# Obliczenia neuro-rozmyte. Projekt zaliczeniowy

Dariusz Kuziemski

31 maja 2011

## 0.1 Wprowadzenie

Rozpatrywanym w tym sprawozdaniu problemem jest porównanie efektywności dwóch różnych metod aproksymacji funkcji - pierwszą jest algorytm generujący proste reguły rozmyte, a drugą jest propagacja wsteczna błędu na jednokierunkowej trójwarstwowej sieci neuronowej.

Funkcje, z którymi będą musiały zmierzyć się powyższe algorytmy to:

1. $f_1(x) = sin(x) + \varepsilon, \qquad x \in [0, 2\pi]$

   (będąca prostym wielomianem stopnia 3-ciego na tym przedziale, $\varepsilon$ jest szumem z przedziału $[-0.05, 0.05]$ w pierwszym przypadku oraz $[-0.1, 0.1]$ w drugim)

2. $f_2(x) = exp[-2log(2)(\frac{x-0.08}{0.854})^2]sin^6(5\pi(x^{3/4} - 0.05)), \qquad x \in [0, 1]$

   (harmoniczny, malejący wielomian stopnia 11-tego na tym przedziale)

3. $f_3(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7), \qquad x, y \in [-5, 5]$

   (funkcja 2 zmiennych w kształcie "siodła")

Dla pierwszej i drugiej funkcji wybieram losowo 100 punktów, dla trzeciej kolejno 1000 i 10000 punktów.

Problem aproksymacji funkcji jest problemem dobrze znanym i rozpatrywanym przy użyciu rozmaitych metod takich jak krzywe Beziera czy aproksymacja liniowa. Zagadnieniem tym przy użyciu prostych reguł rozmytych zajmował się (i jednocześnie zaproponował metodę) Nozaki w pracy [1], natomiast o użyciu jednokierunkowych sieci wielowarstwowych (w kontekście sztucznych sieci neuronowych jako uniwersalnych aproksymatorów funkcji) możemy poczytać w artykule [2].

## 0.2 Metodologia

### 0.2.1 Proste reguły rozmyte

Systemy oparte na regułach rozmytych są sukcesywnie wykorzystywane w przypadkach gdy natura badanego procesu jest nieprecyzyjna, zbyt złożona dla konwencjonalnych technik lub posiadanych informacji jest zbyt mało. W wielu aplikacjach rozmyte reguły $if-then$ były uzyskiwane z eksperckiej wiedzy ludzkiej, jak i uzyskiwane różnorodnymi automatycznymi metodami angażującymi iteracyjne procedury nauczania lub skomplikowanymi mechanizmami generującymi je, takimi jak metody gradientowe, algorytmy genetyczne, metoda najmniejszych kwadratów, metoda k-średnich i metody neuro-rozmyte.

Proponowana przez Nozakiego[1] heurystyczna metoda generowania uproszczonych reguł pomija etap wyostrzania, ustalając jako regułę $if-then$ liczbę rzeczywistą określaną jako średnią ważoną analizowanych danych numerycznych nie angażując przy tym czasochłonnego iteracyjnego procesu nauczania ani skomplikowanych mechanizmów tworzących te reguły.

Chcąc wykonać aproksymację funkcji przy użyciu uproszczonych reguł rozmytych musimy wykonać 2 zasadnicze kroki: identyfikację (dla każdej próbki danych $(\bar{x}_j, u_j)$) oraz wnioskowanie. Parametrami o których decyduje użytkownik to ilość podziałów dziedziny na przedziały (w tym przypadku trójkątne) oraz wartość stałej $\alpha$, będącego stopniem dopasowania.

#### Identyfikacja

Obliczamy wagi (stopnie zgodności obserwacji $\bar{x}$ z rozmytym obszarem):

$w_{k_1 \ldots k_n}(\bar{x}) = (\prod_{j=1}^{n} \mu_{k_j}(\bar{x_j}))^{\alpha}$

gdzie $\mu$ jest funkcją zwracającą stopień przynależności do trójkątnego przedziału. Następnie wyliczamy numeryczne konkluzje:

$$b_{k_1 \ldots k_n} = \frac{\sum_{j=1}^{m} w_{k_1 \ldots k_n}(\bar{x}) * u_j}{\sum_{j=1}^{m} w_{k_1 \ldots k_n}(\bar{x})}$$

#### Wnioskowanie

Na podstawie wyznaczonych konkluzji $b_{k_1 \ldots k_n}$ możemy wyznaczyć wartości $u(\bar{x})$ dla wektora $\bar{x}$ ze wzoru:

