|  |
| --- |
|  |
| Wielowarstwowa, sigmoidalna sieć neuronowa ze wsteczną propagacją błędu. |
| Sprawozdanie |
|  |
| **Marcin Wróbel i Krystian Sikora** |
|  |

|  |
| --- |
|  |

**1. Opis algorytmu**

Przygotowany przez nas projekt jest przykładem wielowarstwowej sieci neuronowej wykożystującej sigmoidalną funkcję aktywacji oraz algorytm wstecznej propagacji błędów.

**Funkcja aktywacji** jest używana do obliczenia wyjściowej wartości neuronu. W naszej sieci używamy funkcji sigmoidalnej unipolarnej, której wynikiem jest wartość zawierająca się w przedziale (0,1). Ma ona postać , dla *x* równego sumie iloczynów wag i wejść danego neuronu:

gdzie to bias danego neuronu. Bias jest dodatkową wagą, której wejście jest zawsze równe 1. Dzięki tej wadze jesteśmy w stanie znaleźć wyjście dla neuronu, którego suma ważona wejść jest równa 0.

Po otrzymaniu wyniku funkcji aktywacyjnej przekazujemy go neuronom w następnej warstwie jako wejście, gdzie proces aktywacji jest powtarzany do momentu uzyskania końcowej wartości wyjściowej w ostatnim neuronie (neuronie wyjściowym).

Porównując otrzymaną wartość wyjściową sieci z oczekiwaną wartością dla danego wzorca otrzymujemy błąd na wyjściu sieci .

Aby zmniejszyć wartość błędu sieci wykorzystujemy **wsteczną propagacje błędów**. Dzięki temu algorytmowi możemy zmienić wagi wszystkich neuronów tak aby ich błąd był minimalny.

Otrzymany błąd należy pomnożyć przez pochochodną funkcji sigmoidalnej

, a więc:

to sygnał błędu dla neuronu wyjściowego *k*.

Dla neuronów w warstwach ukrytych sygnał błędu liczymy mnożąc sumę iloczynów wag i sygnałów błędów warstwy następnej (w tym przypadku wyjściowej), przez pochodną sigmoidalnej.

Zmianę wagi między neuronem *k,* a neuronem *j* z warstwy poprzedniej w epoce *i* liczymy w następujący sposób:

gdzie *xk* to wejście dla neuronu *k*, to stały współczynnik nauki, a to stała wartość momentu sieci.

Proces ten jest powtarzany do momentu otrzymania względnie małego błędu lub po przejściu przez daną ilość epok, dla każdego wzorca uczącego.

Algorytm w postaci krokowej:

1. Przejście przez sieć w celu uzyskania wartości wyjściowej.

2. Obliczenie błędu sieci.

3. Wsteczna propagacja błedu.

4. Aktualizacja wag.

5. Przejście do kolejnego wzorca, powrót do punktu 1.

6. Po przejściu przez wszystkie wzorce, powrót do punktu 1. przechodząc przez wszystkie wzorce od nowa lub zatrzymanie procesu uczenia jeśli nauczone wartości są zgodne z oczekiwaniem.

**2. Opis implementacji**

Algorytm uczący naszą sieć neuronową zaimplementowany został w języku C#.

Sieć zbudowana jest z obiektów klasy Neuron umieszczonych w listach (List<Neuron>) odpowiadającym warstwom. Listy te przechowywane są w obiekcie klasy NeuralNetwork, która zawiera metody uczące i uruchamiające sieć (Train(); oraz Test();). W konstruktorze klasy NeuralNetwork tworzona jest wybrana przez użytkownika ilość neuronów, oraz ustawiane są parametry sieci (współczynnik nauki, momentum, ilość wejść itd.).

