Informatyka, studia dzienne, inż. I st. semestr VI

Sztuczna inteligencja i systemy ekspertowe **2016/2017**

Prowadzący: dr inż. Krzysztof Lichy piątek, 10:15

Data oddania: _____ Ocena: _____

Alicja Gałkiewicz 195589

Maciej Ślusarz 189789

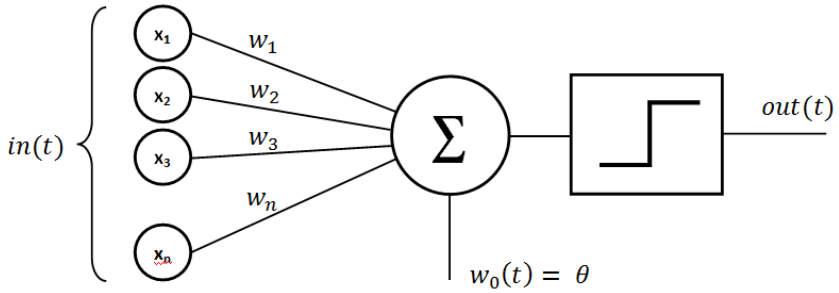
Zadanie 2: Sieć neuronowa

1. Cel

Celem zadania było napisanie programu implementującego sieć neuronową typu wielowarstwowy perceptron, nauczaną metodą wstecznej propagacji błędu. Sieć neuronowa miała nauczyć się obliczania pierwiastka drugiego stopnia liczby – czyli sieć ma za zadanie przeprowadzać aproksymację funkcji. Następnie, zbadano, jak wpływają parametry sieci i parametry uczenia sieci na wynik jej działania.

2. Wprowadzenie

Sieci neuronowe są cyfrową symulacją sposobu działania realnych biologicznych układów nerwowych poprzez symulację komórek nerwowych oraz ich połączeń. Stosując propagację wsteczną błędu przez sieć, możliwe jest dostosowanie współczynników (wag - inicjowanych jako losowe) połączeń między neuronami. W konsekwencji zwiększa się lub zmniejsza emfaza pewnych cech analizowanego sygnału wejściowego. Efektem tego procesu na danych których oczekiwaną wartość znamy, jest nauczenie sieci rozpoznawania cech i wzorców w danych wejściowych, co umożliwia późniejsze stosowanie jej do przetwarzania nieznanych wcześniej danych.

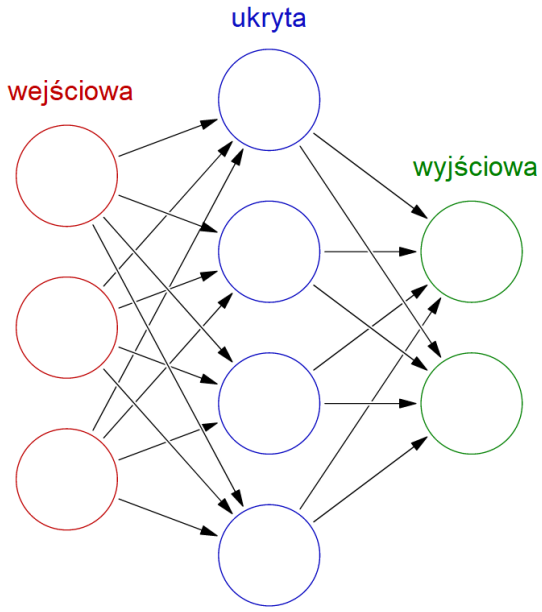


rys. 1. Model pojedynczego neuronu.

Zachowanie, elastyczność i czas uczenia się sieci jest w znacznym stopniu zależna od liczby neuronów wchodzących w jej skład. Neurony są pogrupowane w warstwy i podczas propagacji sygnału przez sieć, kluczowym jest zachowanie narzuconej przez warstwy kolejności rozpatrywania ich. Warstwy sieci neuronowej mogą być podzielone na trzy kategorie:

- wejściowa - składa się z określonej, stałej dla danej sieci liczby neuronów, każdy z nich przyjmujący cały wektor danych wejściowych
- ukryte - dowolna określona dla danej sieci liczba warstw z których pierwsza pobiera dane z warstwy wejściowej a pozostałe z poprzedniej warstwy ukrytej. Z założenia użytkownik nie ma bezpośredniego dostępu ani do sygnału wejściowego ani wyjściowego żadnego neuronu należącego do dowolnej warstwy ukrytej.
- wyjściowa - neurony wyjściowe pobierają dane z ostatniej warstwy ukrytej i są ostatnim czynnikiem przetwarzającym w sieci

neuronowej. Dane wyjściowe neuronów tej warstwy są bezpośrednio dostępne dla użytkownika.



rys. 2. Model sieci neuronowej typu wielowarstwowy perceptron.

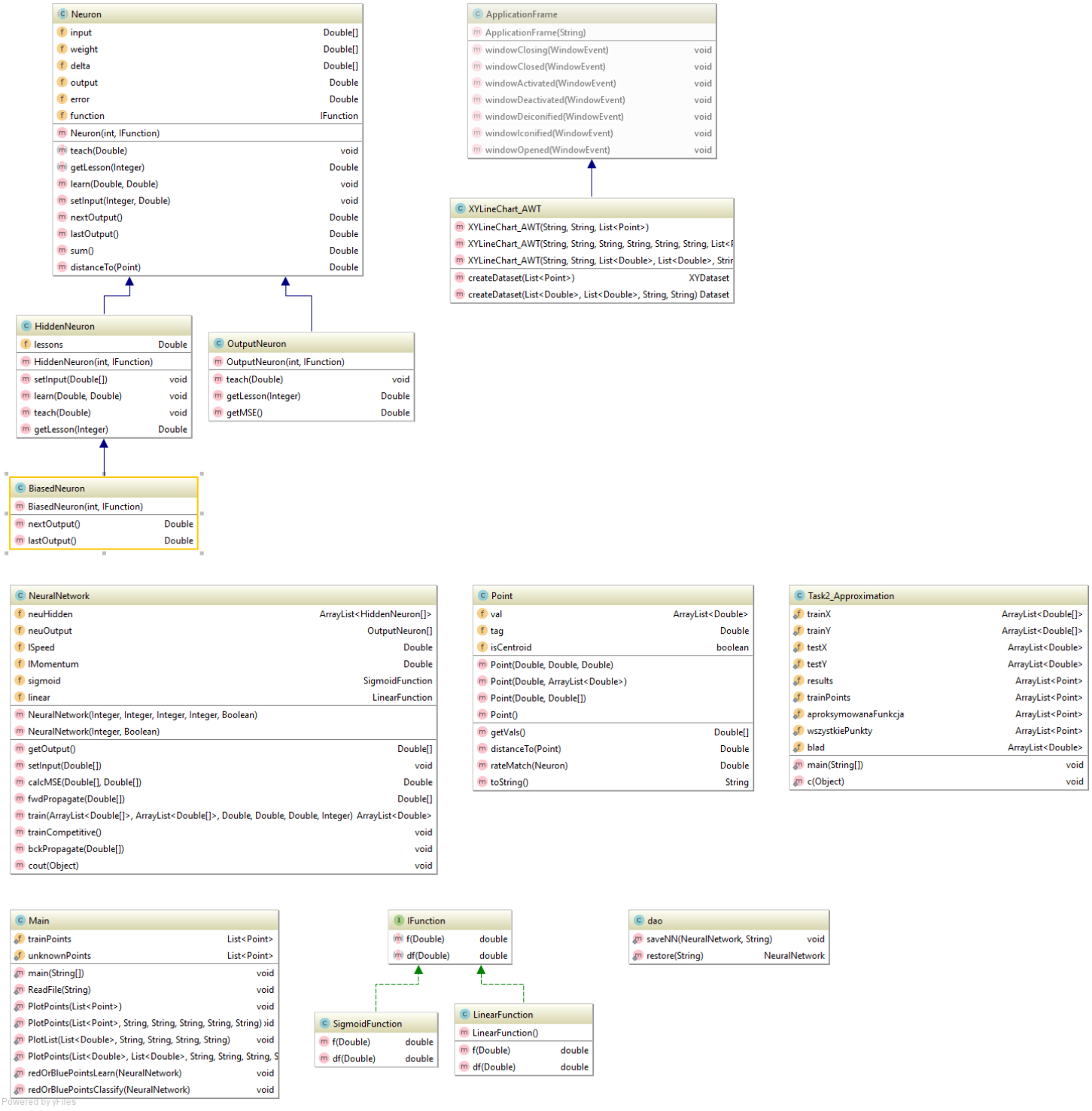
Zaprogramowana przez nas sieć jest przykładem tzw. sieci gęstej. Każdy neuron danej warstwy był połączony z każdym neuronem kolejnej. Dodatkowo, do przeprowadzenia części doświadczeń konieczne było utworzenie neuronu stronniczego (ang. biased neuron). Taki neuron niezależnie od danych jakie dostanie jako sygnał wejściowy, zwraca jedną, z góry określoną wartość wyjściową, najczęściej po prostu 1. Obecność takiego neuronu ma na celu zwiększenie elastyczności sieci poprzez poprawienie jej reakcji na skrajne, znacznie odbiegające od danych treningowych lub po prostu bliskie zeru dane wejściowe.