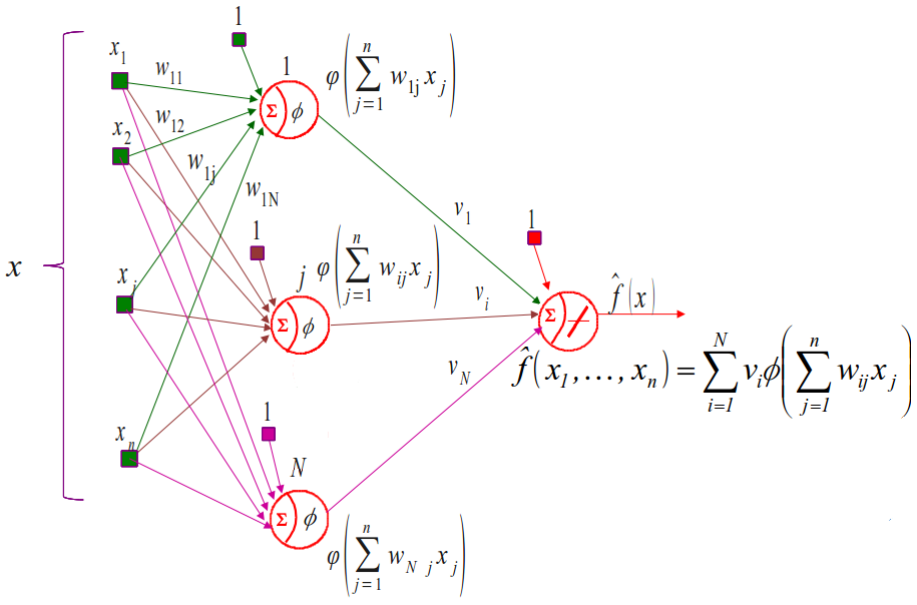
$$u(\bar{x}) = \frac{\sum_{k=1}^{N} \prod_{j=1}^{n} \mu_{k_j}(\bar{x_j}) b^(k)}{\sum_{k=1}^{N} \prod_{j=1}^{n} \mu_{k_j}(\bar{x_j})}$$

### 0.2.2 Propagacja wsteczna błędu

Procesem kluczowym w przetwarzaniu sieci neuronowych jako narzędzia analitycznego jest nauczanie sieci. Propagacja wsteczna, algorytm nauczania nadzorowanego, jest niewątpliwe kamieniem milowym w tej dziedzinie - pierwotnie została zaprezentowana przez Brysona i Ho w [5], niezależnie odkryta także przez Werbosa w 1974 [6]. Proces nauczania przy użyciu tej metody można opisać w skrócie w następujących krokach:

1. Losowe przypisanie wartości wagom (zazwyczaj wartościami z przedziału -1.0 do 1.0).
2. Dla każdej próbki treningowej (lub zestawu próbek) wyliczane są wyjścia dla aktualnych wag w sieci.
3. Wyliczany jest błąd na wyjściu, następnie w warstwach $M, M-1, ...2$, na ich podstawie oraz współczynnika nauczania $\eta$ aktualizowane są wagi.
4. powtarzamy od kroku drugiego dopóki warunek końcowy nie zostanie spełniony (np.: maksymalna ilość epok, osiągnięty wskazany poziom błędu).

Do zadania aproksymacji funkcji skorzystałem z modelu sieci z wyjściem liniowym przedstawionego w prezentacji [7] jako "Sieć neuronowa Funahashiego" (rysunek 1). Za funkcję aktywacji $\varphi$ we warstwie ukrytej posłużył mi sigmoid unipolarny. Parametry (ilość jednostek ukrytych oraz współczynnik uczenia $\eta$ dobierałem wielokrotnie metodą prób i błędów.

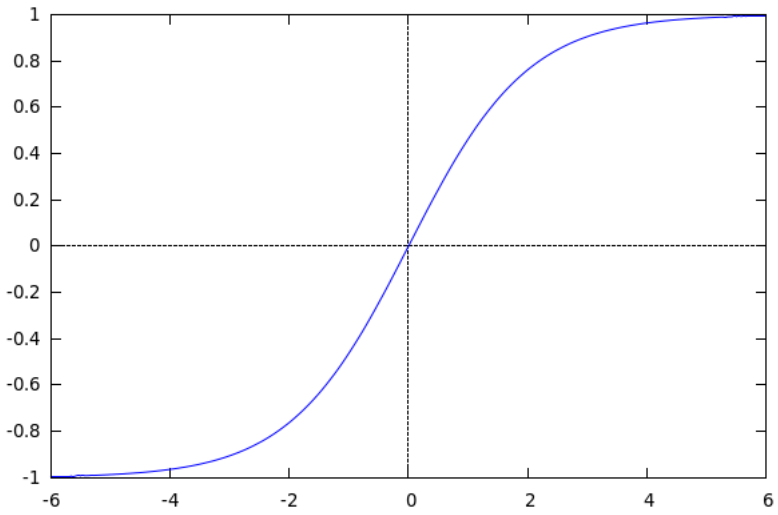



Rysunek 1: Sieć neuronowa Funahashiego

### 0.2.3 Funkcje aktywacji (dla sieci neuronowej)

Funkcją aktywacji dla warstwy ukrytej jest funkcja sigmoidalna bipolarna wyrażana wzorem:

$f(x) = \frac{2}{1+e^{-x}} - 1$




Rysunek 2: Funkcja sigmoidalna bipolarna

Funkcja aktywacji dla wyjścia jest **liniowa**.

### 0.2.4 Błąd średniokwadratowy

Oceniając modele obliczam *błąd średniokwadratowy* (ang. Mean Squares Error) ze wzoru:

$MSE = \frac{\sum_{i=0}^{n}(u_i - o_i)}{n}$

gdzie $n$ jest liczbą obserwacji w próbce, $u_i$ wartością docelową, a $o_i$ uzyskaną wartością.

