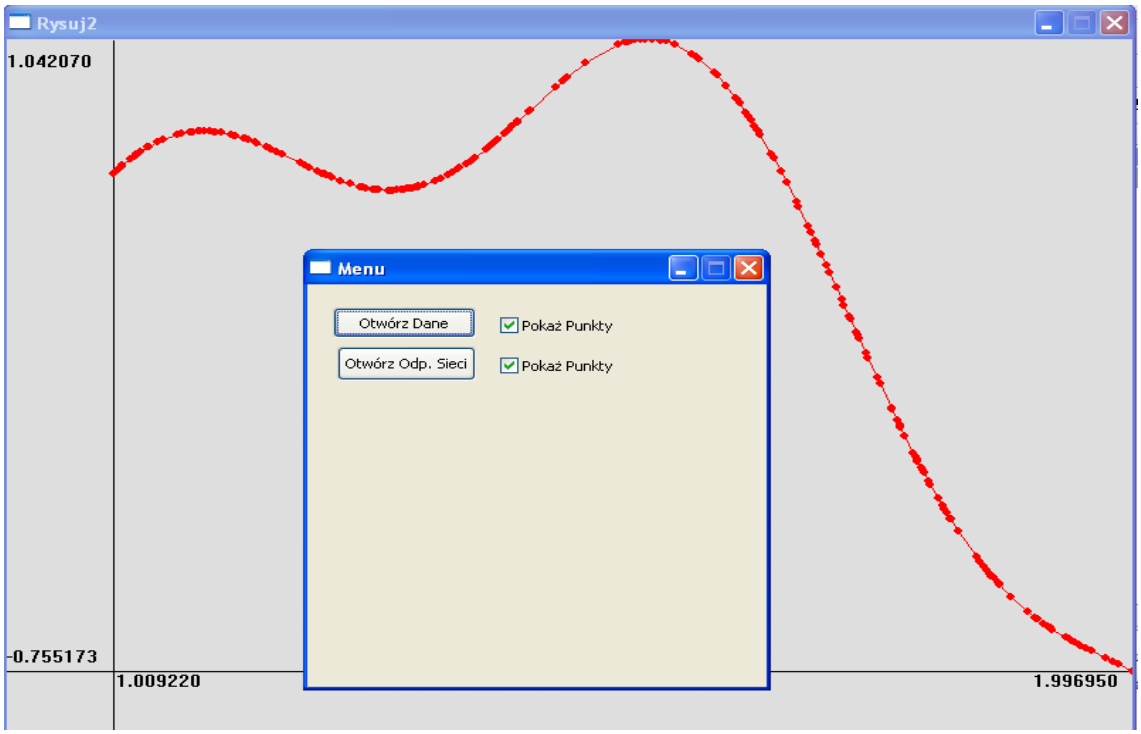
1.01773;0.679728

1.01874;0.682277

itd...

Pobierz ze strony program ***Rysownik.exe*** i z jego pomocą zobacz, jak wygląda przebieg funkcji.

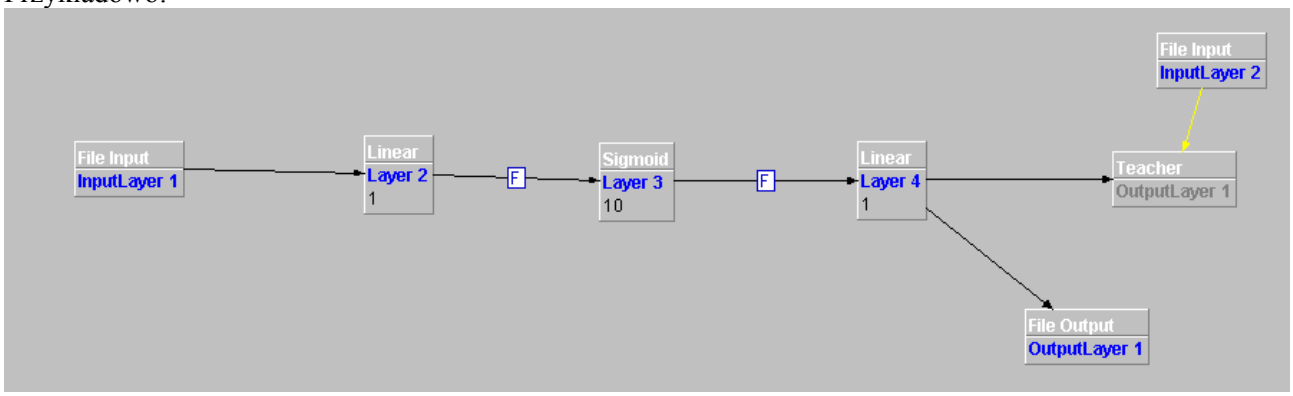
Przykładowo:

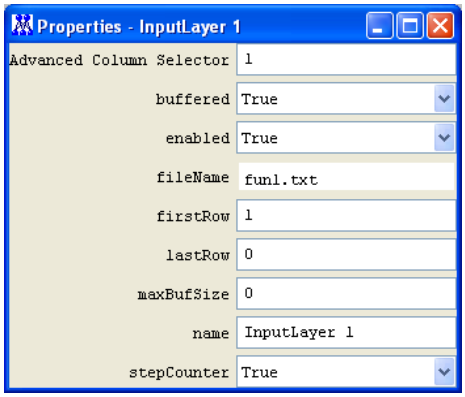


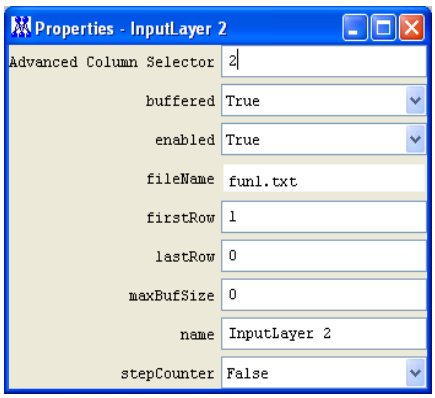
Konstrukcja sieci oraz jej uczenie

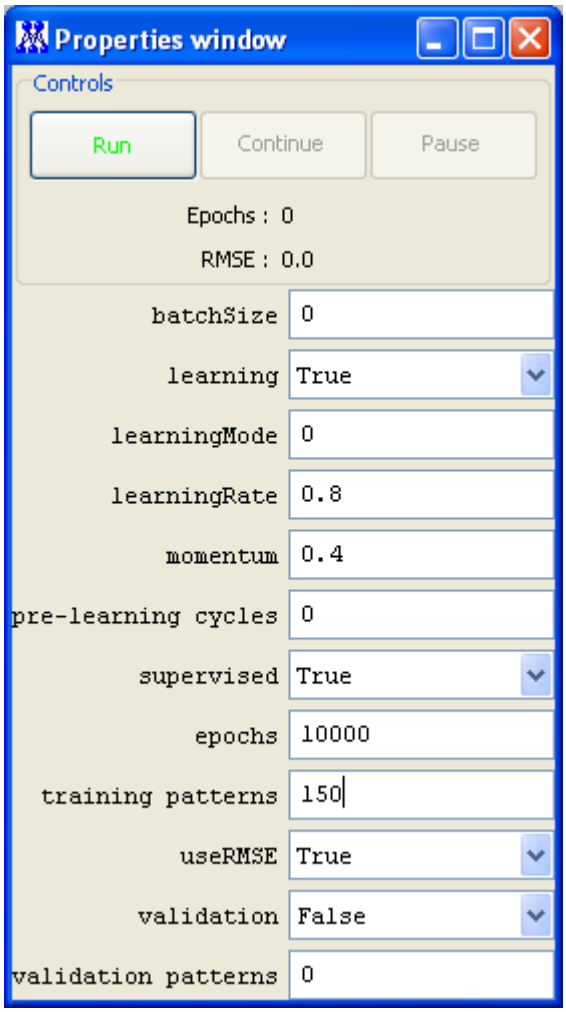
Stwórz w środowisku ***JOONE*** sieć z sigmoidalnymi neuronami w warstwie ukrytej, i jednym neuronem w wartwie wyjściowej. Dadaj nauczyciela, który bedzie korzystał z drugiej kolumny zawartej w pliku. Pierwsza kolumna to dane wejściowe sieci, zatem warstwa wejściowa sieci powinna mieć jeden neuron liniowy. Odpowienio połącz elementy i ustaw parametry uczenia bazując na swoim doświadczeniu.