$$\delta(t+1) = f'\left(\sum_j w_j\,(t)\cdot i_j(t)\right)\cdot\sum_j w_j\,(t)\cdot\delta(t) \qquad (1)$$

Kolejnymi czynnikami wpływającymi na proces uczenia się sieci są prędkość oraz pęd (ang. momentum) uczenia. Podczas procesu propagacji wstecznej błąd $\delta\,(1)$ obarczony poprzednim rezultatem z każdego neuronu wyjściowego jest przekazywany wraz z wagami połączeń do ostatniej warstwy ukrytej a następnie wraz z ich błędami głębiej w sieć. W tym procesie wagi każdego z neuronów są poprawiane (2), a współczynnik wpływu błędu na modyfikację wagi to prędkość uczenia η. Dodatkowo korzystnym jest wykorzystanie wiedzy o modyfikacjach wagi w poprzednich krokach. Zastosowanie poprzedniej zmiany wagi może w znacznym stopniu przyspieszyć uczenie sieci. Współczynnik wpływu poprzedniej modyfikacji wagi na aktualną to pęd α (ang. momentum).

$$\Delta w(t+1) = w(t) + \eta\cdot\delta\cdot f\left(\sum_j w_j\,(t)\cdot i_j(t)\right) + \alpha\Delta w(t) \qquad (2)$$

3. Opis implementacji



Program został napisany obiektowo w języku Java, z wykorzystaniem biblioteki JFreeChart do generowania wykresów.

Jedną z najistotniejszych klas z punktu widzenia kodu jest klasa Neuron, będąca modelem pojedynczego neuronu. Klasa Neuron przechowuje dane wag wejść, wektor wejściowy, sygnał wyjściowy neuronu, błąd, a także funkcję aktywacji każdego neuronu. Klasa ta ma zaimplementowane metody obliczające odpowiednie wartości, np. sumowanie sygnałów wejściowych, obliczanie sygnału wyjściowego, czy modyfikację wag w ramach uczenia.

Po klasie Neuron dziedziczą klasy HiddenNeuron, OutputNeuron i BiasedNeuron, które różnią się chociażby sposobem implementacji uczenia neuronów.

Dodatkowymi klasami są klasy odpowiedzialne za modelowanie funkcji aktywacji: liniowej (dla neuronu z warstwy wyjściowej), lub sigmoidalnej (dla neuronów z warstw ukrytych). Klasy te mają również metody zwracające wartość pochodnej funkcji – implementacja pochodnej jest w postaci wzoru wpisanego ręcznie w kodzie.

Klasa NeuralNetwork odpowiada za przechowywanie neuronów-obiektów w odpowiednich strukturach danych a także za sam algorytm propagacji sygnału przez sieć, oraz wstecznej propagacji błędu. Ma metody przekazujące dane pomiędzy neuronami, obliczające błąd, zwracające wyjście sieci, itp.

4. Materiały i metody

W funkcji main naszego programu, na początku generujemy zbiór danych, który ma przypominać ciągłą funkcję $f(x) = \sqrt[2]{x}$. Jest to zbiór punktów (x_1, y_1), (x_2, y_2) … (x_n, y_n), takich, że dla każdego n: $y_n = \sqrt[2]{x_n}$, oraz dla każdego $n > 1$: $(x_n - x_{n-1}) = 0{,}01$. Ten zbiór punktów reprezentuje naszą funkcję **aproksymowaną**, a także posłuży jako **zbiór argumentów testowych**, by wygenerować podobną – „prawie-ciągłą" funkcję **aproksymującą**.

Następnie, wybraliśmy liczbę punktów treningowych = 50. Zostało wylosowanych 50 liczb z przedziału (1; 100), oraz każdej z tych liczb została przyporządkowana wartość funkcji $f(x) = \sqrt[2]{x}$. W ten sposób otrzymaliśmy listę punktów **treningowych**.

W kolejnym kroku została stworzona sieć neuronowa o 4 warstwach ukrytych oraz o 10 neuronach w każdej z ukrytych warstw.

Następnie sieć została nauczona na podstawie danych treningowych, z następującymi parametrami uczenia sieci:

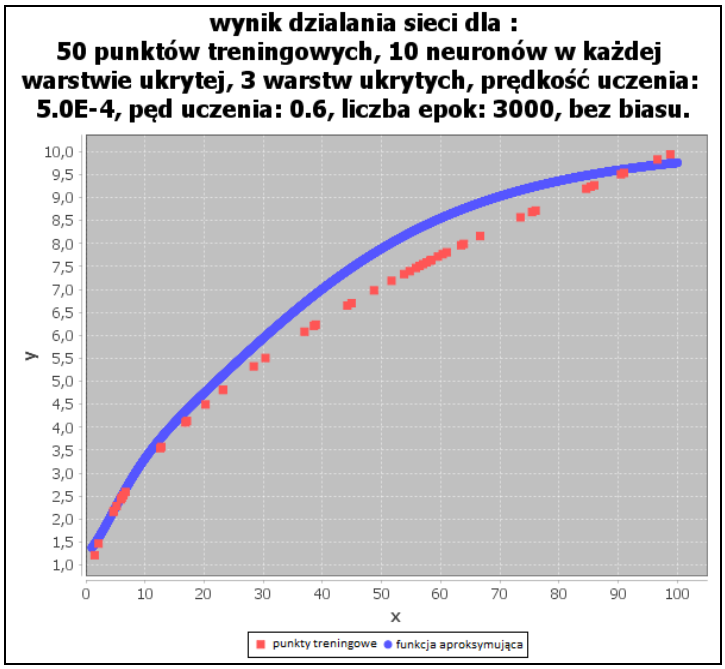
prędkość uczenia	0.0005,
pęd uczenia	0.6,
liczba epok uczenia	3000