Do konstruktora pojedynczego Neuronu, przekazywana jest liczba wejść, która zależy od ilości wejść we wzorcach uczących, a także od numeru warstwy. Liczba wejść w neuronie warstwy 2 będzie odpowiadać ilości wyjść (ilości neuronów) w warstwie 1. W konstruktorze losowane są wagi z przedziału (-1, 1) dla każdego z wejść. Klasa Neuron posiada metodę aktywującą oraz metodę aktualizującą wagi.

public void Train()

{

int epoka = 0;

double error;

double totalError = 0;

while (epoka < maxEpok) //pętla ucząca

{

epoka++;

foreach (Pattern pattern in learningSet.PatternsNorm)

{

outputValue = GetOutput(pattern.Inputs);

error = pattern.Output - outputValue;

UpdateWeights(error, learningRate, momentum);

totalError += Math.Abs(error\*error);

}

Manager.CurrentEpoka = epoka;

Manager.CurrentError = Math.Sqrt(totalError/(double)learningSet.PatternsNorm.Count);

Manager.BGWorker.ReportProgress(epoka);

Manager.Busy.WaitOne();

if (Manager.BGWorker.CancellationPending == true)

{

break;

}

if (Manager.CurrentError < Manager.ErrorMax)

{

break;

}

else totalError = 0;

}

}

Metoda aktywująca Activate(double[] inputs) sumuje iloczyny wag i wejść neuronu i zwraca wyjście uzywając funkcji sigmoidalnej. Funkcję tę wykonuje statyczna metoda Sigmoid(double x) klasy (MathFunctions). W parametrze metody Activate() podajemy tablicę double[] zawierającą wartości wejściowe.

Uczenie sieci zachodzi w metodzie Train(). W pętli while wykonujemy dla każdego wzorca Pattern (klasa zawierająca tylko tablicę wejść i docelowe wyjście) kolejno:

* Obliczenie wyjściowej wartości sieci:
  + Uruchomienie metody Activate dla każdego neuronu w danej warstwie
  + Przekazanie wyjść warstwy jako wejście warstwy następnej
  + Powtórzenie aktywacji aż do otrzymania ostatniego wyniku (dojścia do warstwy wyjściowej)
* Obliczenie błędu.
* Uruchomienie metody UpdateWeights, która wykonuje algorytm wstecznej propagacji błędów i aktualizuje wagi wszystkich neuronów.
* Dodanie do zmiennej totalError wartości błędu podniesionej do kwadratu.

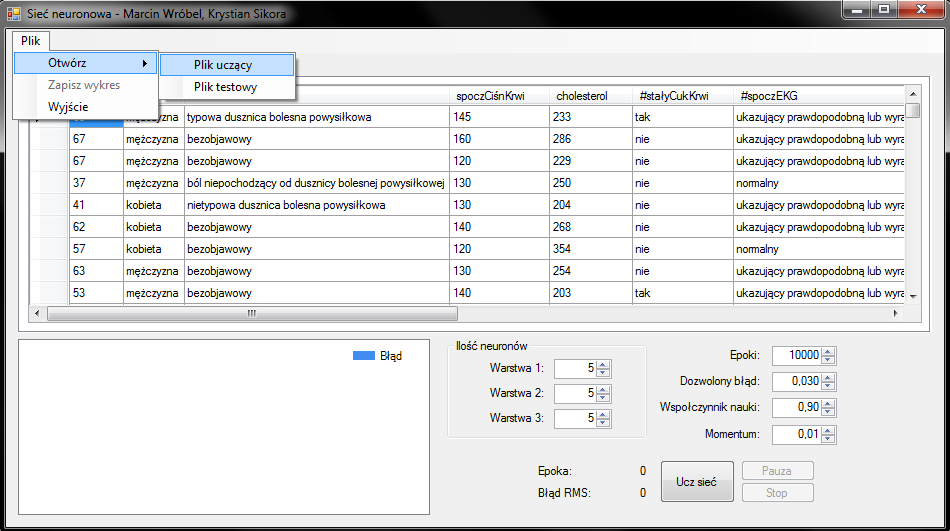
Po wykonaniu powyższych podpunktów dla wszystkich wzorców obliczany jest błąd RMS (root mean square – średnia kwadratowa). Jeśli błąd jest mniejszy od maksymalnego błędu wyznaczonego wcześniej przez użytkownika pętla jest przerywana. Pętlę również można przerwać za pomocą przycisku „Stop” (stąd warunek if (Manager.BGWorker.CancellationPending == true)) .

Jeśli pętla nie zostanie przerwana, jest wykonywana od początku dla wszystkich wzorców do momentu osiągnięcia maksymalnej ilości epok.