Przykładowo:



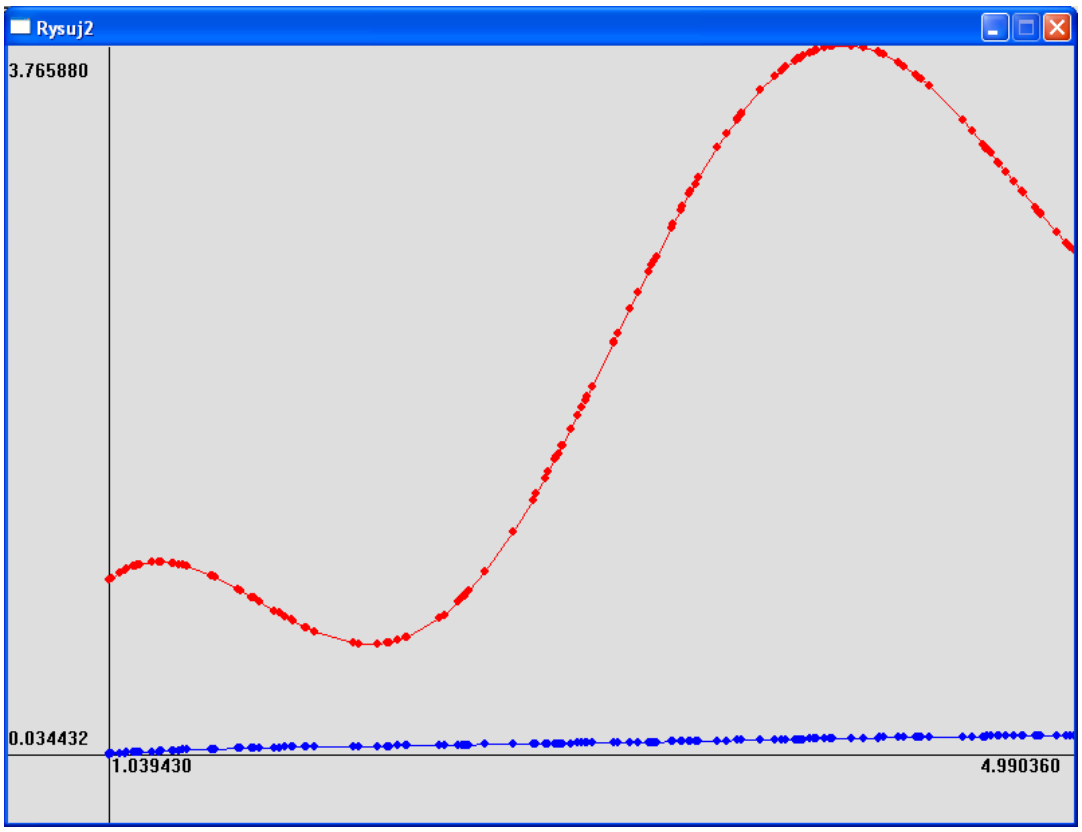




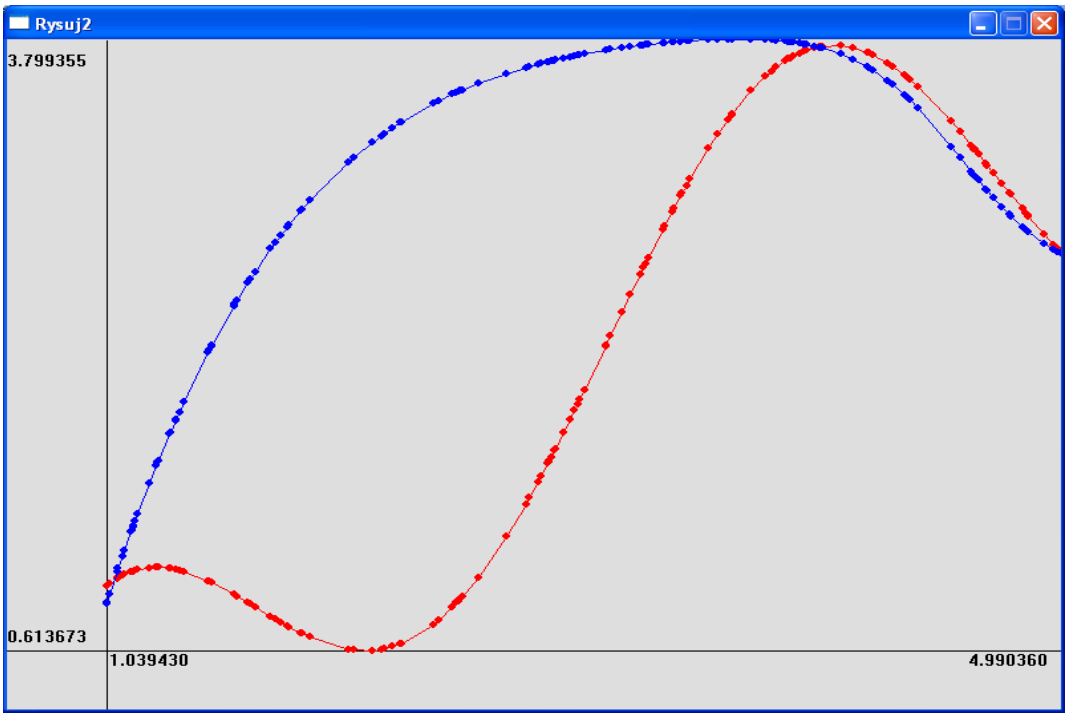


Jeśli przekierujemy odpowiedź sieci do pliku, otrzymamy w nim jedną kolumnę. Plik tak można wykorzystać w programie ***Rysownik.exe***.