## 0.3 Wyniki

### 0.3.1 Funkcja $f_1(x)$

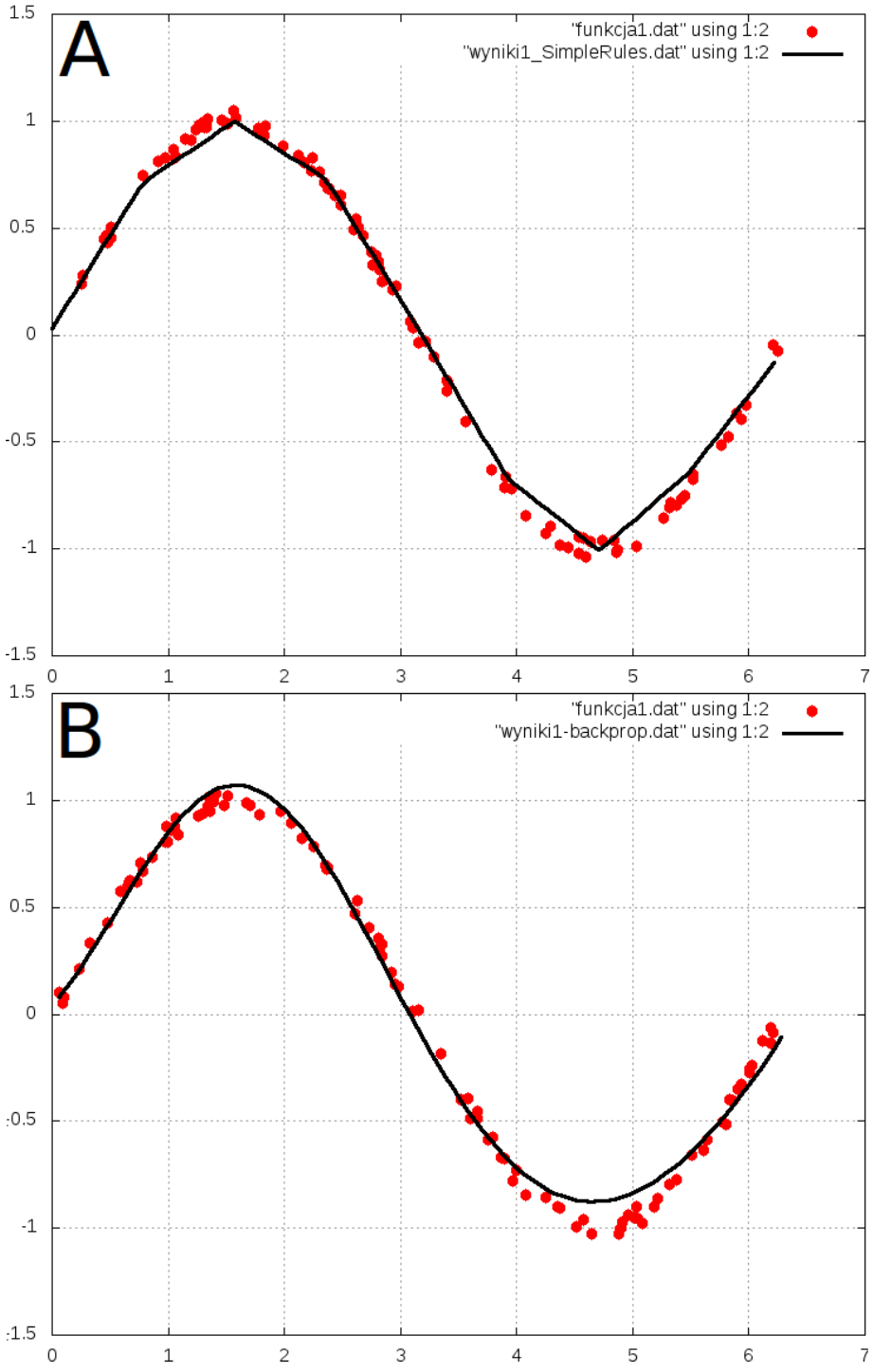
$f_1(x) = sin(x) + \varepsilon$

Konfiguracja oraz wyniki

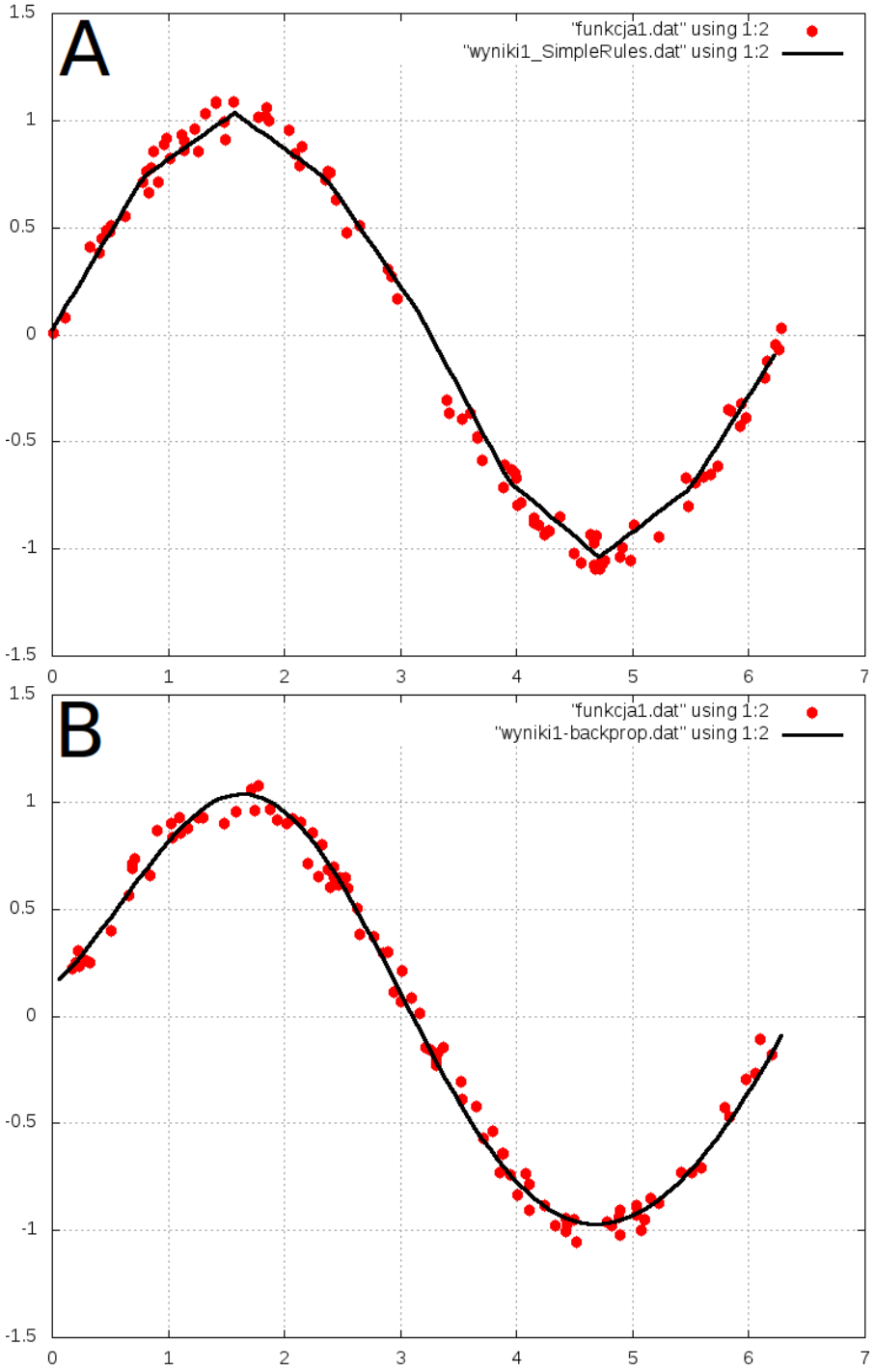
| parametr | Uproszczone reguły | Backpropagation |
|---|---|---|
| liczba przedziałów rozmytych lub jednostek ukrytych | 9 | 9 |
| stopień przystosowania $\alpha$ lub współczynnik uczenia $\eta$ | $\alpha = 10$ | $\eta = 0.00145$ |
| $\varepsilon \in [-0.05, 0.05]$ | | |
| liczba przeliczonych epok | - | 105225 |
| uzyskany błąd MSE | 0,0032 | 0,005 |
| $\varepsilon \in [-0.1, 0.1]$ | | |
| liczba przeliczonych epok | - | 1204425 |
| uzyskany błąd MSE | 0,0052 | 0,005 |

Po zwiększeniu szumu można było oczekiwać również zwiększenie błędu - tak się stało w przypadku prostych reguł rozmytych - sieć neuronowa wyrównała (*docelowy*) poziom błędu kosztem dużej ilości dodatkowo przeliczonych epok. Oczywiście błąd dla prostych reguł można także zmniejszyć zwiększając ilość obszarów rozmytych (przykładowo dla 25 obszarów, dla szumu $\varepsilon \in [-0.1, 0.1]$ otrzymałem *mse* rzędu 0.0025).

Jedną z zalet sieci neuronowych jest odporność na zaszumione dane (wymieniane między innymi w [10]) jednak ponoszony koszt w obliczeniach dla przypadku tak prostej funkcji jak $f_1(x)$ wydaje się być z mojego punktu widzenia (po zapoznaniu się z uproszczonymi regułami rozmytymi) bardzo duży.




Rysunek 3: $f_1(x)$ dla $\varepsilon \in [-0.05, 0.05]$, A: proste reguły rozmyte, B: propagacja wsteczna




Rysunek 4: $f_1(x)$ dla $\varepsilon \in [-0.1, 0.1]$, A: proste reguły rozmyte, B: propagacja wsteczna