Po zakończeniu uczenia, na wejście sieci została wprowadzona każda z liczb *x* ze **zbioru argumentów testowych**, i zostały utworzone pary liczb

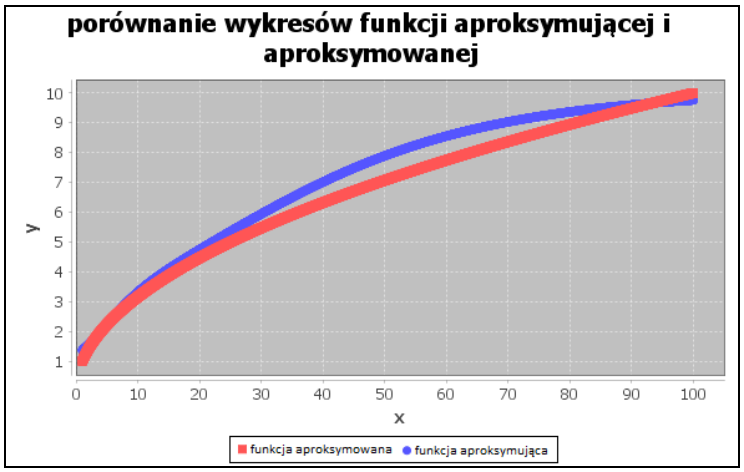
(x_n, y_n) takich, że y_n jest odpowiedzią sieci na wejście x_n. W ten sposób powstała lista punktów odpowiadająca **funkcji aproksymującej**.

Eksperyment powtórzono dla różnych parametrów sieci (liczba neuronów, liczba warstw, obecność neuronu - biasu), a także dla różnych parametrów uczenia sieci (prędkość uczenia, pęd uczenia, liczba epok). Wyniki działania sieci zostały przedstawione na wykresach poniżej.

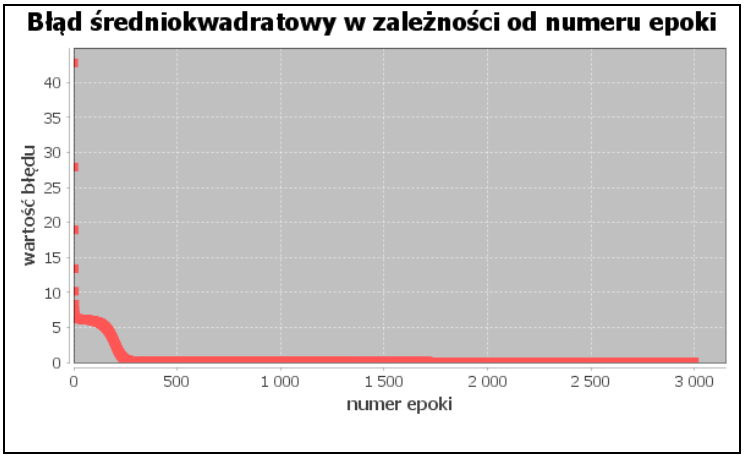
5. Wyniki
5.1



wykres 1.1 Wynik działania sieci dla pierwszego zestawu parametrów

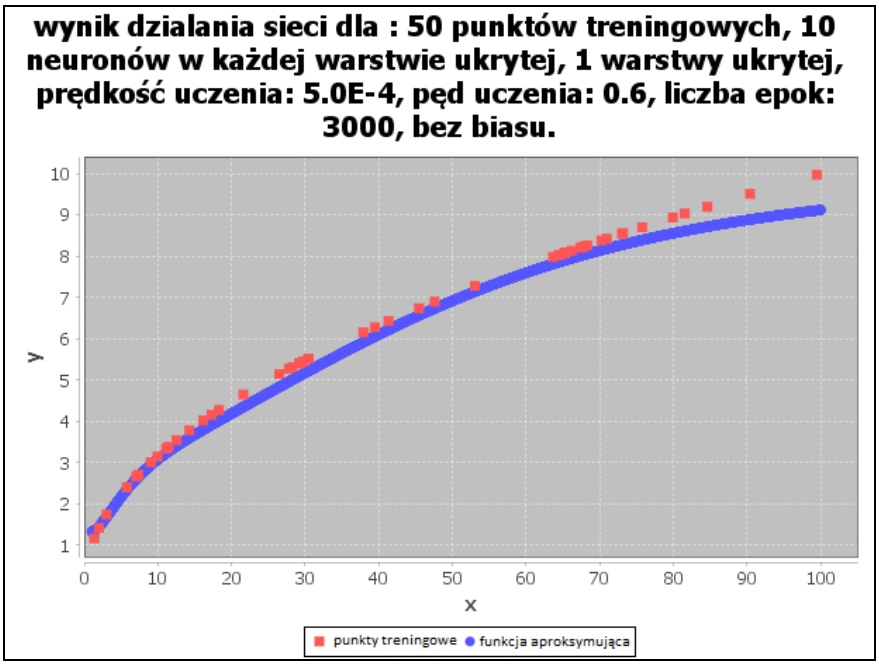


wykres 1.2 Porównanie funkcji aproksymującej i aproksymowanej dla pierwszego zestawu parametrów

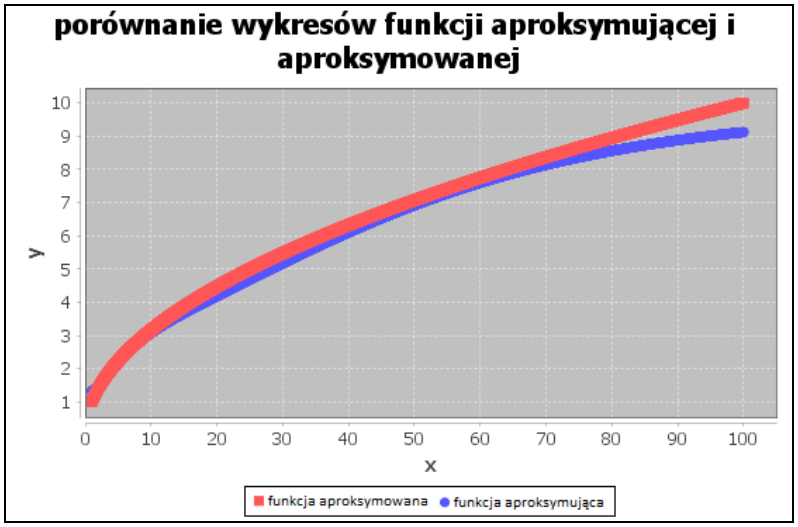


wykres 1.3 Wykres błędu w zależności od numeru epoki dla pierwszego zestawu parametrów.