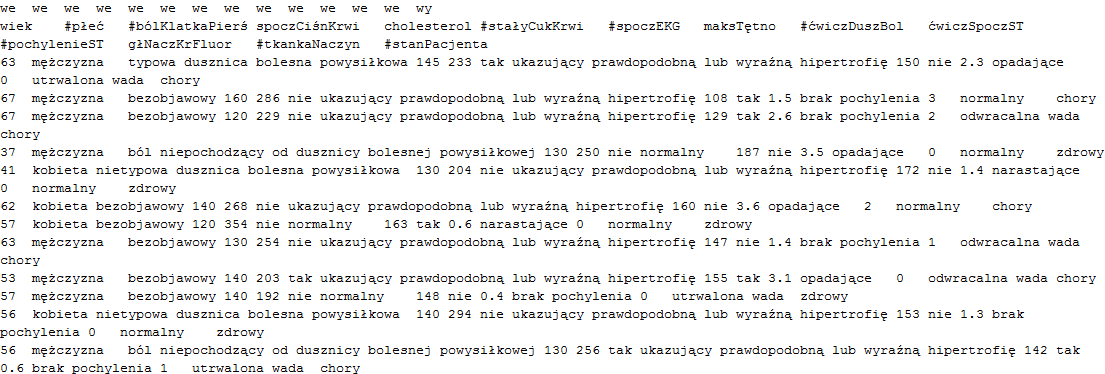
Wzorce przechowywane są w obiekcie klasy TrainingData w postaci znormalizowanej. Po zakończeniu nauki wyjście denormalizujemy, by pozwolić użytkownikowi na czytelniejszy podgląd wyników nauki.

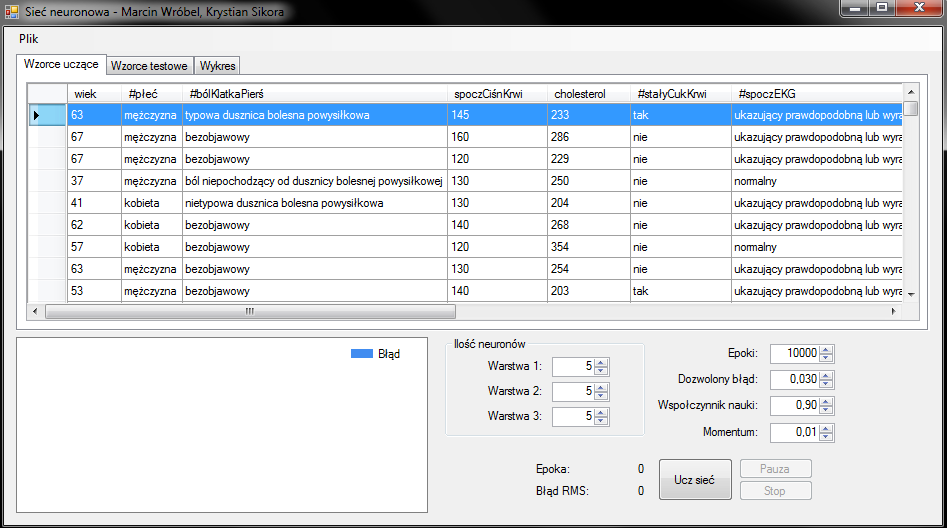
Metoda Test() wywoływana jest podczas testowania sieci. Wykonuje ona metodę GetOutput() (czyli metodę uzyskującą wyjście sieci) dla każdego z wzorców, a następnie zwraca zdenormalizowane wartości do listy. Wyniki te można podejżeć w tabeli danych w zakładce „Wzorce testowe”.

**3. Przykład działania sieci**



Z menu „Plik” wybieramy „Otwórz -> Plik uczący”. Program obsługuje pliki tekstowe z rozszeżeniem \*.lrn z danymi w postaci tabeli (można swobodnie używać plików .lrn zawartych w pakiecie Sphinx 4.0).