### 0.3.2 Funkcja $f_2(x)$

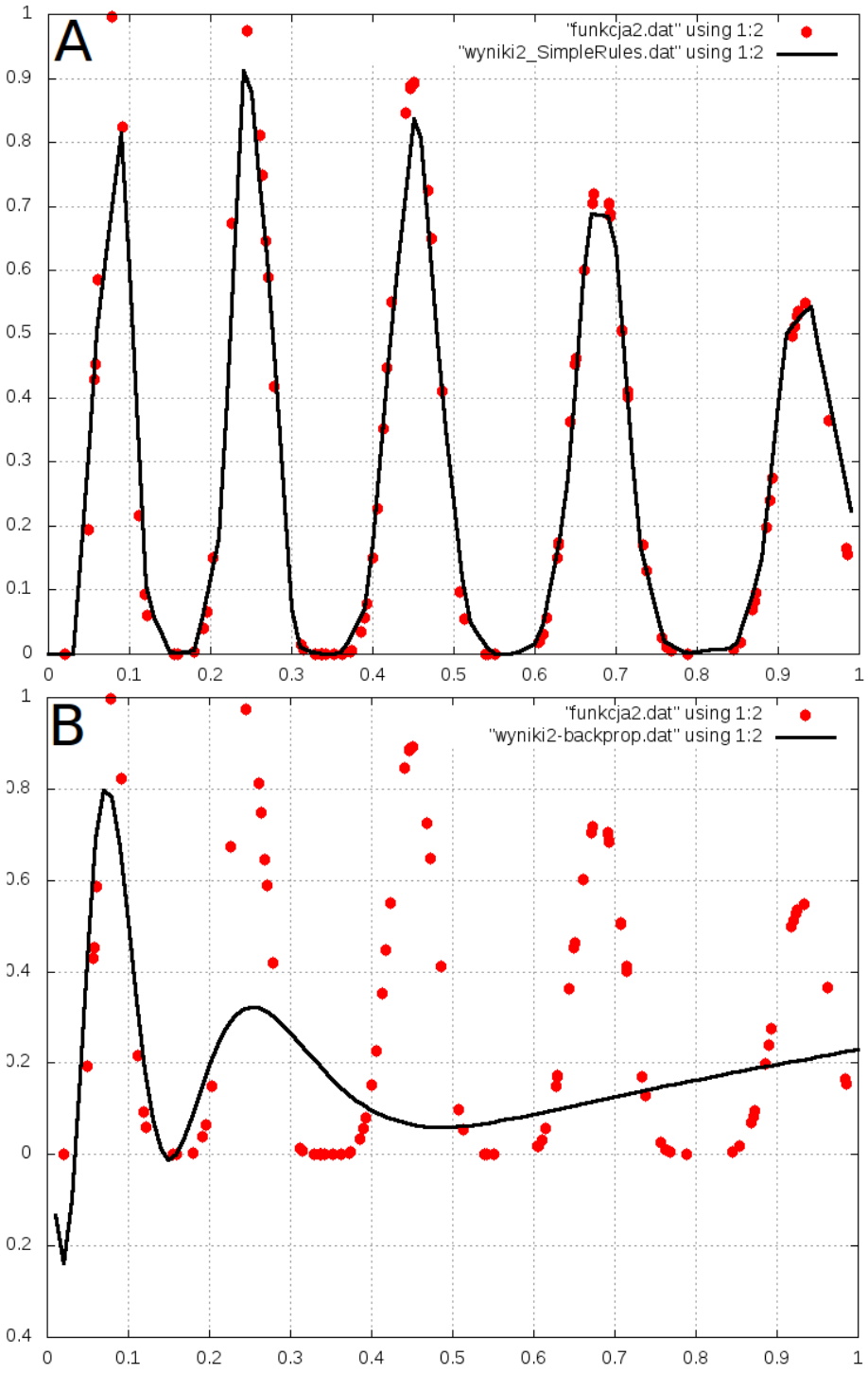
$f_2(x) = exp[-2log(2)(\frac{x-0.08}{0.854})^2]sin^6(5\pi(x^{3/4} - 0.05))$

Konfiguracja oraz wyniki

| parametr | Uproszczone reguły | Backpropagation |
|---|---|---|
| liczba przedziałów rozmytych lub jednostek ukrytych | 34 | 36 |
| stopień przystosowania $\alpha$ lub współczynnik uczenia $\eta$ | $\alpha = 10$ | $\eta = 0.01$ |
| liczba przeliczonych epok | - | 33105780 |
| uzyskany błąd MSE | 0,0024 | 0,064 |

Podczas aproksymacji funkcji drugiej przy użyciu sieci neuronowej (z propagacją wsteczną jako metodą nauczania) spotkałem się z dużymi trudnościami, mimo wielu prób dobierania wag i współczynnika nauczania udawało mi się uzyskać maksymalnie wielomiany 5-6 stopnia. Uproszczone reguły rozmyte okazały się niewątpliwie dużo stabilniejszą i pewną metodą niewątpliwie ujawniając jedną ze swoich zalet - brak iteracyjnego procesu nauczania i uzyskując względnie dobre wyniki końcowe (niski błąd i wizualne przybliżenie funkcji (wykres 5)).

Najprawdopodobniej przy niewielkim współczynniku nauczania i nieograniczonym czasie także propagacja wsteczna w końcu byłaby w stanie aproksymować dobrze dany wielomian 11-tego stopnia (funkcję $f_2(x)$), jednak musimy mieć na względzie iż w realnych problemach nie pracujemy w teoretycznym modelu, którego jedną z cech jest nieograniczony czas (w przełożeniu na względną moc obliczeniową). Zastanawiałem się także nad aproksymacją tej funkcji przedziałami, mogłoby to przynieść dobre wyniki (niski błąd i dobre przybliżenie faktycznego kształtu aproksymowanej funkcji), mijało się to jednak z jedną z idei sieci neuronowych czyli tego iż mają być one także wysoce skuteczne tam gdzie natura problemu nie jest nam znana.