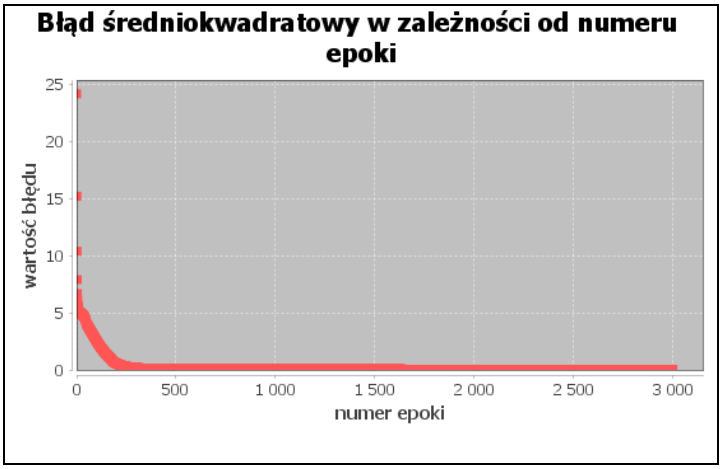
5.2



wykres 2.1 Wynik działania sieci dla drugiego zestawu parametrów

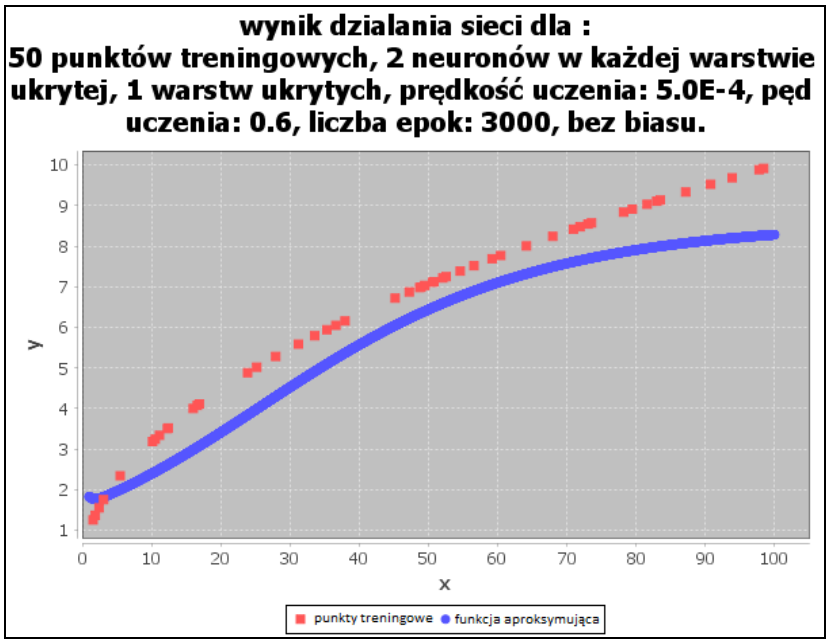


wykres 2.2 Porównanie funkcji aproksymującej i aproksymowanej dla drugiego zestawu parametrów

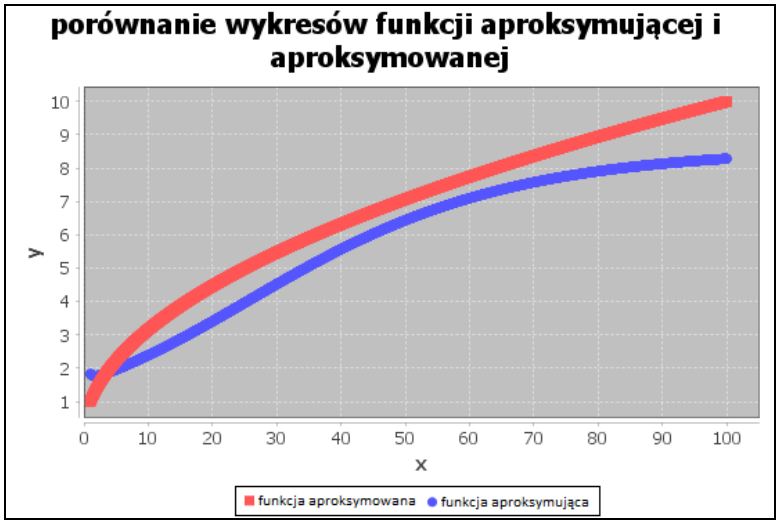


wykres 2.3 Wykres błędu w zależności od numeru epoki dla drugiego zestawu parametrów.

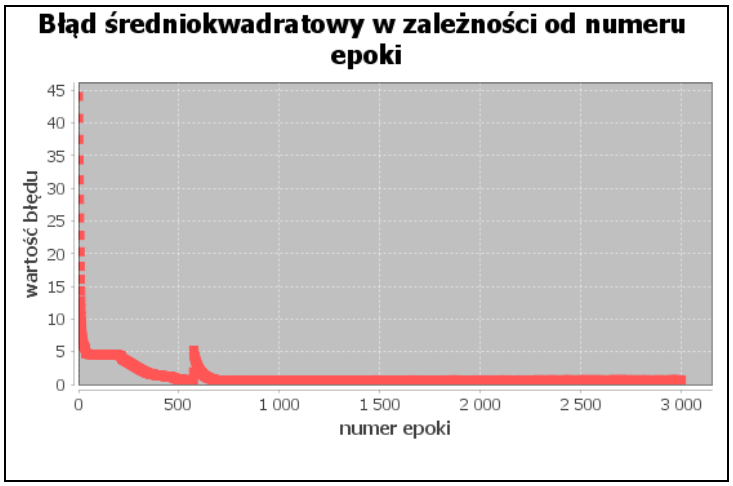
5.3



wykres 3.1 Wynik działania sieci dla trzeciego zestawu parametrów

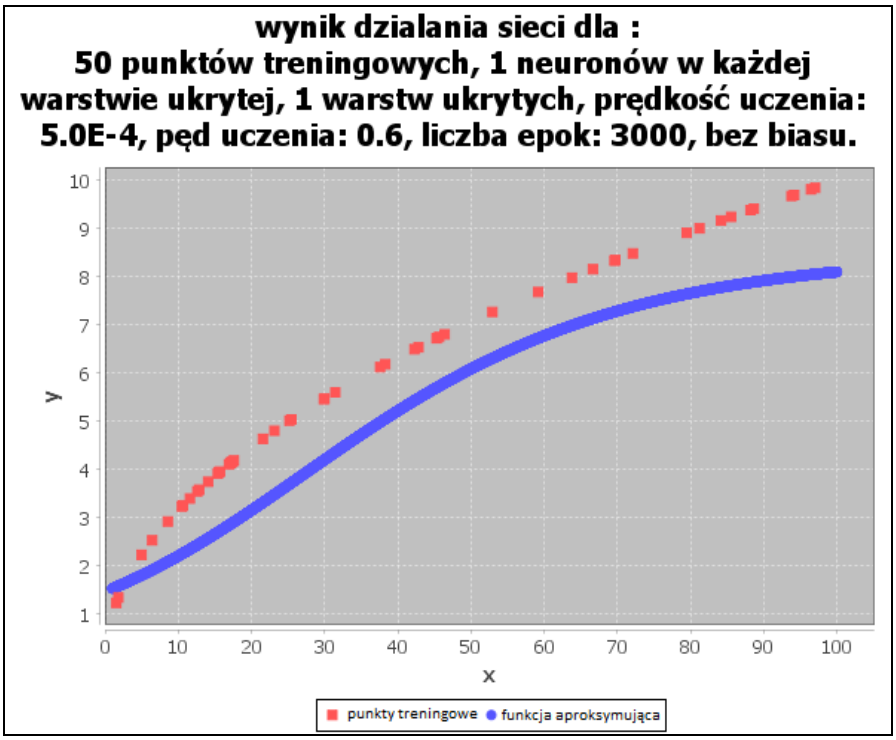


wykres 3.2 Porównanie funkcji aproksymującej i aproksymowanej dla trzeciego zestawu parametrów