Przykładowo: Odpowiedź sieci (na niebiesko) przed nauką:

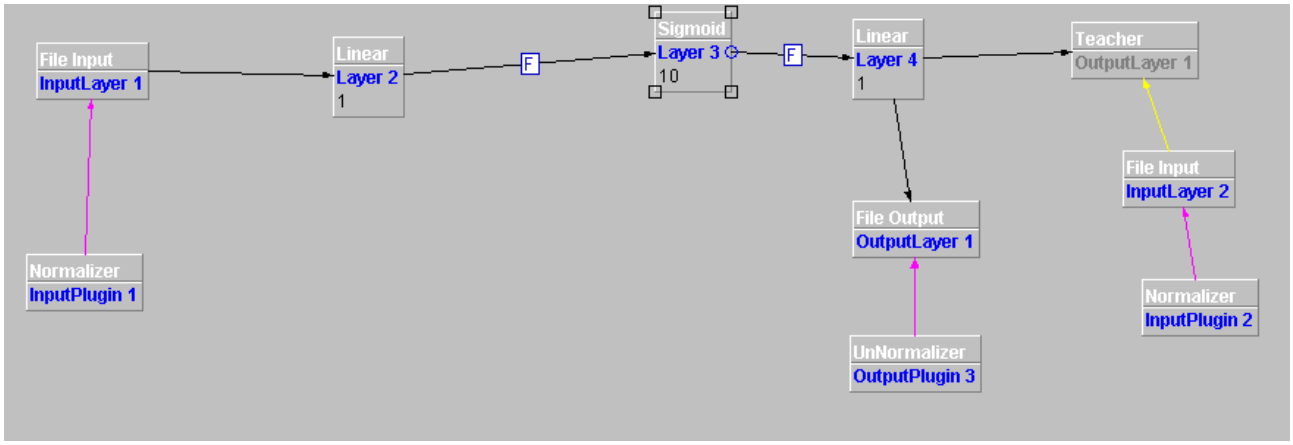


Po nauce przykładowo otrzymamy:

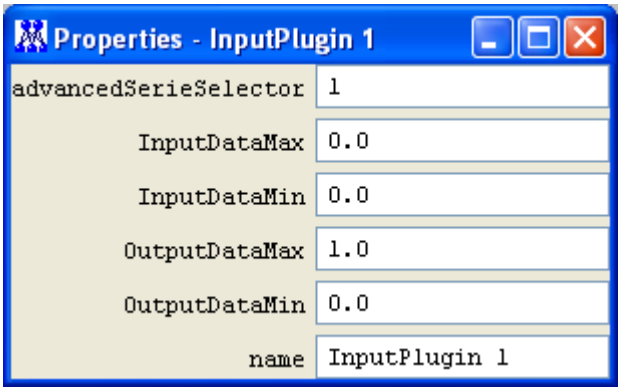


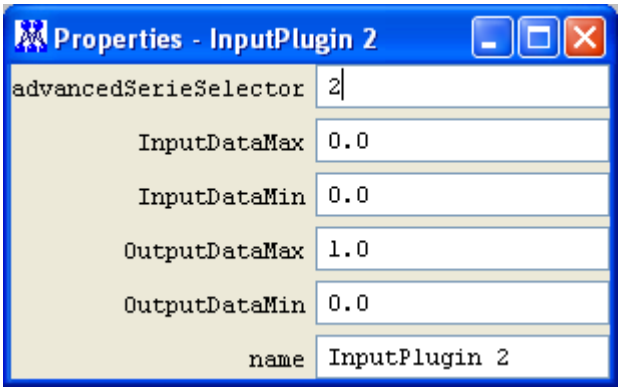
Jak widać rezultat jest daleki od idealnego. Tak może się zdarzyć, ale nie musi. Zależy to od danych uczących. Jeśli sygnały wejściowe mają duże wartości (np. w tym przypadku maksymalna wartość to prawie 5 – patrz: oś x) to sieć będzie miała problemy z nauczeniem się, gdyż występuje zjawisko tzw. nasycenia neuronów. Aby tego uniknąć, trzeba znormalizować dane, najlepiej wejściowy i wyjściowe, przykładowo do przedziału [0,1]. Możemy to zrobić już w środowisku ***JONNE***. W tym celu dodajemy ***NewNormalizer*** do ***FileInput*** przy warstwie wejściowej oraz nauczyielu. W ustawieniach zostawiamy domyślną normalizację