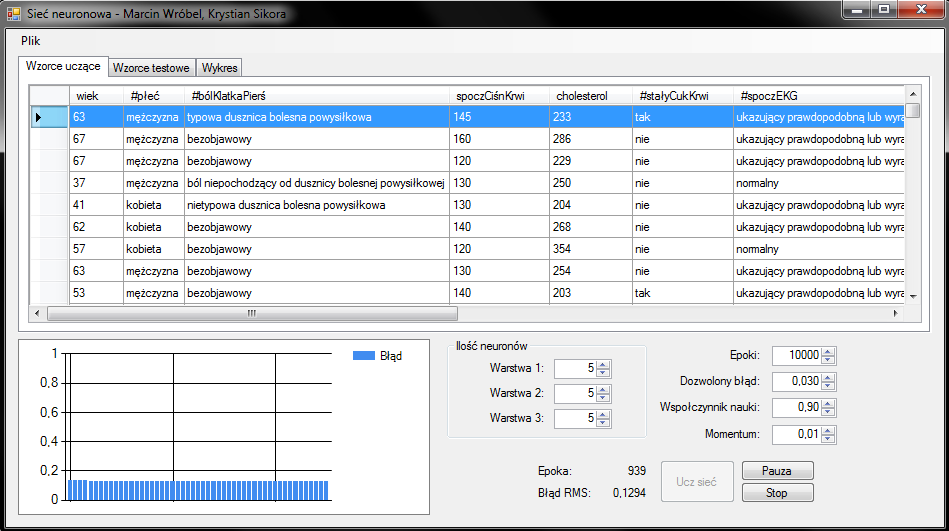




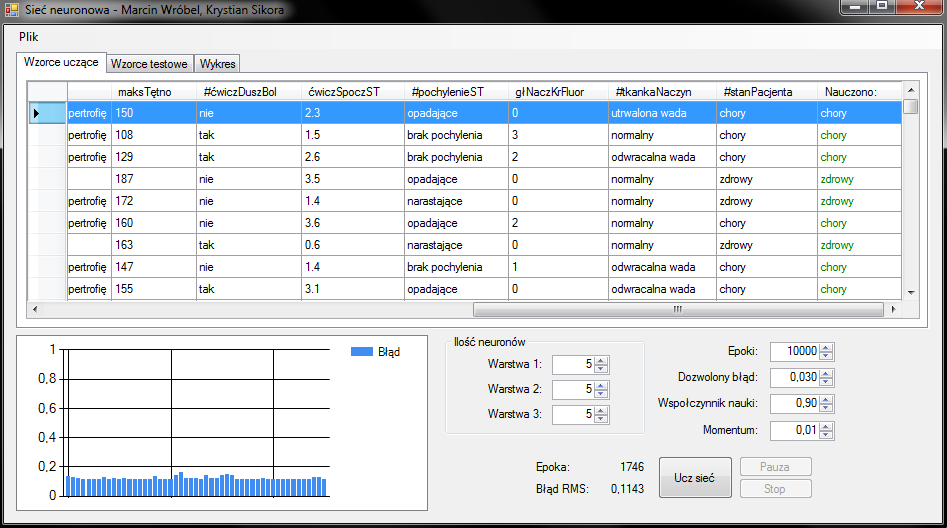
Wybrany przez nas plik wyświetli się nam w zakładce „Wzorce uczące”

Jeśli dany plik uczący był już wcześniej używany w naszej sieci, zostaną załadowane poprzednio używane parametry sieci (wczytywane z pliku \*.set).