Rysunek 5: $f_2(x)$, A: proste reguły rozmyte, B: propagacja wsteczna

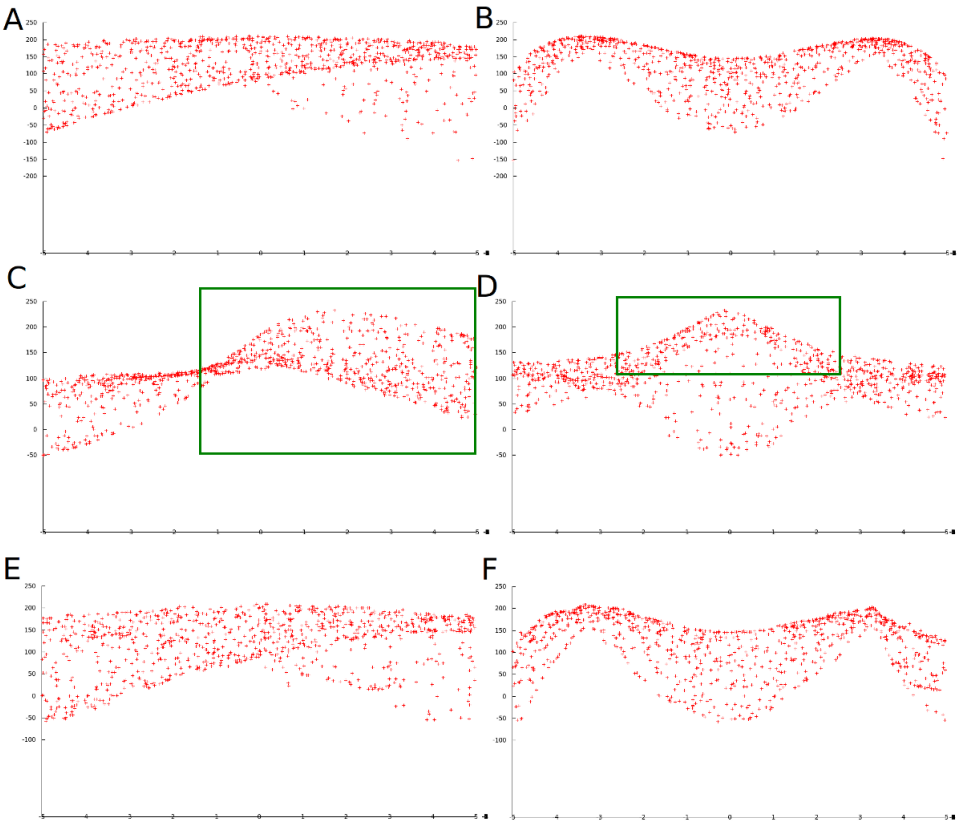
### 0.3.3 Funkcja $f_3(x, y)$

$f_3(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7), \qquad x, y \in [-5, 5]$

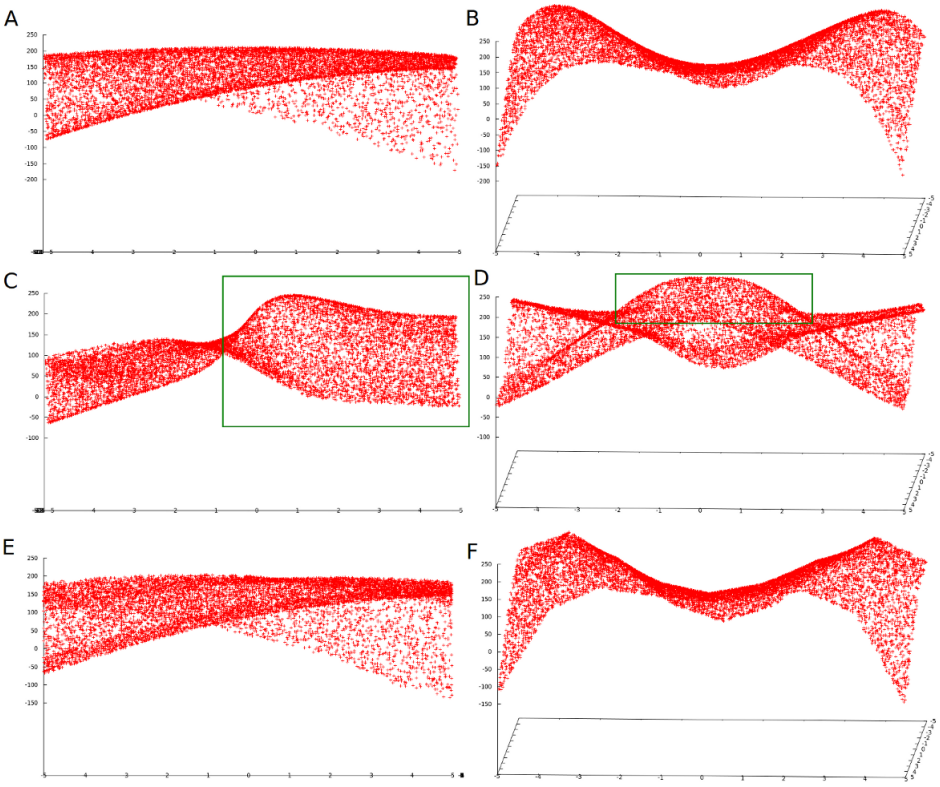
Konfiguracja oraz wyniki

| parametr | Uproszczone reguły | Backpropagation |
|---|---|---|
| iczba punktów: 1000 | | |
| liczba przedziałów rozmytych lub jednostek ukrytych | 169 | 250 |
| stopień przystosowania $\alpha$ lub współczynnik uczenia $\eta$ | $\alpha = 10$ | $\eta = 0{,}0025$ |
| liczba przeliczonych epok | - | 558404 |
| uzyskany błąd MSE | 464,1505 | 3885,0936 |
| liczba punktów: 10000 | | |
| liczba przedziałów rozmytych lub jednostek ukrytych | 169 | 350 |
| stopień przystosowania $\alpha$ lub współczynnik uczenia $\eta$ | $\alpha = 10$ | $\eta = 0{,}00025$ |
| liczba przeliczonych epok | - | 3704 |
| uzyskany błąd MSE | 348,0395 | 5332,2558976 |

Dla prostych reguł rozmytych zwiększona ilość punktów równała się zmniejszeniu błędu, wykresy funkcji wizualnie także jest bardzo zbliżony czego nie można powiedzieć o wynikach metody drugiej - propagacji wstecznej. Jak zaznaczono na wykresach 6, 7 (C oraz D), wydaje się jakby wynikiem działania była zaproksymowana tylko część wykresu, a pozostała (zaznaczona zielonym prostokątem) nie zmieniała się już pomimo wzrastającej ilości przeliczonych epok. Niestety nie wykonałem pomiarów czasowych dla porównania kosztów wzrostu ilości badanych obserwacji w porównaniu dla obydwu metod.