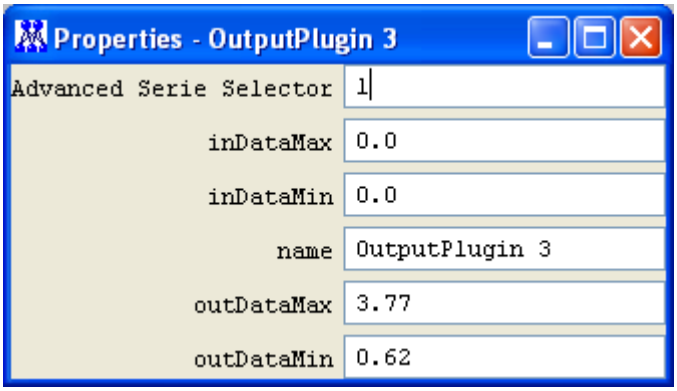
na przedział [0,1]. Aby wrócić do orginalnego przedziału wartości funkcji, użyjemy ***NewUnNormalizer*** , który podepniemy do ***FileOutput*** , do którego przekierowujemy odpowiedzi sieci. Schemat powinien wyglądać tak:



a ustawienia:

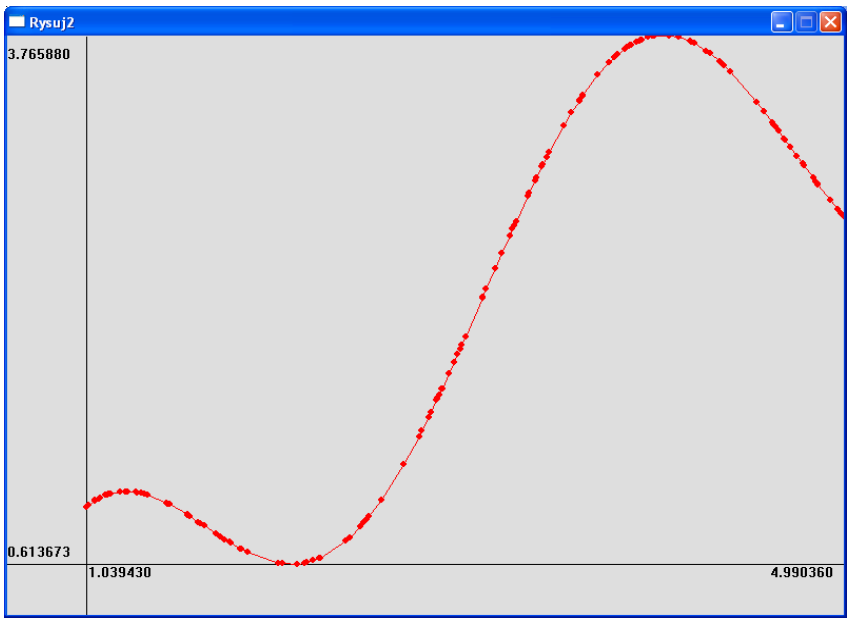




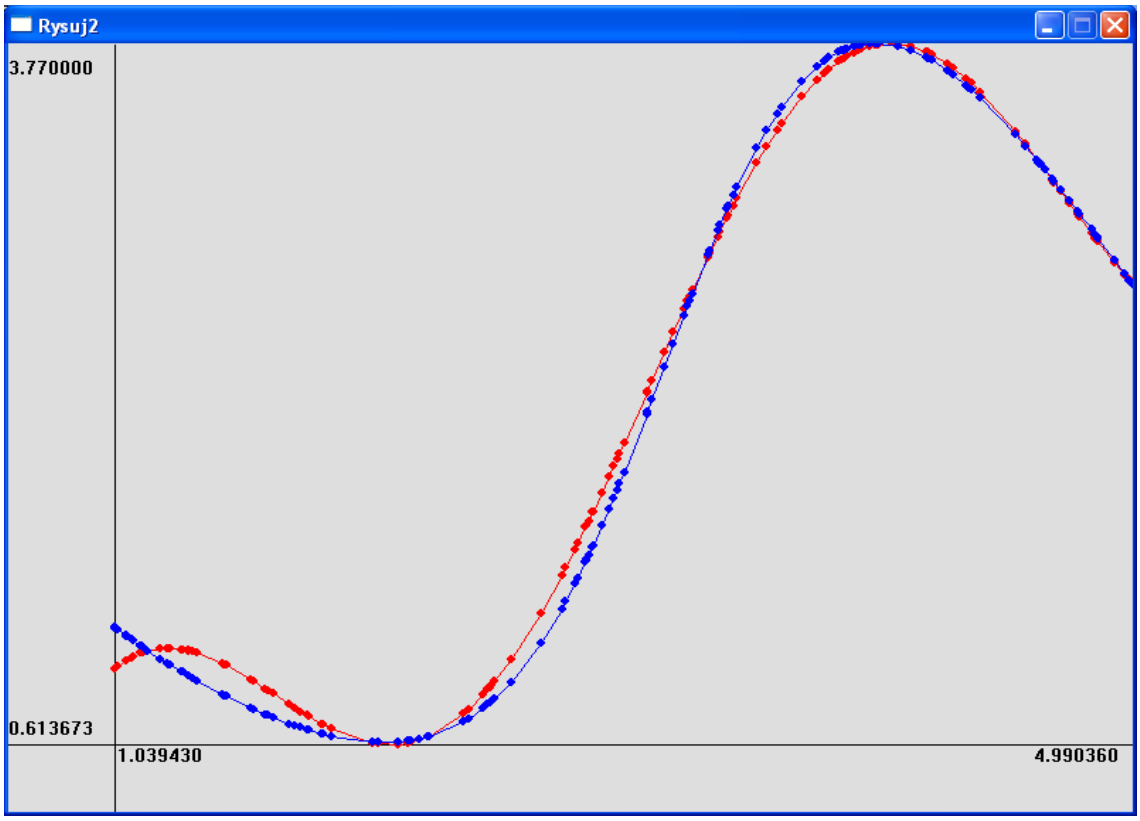


Pozostawienie 0 (zer) dla ***InputDataMin***, ***InputDataMax***, ***inDataMin*** oraz ***inDataMax*** oznacza, że program sam znajdzie min oraz max wejściowych danych.

W przypadku ***DataUnNormalizer*** (ostatnie okienko) parametry ***outDataMax*** oraz ***outDataMin*** należy odczytać korzystając z wykresu w programie ***Rysownik*** sprawdzając, jaka jest minimalna oraz maksymalna wartość danej funkcji w zadanym przedziale. W powyższym przypadku mamy min około 0.62 a max około 3.77. (patrz poniżej)