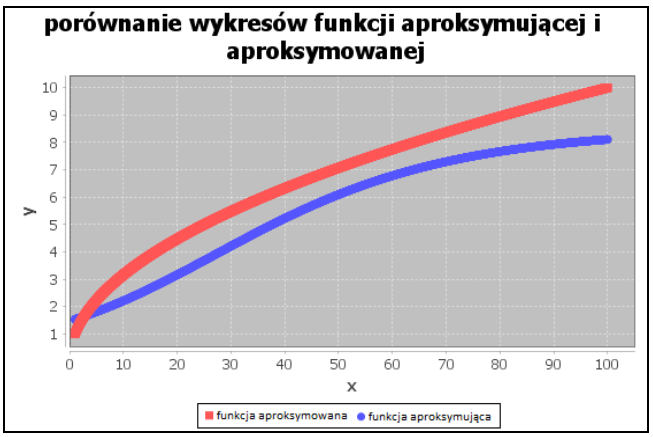


wykres 3.3 Wykres błędu w zależności od numeru epoki dla trzeciego zestawu parametrów.

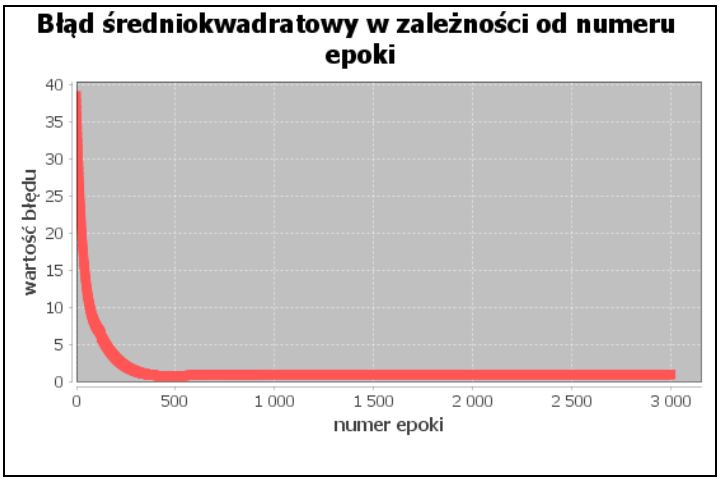
5.4



wykres 4.1 Wynik działania sieci dla czwartego zestawu parametrów

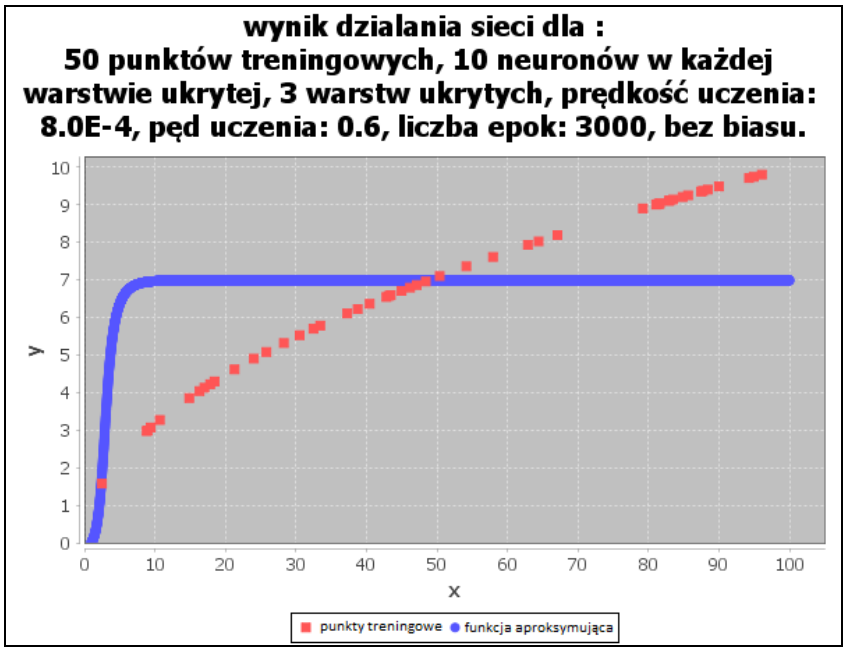


wykres 4.2 Porównanie funkcji aproksymującej i aproksymowanej dla czwartego zestawu parametrów

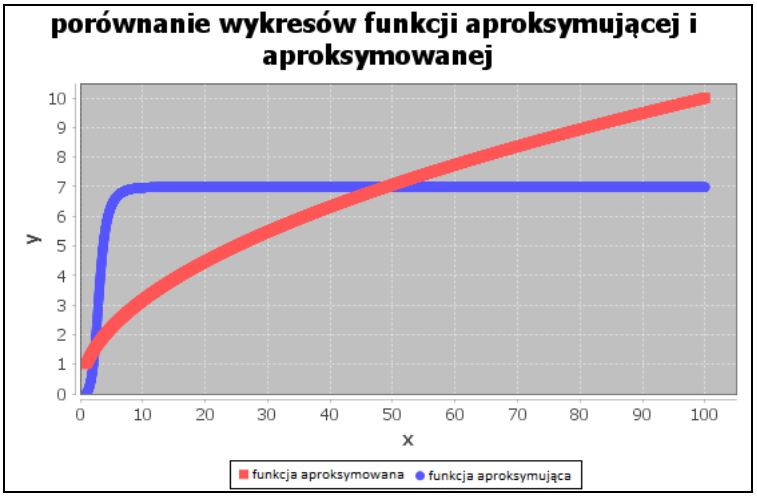


wykres 4.3 Wykres błędu w zależności od numeru epoki dla czwartego zestawu parametrów.