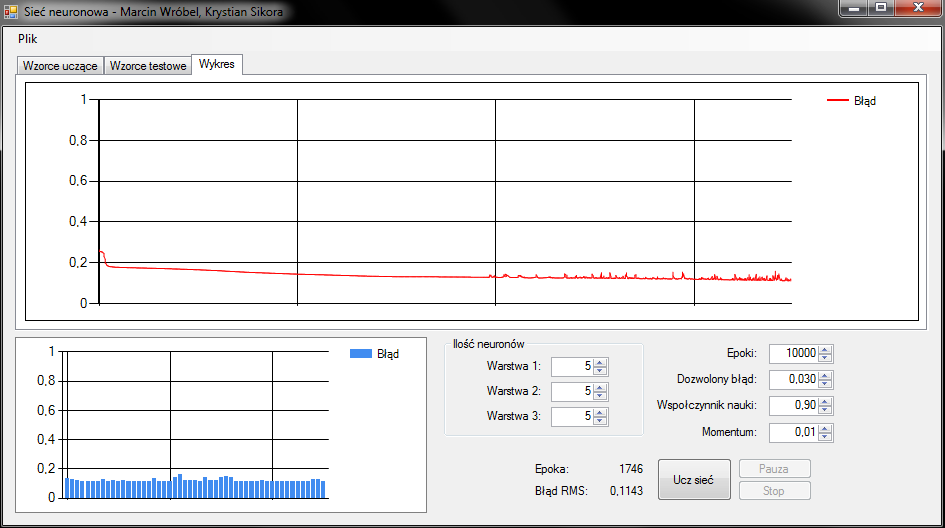
Po wprowadzeniu interesujących nas parametrów sieci (ilość neuronów, epok itd.) można przystąpić do uczenia sieci. Po wciśnięciu przycisku „Ucz sieć” uruchamiana jest medota Train() klasy NeuralNetwork(). Za każdym kliknięciem „Ucz sieć” początkowe wagi są losowane z przedziału (-1, 1).



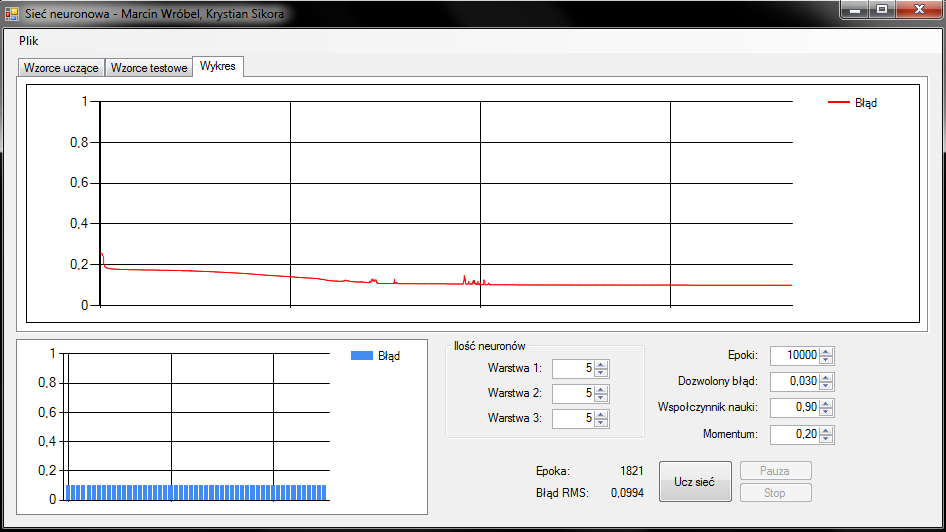
W trakcie trwania procesu uczenia widoczna jest aktualna epoka oraz błąd RMS w postaci liczbowej oraz wykresu, który na bierząco pokazuje błędy dla ostatnich 50 epok. Uczenie można całkowicie przerwać lub tymczasowo wstrzymać (przyciski „Pauza” i „Stop”).



Po zakonczeniu procesu nauki w ostatniej kolumnie tabeli wzorców można sprawdzić czy wartości nauczone przez sieć są zgodne z oczekiwanymi wartościami wyjściowymi (nauczone prawidłowo są oznaczone zieloną czcionką, a nieprawidłowe czerwoną).