Tak wzbogaconą sieć uczymy (pamiętaj o zresetowaniu wag : ***Tools->Randomize*** !), następnie odłączamy nauczyciela (***enabled*** na ***false***), ustawiamy w ***ControlPanel learning*** na ***false***, ***epochs*** na ***1*** i przekierowujemy wyjście sieci do pliku. Korzystając z powstałego pliku z odpowiedziami sieci, widzimy, że teraz sieć, dzięki normalizacji, nauczyła się dobrze.

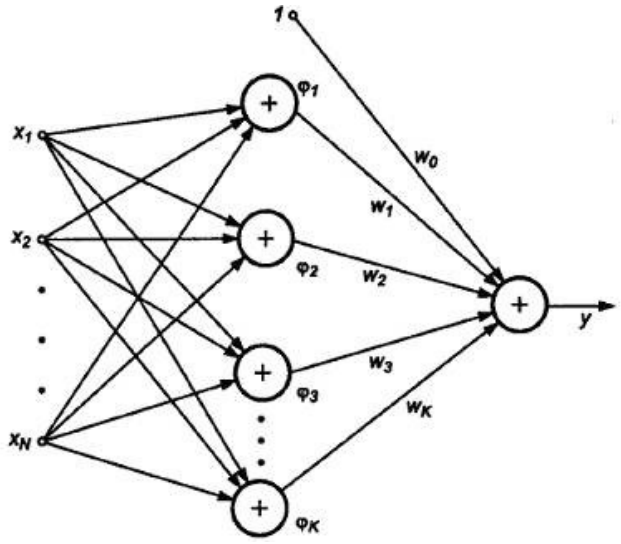


Zadanie : Dla wskazanej przez prowadzącego funkcji postaraj się dobrać odpowiednią architekturę sieci, tak aby błąd RMSE był najmniejszy. Dla funkcji wygeneruj dwa zestawy danych, trenujące (uczące) oraz testowe. Naucz sieć za pomocą danych uczących i przedstaw na wykresie jak prezentuje się odpowiedź sieci. Dane testowe nie są wykorzystywane w trakcie nauki, a jedynie do sprawdzenia, jak sieć sobie radzi z danymi, które nie były jej prezentowane podczas nauki. Przedstaw odpowiedzi sieci również dla danych testowych. W tym celu, po zakończeniu nauki sieci, pod odpowiedni ***InputLayer*** podepnij plik z danymi wejściowymi testowymi i przekieruj odpowiedź sieci do nowego pliku, a następnie użyj programu