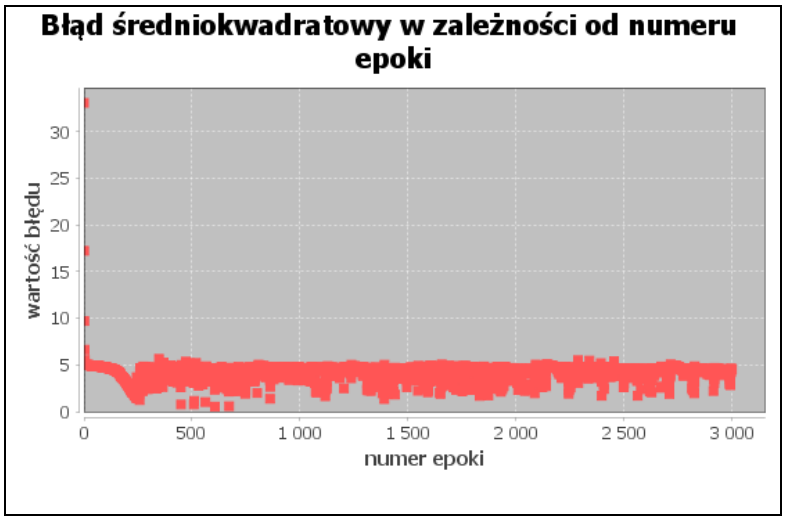
5.5



wykres 5.1 Wynik działania sieci dla piątego zestawu parametrów

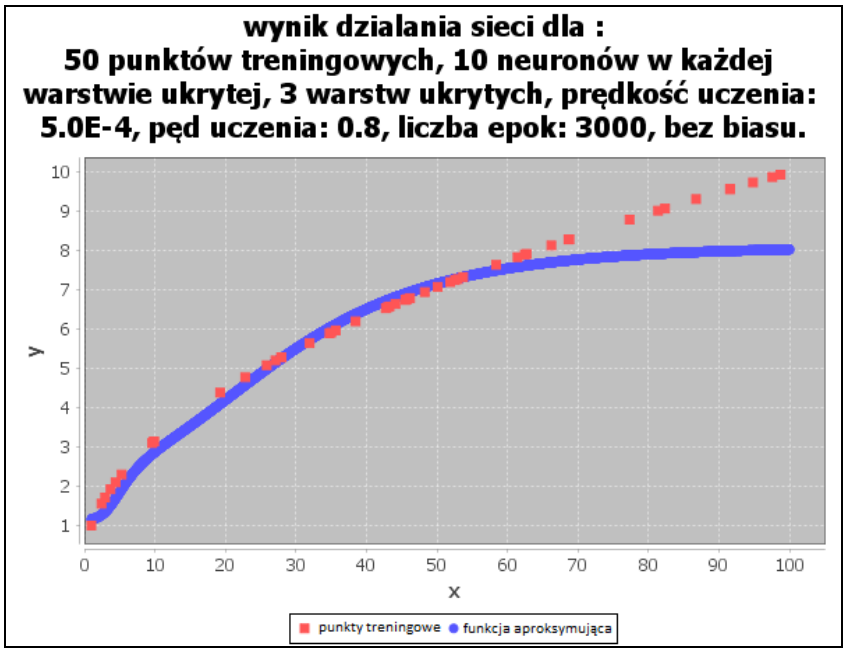


wykres 5.2 Porównanie funkcji aproksymującej i aproksymowanej dla piątego zestawu parametrów

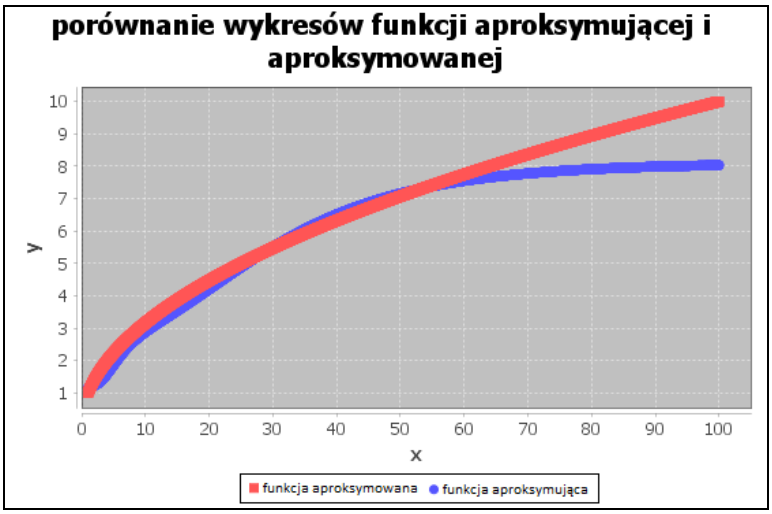


wykres 5.3 Wykres błędu w zależności od numeru epoki dla piątego zestawu parametrów.

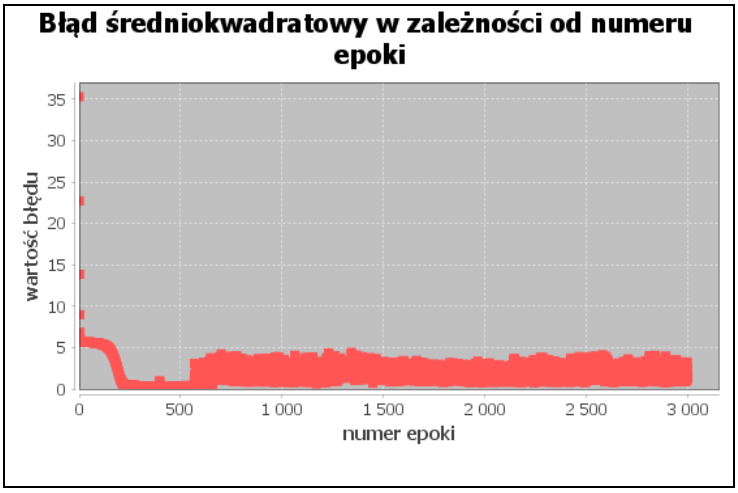
5.6



wykres 6.1 Wynik działania sieci dla szóstego zestawu parametrów

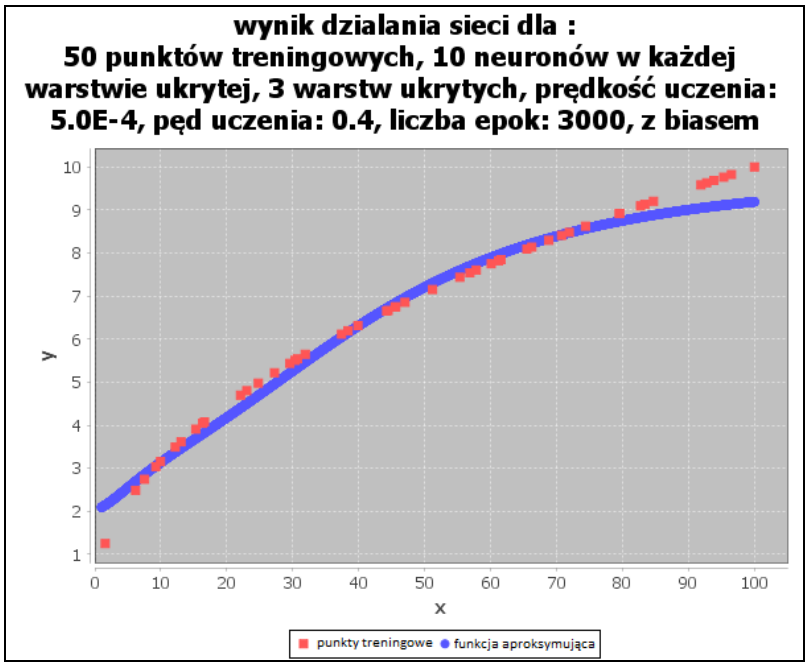


wykres 6.2 Porównanie funkcji aproksymującej i aproksymowanej dla szóstego zestawu parametrów