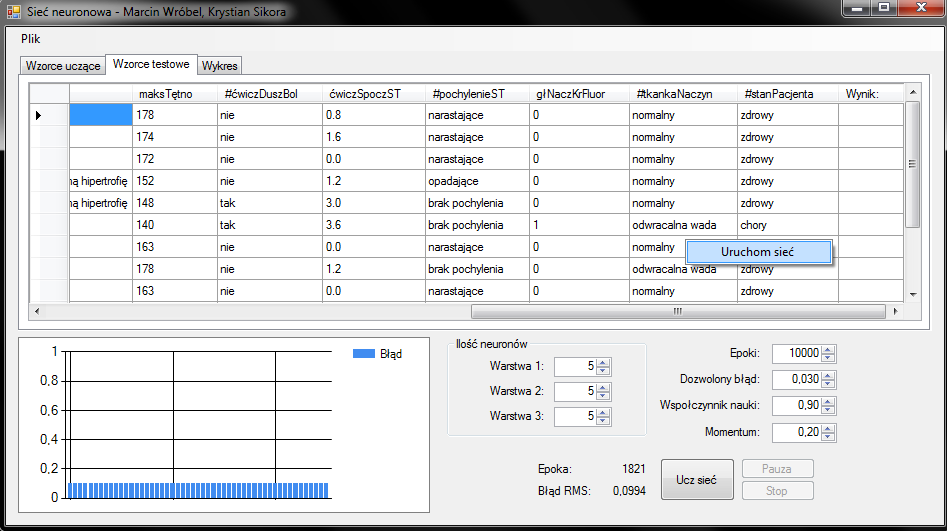


Zakładka wykres przedstawia błąd RMS dla wszystkich epok. Jak widać dla wzorców z przykładowego pliku uczącego ChorobySerca.lrn zawartego w pakiecie Sphinx 4.0, błąd zmniejsza się z początku w bardzo szybkim tempie, następnie zwalnia. Przy końcowych epokach wykres wskazuje wachającą się wartość błędu. Taki efekt występuje, gdy sieć natrafi na lokalne minimum. Aby nasza sieć nie „utknęła” na jednej wartości należy odpowiednio dostosować wartość momentum oraz współczynnik nauki.

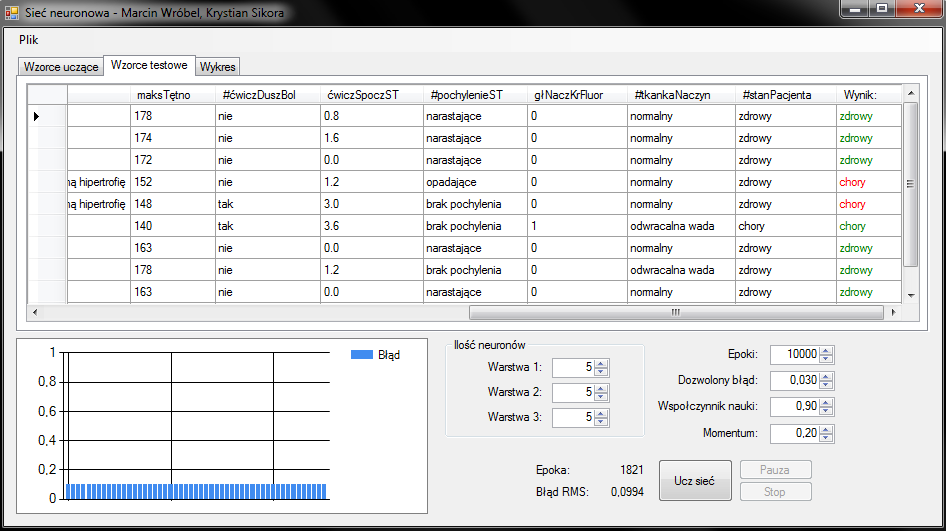


Wachania są znacznie mniejsze po zmianie momentum z 0,01 na 0,2.

Aby móc przetestować naszą nauczoną już sieć należy z menu „Plik” wybrać opcję „Otwórz -> Plik testowy”, który powinien zawierać wzorce testowe. Ważne jest, by etykiety oraz ilości wejść zgadzały się z plikiem uczącym. W przeciwnym wypadku plik testowy nie załaduje się poprawnie.



Przechodzimy do zakładki „Wzorce testowe” następnie klikamy prawym klawiszem myszki na naszą tabelę danych i wybieramy opcję „Uruchom sieć”.



Uzyskamy wyniki, które sieć uznaje za poprawne. Jak widać na powyżej sieć nie zwróciła prawidłowo dwóch wartości. Aby uzyskać bardziej dokładne dane należy naszą sieć uczyć do momentu uzyskania możliwie najmniejszego błędu. W niektórych przypadkach uzyskanie 100% dokładnych wartości może być niemożliwe.