Rysownik. Wyniki obliczeń i parametry algorytmu przedstaw w tabelkach. Wypróbuj różne ustawienia sieci, również z wiecej niż jedną warstwą ukrytą. Odpowiedz, jaka architektura była najlepsza dla rozważanego problemu (funkcji).

Sieci neuronowe radialne RBF (Radial Basis Function)

Sieci neuronowe radialne składają się jedynie z dwóch warstw jak przedstawiono to na rysunku 7.

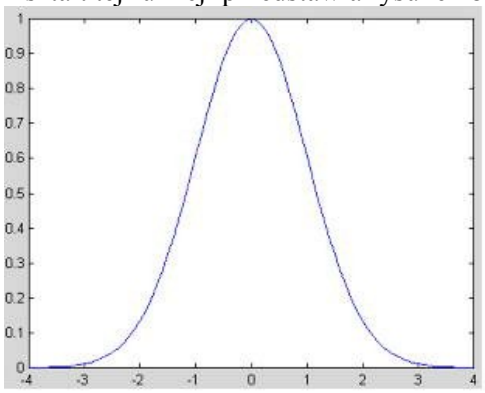


Rysunek 7 Sieć radialna

Warstwa ukryta, do której trafiają bezpośrednio sygnały wejściowe jest tworzona z neuronów posiadających radialne funkcje aktywacji, w przeciwieństwie do poprzednio poznanych sigmoidalnych funkcji aktywacji sieci MLP. Najczęściej stosowaną funkcją radialną jest funkcja Gaussa określona wzorem:

$$\varphi(x) = \exp\left(-\frac{\| x - c_i \|^2}{2\sigma_i^2}\right)$$

Kształt tej funkcji przedstawia rysunek 8.



Rysunek 8 Funkcja Gaussa: c=0, sigma=1

Zakres argumentów dla których funkcja tego typu ma wartości wyraźnie większe od zera jest ograniczony i określony przez dwa parametry: centrum radialne c_i oraz wariancję σ_i odpowiedzialną za jej szerokość. Cała sieć realizuje zatem odwzorowanie

$$F(x) = \sum_{i=1}^{K} w_i \varphi\left(\| x - c_i \|\right)$$

W sieciach sigmoidalnych separacja danych odbywała się za pomocą hiperpłaszczyzn, w sieciach radialnych natomiast ma miejsce separacja kołowa za pomocą hipersfer. Warstwa radialna realizuje rzutowanie nieliniowe na przestrzeń o większej ilości wymiarów. Zgodnie z twierdzeniem Covera, złożony problem klasyfikacyjny zrzutowany nieliniowo w przestrzeń wielowymiarową może z większym prawdopodobieństwem liniowo separowalny, niż przy rzutowaniu w przestrzeń o mniejszym wymiarze. Udowodniono, ze każdy zbiór wzorców jest linowo separowalny przy odpowiednio dużym wymiarze K.

Uczenie sieci RBF

Uczenie sieci RBF w najprostszym przypadku polega na minimalizacji funkcji celu

$$E = \sum_{i=1}^{p} \left[\sum_{j=1}^{K} w_j \varphi\left(\| x_i - c_j \|\right) - d_i \right]^2$$

gdzie `K` to liczba neuronów ukrytych a `p` to liczba par uczących. Przy założeniu znajomości parametrów funkcji radialnych, aby określić wartości wag należy rozwiązać układ równań linowych względem wag.

$$Gw = d$$

gdzie G jest macierzą radialną, zwaną również macierzą Gaussa.

$$\begin{bmatrix} \varphi\left(\| x_1 - c_1 \|\right) & .. & \varphi\left(\| x_1 - c_K \|\right) \\ .. & .. & .. \\ \varphi\left(\| x_p - c_1 \|\right) & .. & \varphi\left(\| x_p - c_K \|\right) \end{bmatrix}$$

Macierz G jest przeważnie macierzą prostokątną, wektor wag oblicza się zatem za pomocą operacji pseudo inwersji macierzy G:

$$w = G^{+} d = (G^T G)^{-1} G^T d$$

Centra funkcji radialnych ustalić można np.:

- losowo, losując centra w przestrzeni danych stosując rozkład normalny
- statystycznie, używając metody uśrednień (k-means)w celu identyfikacji klastrów
- z użyciem samoorganizacji
- z użyciem algorytmu genetycznego itd.