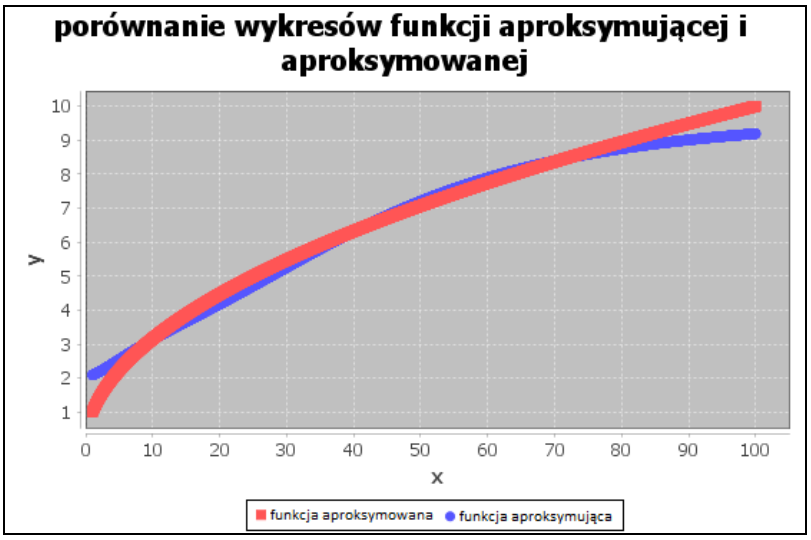


wykres 6.3 Wykres błędu w zależności od numeru epoki dla szóstego zestawu parametrów.

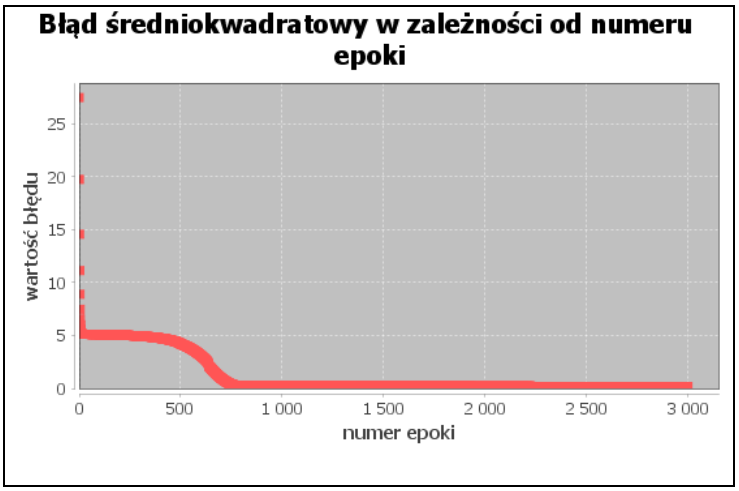
5.7



wykres 7.1 Wynik działania sieci dla siódmego zestawu parametrów

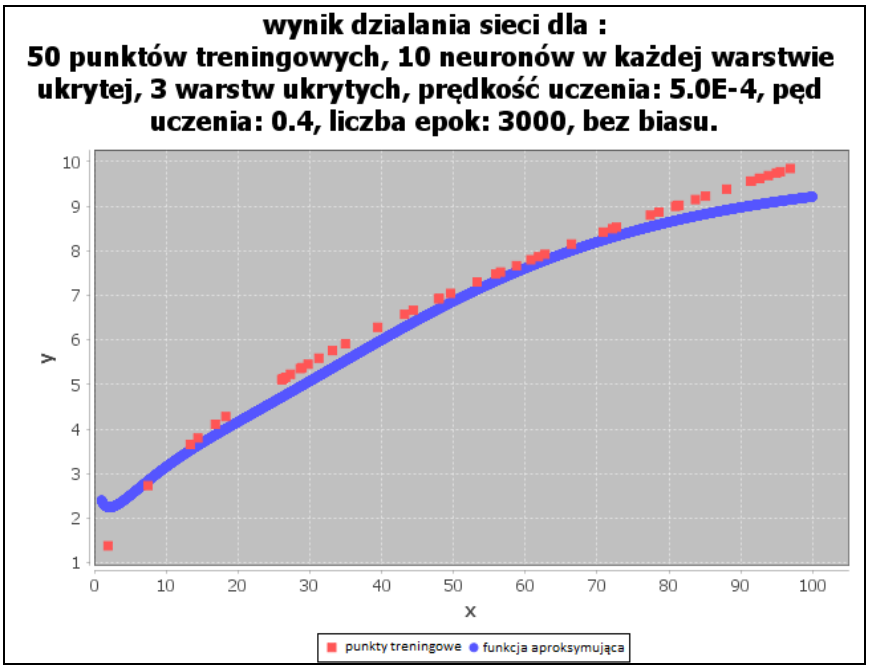


wykres 7.2 Porównanie funkcji aproksymującej i aproksymowanej dla siódmego zestawu parametrów

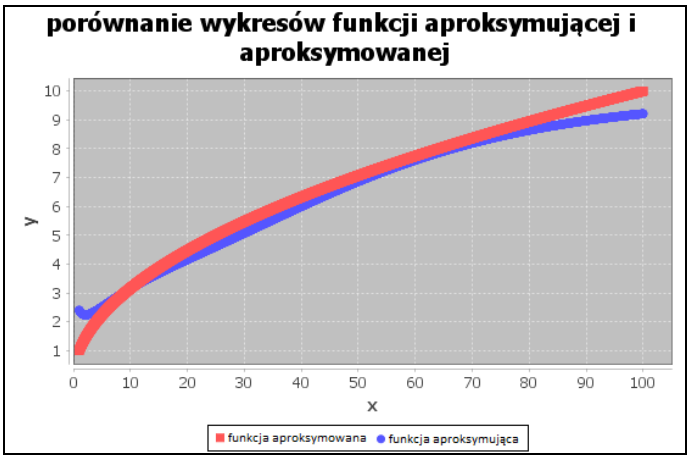


wykres 7.3 Wykres błędu w zależności od numeru epoki dla siódmego zestawu parametrów.

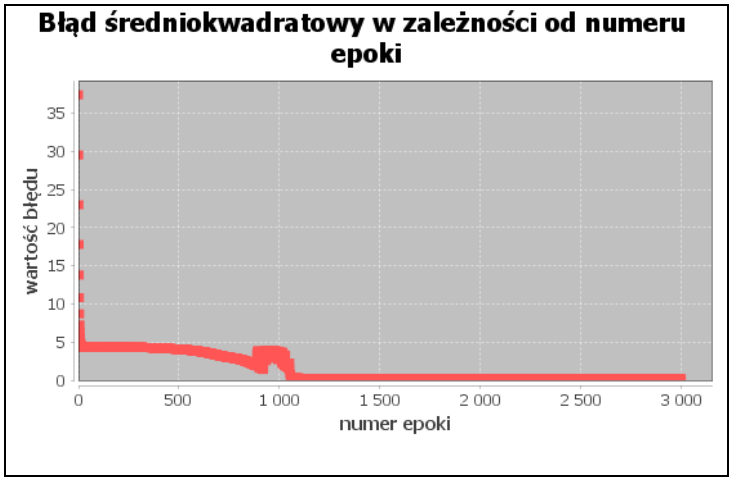
5.8



wykres 8.1 Wynik działania sieci dla ósmego zestawu parametrów



wykres 8.2 Porównanie funkcji aproksymującej i aproksymowanej dla ósmego zestawu parametrów



wykres 8.3 Wykres błędu w zależności od numeru epoki dla ósmego zestawu parametrów.

6. Dyskusja

Analizując wykresy funkcji aproksymującej dla **pierwszego** przykładowego zestawu parametrów, można dojść do wniosku, że sieć wykonuje zadanie zgodnie z oczekiwaniami. Wykres funkcji aproksymującej częściowo pokrywa się z wykresem funkcji aproksymowanej. Oceniając wzrokowo dokładność odwzorowania można stwierdzić że jest ona zadowalająca, mimo tego że funkcja w pewnych przedziałach nie pokrywa idealnie funkcji aproksymowanej. Zastanawiające jest to, że w przedziale w którym wylosowane punkty zostały rozmieszczone „najgęściej", funkcja przyjmuje wartości najbardziej oddalona od oczekiwanych w porównaniu z innymi przedziałami argumentów.

Analizując wykres błędu w zależności od numeru epoki można zaobserwować, że funkcja jest monotoniczna. Widoczny jest spadek wartości funkcji wraz ze wzrostem wartości argumentu. W okolicach epoki numer 200 występuje najszybszy spadek wartości błędu, później funkcja praktycznie stabilizuje się w bliskim otoczeniu zera. Być może do nauczenia sieci w tym przypadku do zadowalającego poziomu wystarczyłoby nawet mniej niż 500 epok.

Analizując działanie sieci z podobnymi parametrami, ale z tylko jedną warstwą ukrytą (**drugi** przykładowy zestaw parametrów), widać, że sieć praktycznie równie dobrze radzi sobie z nauczeniem się i z zadaniem aproksymacji. Dodatkowo, ograniczając liczbę neuronów między którymi dane muszą zostać wymienione, znacznie skraca się czas uczenia takiej sieci, jednakże nie zostało to przedstawione na żadnym z wykresów. Błąd podczas uczenia takiej sieci zachowuje się stabilniej niż w poprzednim przypadku – nie ma żadnych „gwałtownych skoków".

W przypadku **trzeciego** zestawu parametrów, użyliśmy dość małej liczby neuronów: tylko dwóch, w jednej warstwie ukrytej. W tym przypadku wyraźniej widać wpływ zmiany parametru na wynik działania sieci – wynikowa funkcja jest o wiele mniej dokładna. Mimo że ostatecznie osiągnięty błąd podczas uczenia sieci był porównywalnie mały w stosunku do poprzednich zestawów parametrów, to można zauważyć nietypowy „skok" w górę, w pewnym miejscu na wykresie. Jednakże nadal widać w pewnym stopniu podobieństwo funkcji aproksymowanej i aproksymującej.

W kolejnym – **czwartym** zestawie parametrów testowych, użyliśmy tylko jednego neuronu. Wyniki aproksymacji są równie słabe co w poprzednim przypadku, mimo że funkcja w pewnym stopniu przybliża funkcję $f(x) = \sqrt[2]{x}$. Ciekawą rzeczą, jaką można tu zauważyć jest fakt, że funkcja aproksymująca ma bardzo zbliżony kształt do funkcji aktywacji w neuronach warstwy ukrytej – przypomina wykresem funkcję sigmoidalną. Błąd podczas uczenia maleje w bardziej stabilny sposób niż w przypadku użycia dwóch neuronów.

Piąty przypadek pokazuje zestaw parametrów, gdzie odpowiednie nauczenie sieci nie było możliwe – co ciekawe, nie było to spowodowane zbyt

małą liczbą epok, neuronów lub warstw, a **zbyt dużym współczynnikiem prędkości uczenia.** Funkcja aproksymująca jest praktycznie stała na przedziale (10; 100), a bardzo gwałtownie rośnie w przedziale (0; 10), przypomina kształtem funkcję sigmoidalną. Błąd się nie stabilizuje, widoczne są anomalne skoki wartości błędu. Należy więc stwierdzić że w tym przypadku sieć zachowuje się nieprzewidywalnie.

W **szóstym** przypadku rozważamy wpływ pędu nauki na uczenie sieci. W początkowym przedziale wartości funkcja aproksymująca dość dokładnie pokrywa się z wartościami oczekiwanymi. Jednakże, w dalszym przedziale – około (60; 100), funkcja niedokładnie odwzorowuje oczekiwane wartości. Na wykresie błędu również jest widoczny wpływ pędu nauki na uczenie sieci – błąd zachowuje się niestabilnie.

W przypadku **siódmego** i **ósmego** zestawu parametrów, porównujemy wpływ obecności neuronu-biasu na działanie sieci, przy takich samych pozostałych parametrach. Biorąc pod uwagę wygląd wykresu błędu, można by stwierdzić, że bias pomaga przyspieszyć proces uczenia. Jednakże, jeśli chodzi o dokładność aproksymacji, to nie są zauważalne żadne znaczne różnice.

Pewnym utrudnieniem analizy wyników jest fakt, że zbiór treningowy był za każdym razem **losowany** – zgodnie z treścią polecenia. To znaczy, że wyniki nie są do końca porównywalne, ponieważ do każdego procesu uczenia wzięty został inny zbiór punktów.

Dodatkowo, należy zauważyć że wpływ kilku parametrów nie został przez nas przeanalizowany: chodzi o **liczbę punktów treningowych** oraz o **liczbę epok**. Można się jednak spodziewać, że w ogólności zwiększenie wartości tych parametrów poprawiłoby dokładność uczenia i działania sieci.

7. Wnioski

- Zwiększenie liczby warstw ukrytych niekoniecznie poprawia wynik działania sieci.
- Mała liczba warstw nie tylko nie pogarsza wyników, ale przyspiesza uczenie sieci, dzięki szybszej propagacji sygnałów między warstwami.
- Zbyt duża prędkość uczenia sieci uniemożliwia poprawne nauczenie sieci.
- Mała prędkość uczenia wymaga większej liczby epok do poprawnego nauczenia sieci, ale jest czasami opłacalna ze względu na dokładne wyniki.
- Współczynnik-pęd uczenia w niektórych przypadkach może pomóc nauczyć sieć, a w niektórych przypadkach sprawia że uczenie jest niedokładne, tak samo jak w przypadku zbyt dużej prędkości uczenia.
- Sieć neuronowa równie dobrze nadaje się do zastosowania przy problemie aproksymacji, co inne poznane przez nas wcześniej metody numeryczne.

Literatura

1. http://ml.informatik.uni-freiburg.de/_media/teaching/ss10/05_mlps.printer.pdf
2. https://www.hiit.fi/u/ahonkela/dippa/node41.html
3. http://www.cs.bham.ac.uk/~jxb/INC/l7.pdf