Rysunek 6: Porównanie dla $f_3(x, y)$ dla 1000 punktów: wykresy A i B - funkcja oryginalna, C i D - funkcja uzyskana przy użyciu propagacji, E i F - wynik dla prostych reguł rozmytych. Kolumna pierwsza to rzut od strony osi Y, druga od strony osi X.




Rysunek 7: Porównanie dla $f_3(x, y)$ dla 10000 punktów: wykresy A i B - funkcja oryginalna, C i D - funkcja uzyskana przy użyciu propagacji, E i F - wynik dla prostych reguł rozmytych. Kolumna pierwsza to rzut od strony osi Y, druga - ukośny od strony osi X.

## 0.4 Konkluzje

Propagacja wsteczna w swojej "podstawowej wersji" jest algorytmem znanym z niskiej zbieżności na co chociażby wskazywali Girosi wraz z Poggio w pracy [4]. Zostało już zaproponowane wiele zmian, przykładowo zmieniający się współczynnik nauczania w trakcie uczenia[8], zastosowanie *momentum* regulującego tempo zmian w wagach, regulując zbieżność w trakcie nauczania stochastycznego (*ang. Momentum BackPropagation*), lub zastosowanie algorytmu drugiego rzędu takiego tak QuickProp[9] łączący elementy metody newtonowskiej z wiedzą heurystyczną.

Pomimo wielu plusów uproszczonych reguł rozmytych warto zauważyć iż algorytm musi znać dziedzinę na której pracuje (by przygotować podziały przedziałów dla których wyznacza przynależności). Sieci neuronowe mają tę zaletę, że nawet nie dysponując dokładną wiedzą o naturze problemu możemy wykorzystać je do analiz (najprostszym przykładem będzie tu właśnie wspomniany brak potrzeby znajomości dziedziny funkcji).

Przypuśćmy iż analizowany problem posiada dużą rozpiętość dziedziny oraz jest on podobny naturą do problemów z lokalnymi katastrofami (lokalnie niemonotoniczny). Chcąc polepszyć jakość aproksymacji warto byłoby zastanowić się nad algorytmem rozmieszczającym podział przedziałów (ja w swoim programie dzieliłem dziedzinę generując regularne przedziały) - powierzchownie zbadać dziedzinę sprawdzając monotoniczność obszarami (metodą *dziel i zwyciężaj*) i tam gdzie zmienność jest najwyższa zagęścić podziały. Nawet dysponując przeciętną mocą obliczeniową obliczenia jakie będziemy zmuszeni w takim przypadku wykonać nie będą dużym kosztem, a końcowy wynik aproksymacji może być znacznie lepszy.

## 0.5 Krótki opis działania programu

Program został napisany w C# w zgodności z .Net w wersji 2.0, wygodnie obejrzeć (w zwykłych edytorach kod *.cs ma wysokie preferencje do rozjeżdżania przez co staje się nieczytelny) oraz skompilować można chociażby w darmowym IDE MonoDevelop *http://monodevelop.com/Download*.

Na schemacie klas (8) przedstawiony jest diagram klas w programie, wersję w wysokiej rozdzielczości załaczam jako plik *Schemat.png*. Sam kod starałem się dobrze komentować, a nazwy zmiennych dobierać celnie, zgodnie ihc intuicyjnym przeznaczeniem.

**Program.cs** - funkcja inicjująca, tutaj znajduje się mechanizm wybóru metody.

**BackpropagationTest.cs & SimplifiedFuzzyRulesTest.cs** - klasy inicjujące konfiguracje metod, ich parametrów oraz wybór aproksymowanej funkcji (w plikach **SimplifiedFuzzyRulesPreconfig.cs** i **BackpropagationTestPreconfig.cs** kolejno znajdują się metody powyższych klas). W trakcie wyboru konfiguracji program zapyta czy plik z obserwacjami wejściowymi ma zostać wygenerowany, jeżeli tak to zapyta ile ich ma być (w przypadku gdy mamy własne obserwacje z których chcemy skorzystać).

**DataGenerator.cs** - statyczna klasa generująca pliki z danymi.

**Backpropagation.cs** - klasa propagacji wstecznej, inicjuje sieć z klasy **NeuralNetwork.cs**, na której operuje w trakcie nauczania, obsługa danych odbywa się za pomocą klasy z pliku **DataSet.cs**.

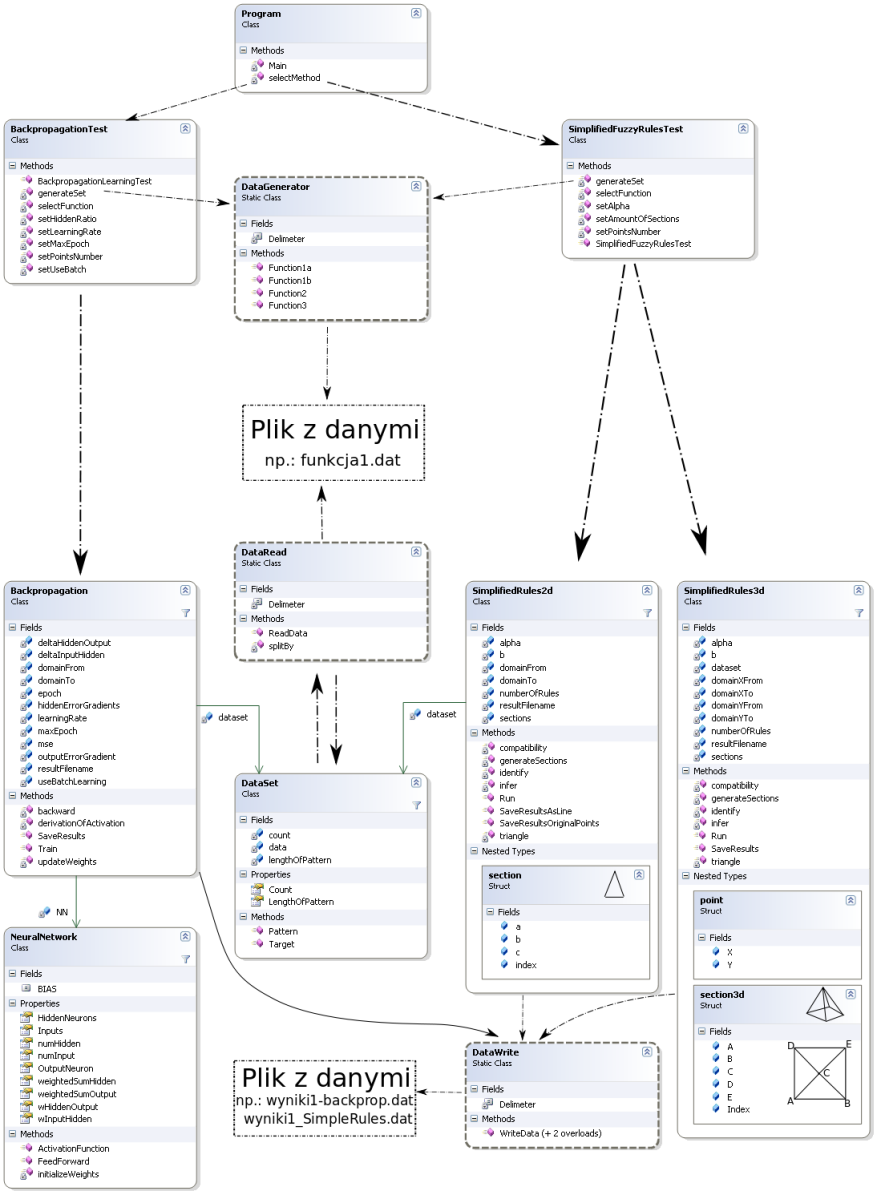
**NeuralNetwork.cs** - sieć neuronowa typu MLP.

**DataSet.cs** - klasa obsługująca przepływ danych, metod *double[] Pattern(index)* zwraca wektor obserwacji dla wskazanego indeksu, a *double Target(index)* wartość docelową dla wskazanego indeksu.

**SimplifiedRules2d.cs** - uproszczone reguły rozmyte dla funkcji jednej zmiennej, obszary trójkątne.

**SimplifiedRules3d.cs** - uproszczone reguły rozmyte dla funkcji dwóch zmiennych, obszary stożkowe.

**DataRead.cs & DataWrite.cs** - statyczne klasy do odczytu i zapisu do plików.




Rysunek 8: Schemat działania programu

# Bibliografia


[1] Ken Nozaki and Hisao Ishibuchi and Hideo Tanaka, A simple but powerful heuristic method for generating fuzzy rules from numerical data. *Fuzzy Sets and Systems*, 86, 1997, 251-270

[2] Funahashi, Ken-ichi and Nakamura, Yuichi, *Neural Networks, Approximation Theory, and Dynamical Systems*, 804, 1992

[3] Wierzchoń, S.T., *Obliczenia neuro-rozmyte*, Instytut Podstaw Informatyki PAN, Instytut Informatyki Uniwersytetu Gdańskiego, 16 maja 2011

[4] F. Girosi and T. Poggio. *Networks and the best approximation property* Technical Report A.I. Memo No. 1164, C.B.C.L Paper No. 45, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Department of Brain and Cognitive Sciences, October 1989.

[5] Bryson, A.E., and Ho, Yu-Chi. *Applied Optimal Control* Blaisdell, New York, 1969

[6] Werbos, Paul J., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Thesis, Applied Mathematics, Harvard University, November, 1974

[7] Duzinkiewicz K., *Sieci wielowarstwowe, uczenie*, Modelowanie i podstawy identyfikacji 2010/2011

[8] Park D.J., Jun B.E., Kim J.H. *Novel fast training algorithm for multilayer feedforward neural network*. KAIST, 1992

[9] Fahlman, S. *Faster-learning variations on back-propagation: An empirical study*. W: Proc. of 1988 Connectionist Models Summer School, D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski (eds.), Morgan Kaufmann Publishers, Los Altos CA, pp. 38-51

[10] D. T. Larose, *Discovering Knowledge in Data. An Introduction to DATA MINING*, J. Wiley & Sons 2005. Polskie tłumaczenie: Odkrywanie wiedzy z danych. Wprowadzenie do eksploracji danych, Wydawnictwo Naukowe PWN, Warszawa 2006, s.132