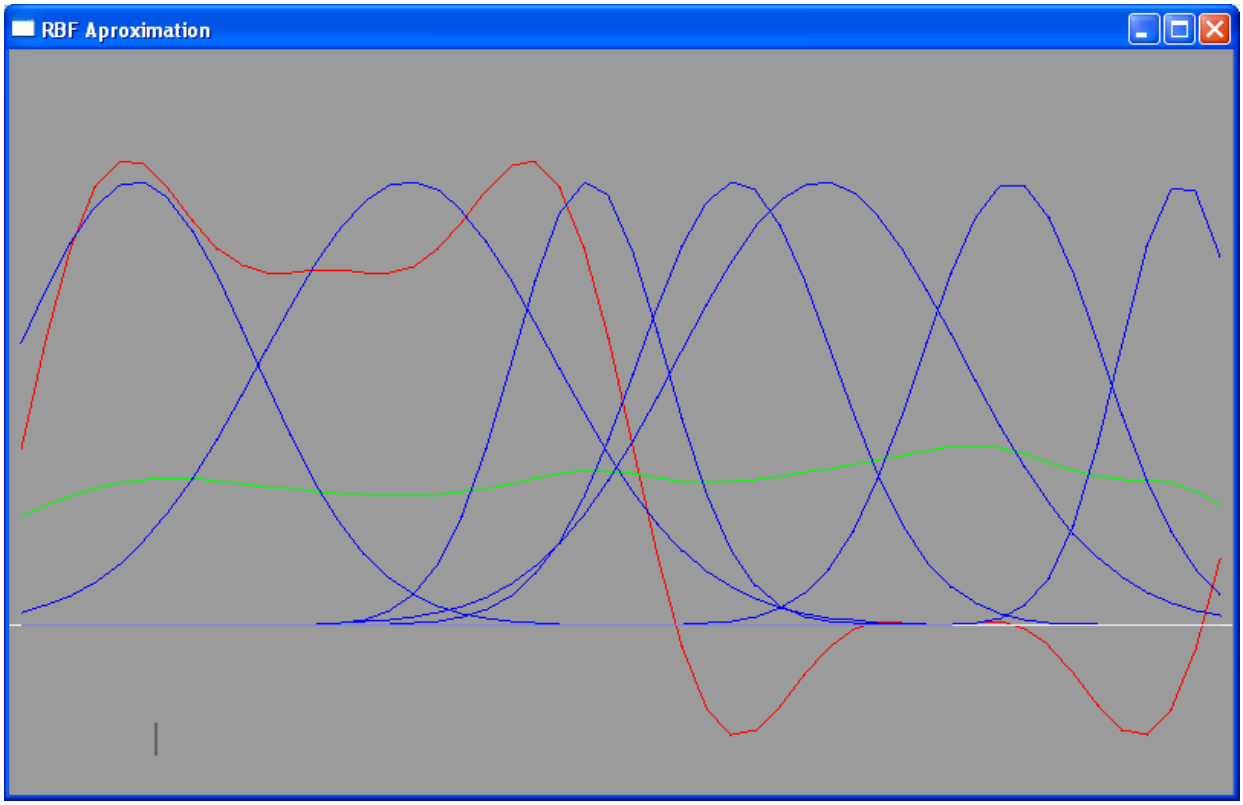
Szerokości funkcji radialnych dobiera się tak, aby pokryta została cała przestrzeń.
Inną metodą jest zastosowanie zasady wstecznej propagacji błędów zarówno do wartości wag jak i parametrów funkcji radialnych.

Aproksymacja funkcji jednej zmiennej za pomocą sieci RBF

Pobierz ze strony program demonstracyjny ***RBF_Demo***. Zaobserwuj, jak sieć neuronowa radialna uczy się aproksymować jednowymiarową funkcję. Sieć ta ma 7 neuronów radialnych ukrytych (kolor niebieski w programie) oraz jeden neuron liniowy w warstwie wyjściowej, dający konkretną wartość aproksymacji w danym punkcie dziedziny. Przebieg funkcji przedstawiono kolorem czerwonym, aproksymację dokonywaną

przez sieć natomiast kolorem zielonym. Zaobserwuj, jak w trakcie nauki są modyfikowane funkcje Gaussa neuronów radialnych. Naukę zaczyna się klikając na okienko programu prawym przyciskiem myszy.

Przed nauką:



Po nauce: