Dokumentacja projektu z przedmiotu:

Sztuczne sieci neuronowe

Temat projektu:

Zastosowanie sieci neuronowej do gry w kółko i krzyżyk

Autorzy:

Andrzej Węgrzyn

Justyna Michalik

Paweł Noga

Marcin Komisarz

Data wykonania:

13.06.2016

Spis treści

1 Wstęp

1.1 Cel projektu

Projekt ma na celu zaimplementowanie gry kółko i krzyżyk gdzie gracz będzie mógł zmierzyć się przeciwko komputerowi. W tym przypadku role przeciwnika dla gracza będzie pełniła sieć neuronowa. W ramach projektu poza zaimplementowaniem rozgrywki istotnym elementem jest zaprojektowanie sieci neuronowej, która będzie stanowiła odpowiednie wyzwanie dla realnego gracza.

1.2 Zasady gry w kółko i krzyżyk

Gra rozgrywa się na planszy o rozmiarach 5x5. Każdy z graczy ma przypisany do siebie symbol, kółko lub krzyżyk. W każdej turze jeden z graczy na wolnym polu planszy umieszcza swój symbol. Następnie następuje tura przeciwnika. Gra toczy się do momentu aż któremuś z graczy uda się ułożyć 4 symbole obok siebie, w linii poziomej, pionowej lub po skosie. Rozgrywkę rozpoczyna losowy gracz.

2 Rozwiązanie problemu

2.1 Przygotowanie algorytmu generującego dane uczące

Pierwszym zadaniem było przygotowanie formatu danych uczących. Dane wejściowe, na które składają się poszczególne stany planszy, zostały zaimplementowane jako zbiór wektorów. Każdy wektor reprezentuje stan planszy, dla którego sieć ma znaleźć odpowiedź. Odpowiedzią na taki wektor jest para punktów, mówiąca w jakim miejscu sieć ma postawić swój symbol.

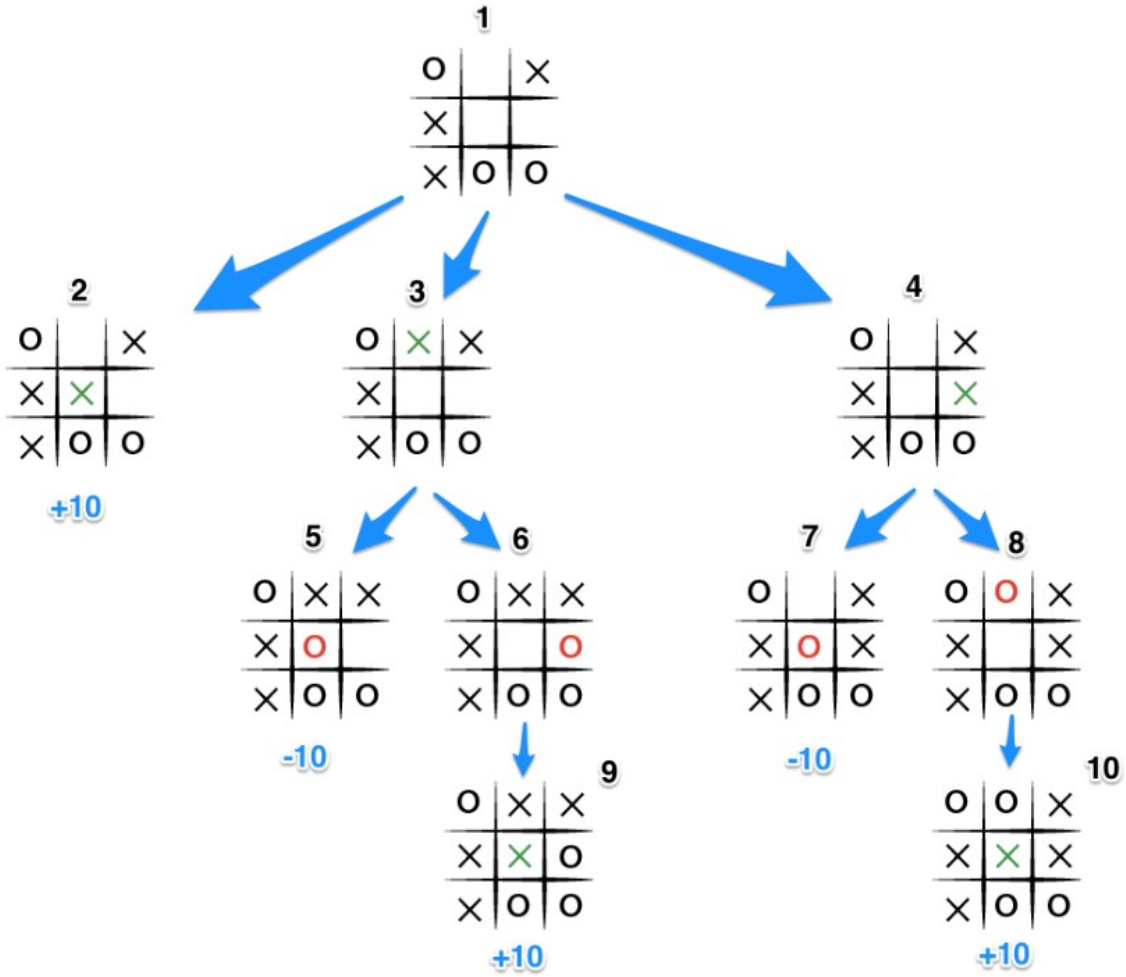
Przy tak zdefiniowanym formacie danych należało znaleźć algorytm, który wygeneruje nam dowolna ilość wektorów reprezentujących plansze, oraz optymalne punkty w którym należy postawić symbol w następnej kolejce.

Zdecydowaliśmy się na wykorzystanie algorytmu min-max. Jest to wywodzący się z teorii gry o sumie zerowej algorytm, który służy do minimalizacji maksymalnych strat. Przenosząc to na grunt gry w kółko i krzyżyk, możemy założyć, że wygrana gra daje nam +10 punktów, przegrana -10 (ponieważ to przeciwnik zdobywa 10 punktów), zaś remis daje nam 0. Celem każdego z graczy jest maksymalizacja swojego wyniku. Głównie oznacza to wygrana, ale w niektórych przypadkach, gdy wygrana jest niemożliwa maksymalizacja wyniku będzie remis.

Dzięki takiemu opisowy gry, możemy stworzyć algorytm, który będzie starał się maksymalizować wygraną w każdym ruchu. Algorytm składa się z następujących kroków:

- stwórz listę możliwych plansz na podstawie obecnego stanu planszy
- dla każdego ze stanów przypisz wartość punktową dla każdego z graczy
- jeżeli gra nie jest zakończona dla planszy, przejdź do kolejnych wygenerowanych plansz i ponownie rozpocznij algorytm

Poniżej znajduje się przykład algorytmu dla przykładu 3x3.



Stan 1 to obecny stan planszy. Dla gracza grającego krzyżyk mamy 3 możliwe plansze 2, 3, 4. Plansza 2 daje nam zakończenie gry ze zwycięstwem gracza X, przez co nadajemy jej wartość +10. Plansze 3 i 4 nie kończą gry, dlatego dla nich generujemy kolejne plansze, tym razem z możliwymi ruchami gracza O. Jako że dwie z nich kończą się zwycięstwem gracza O przypisujemy im wartość -10. Dla pozostałych dwóch generujemy kolejne możliwe plansze 9 i 10, które kończą się zwycięstwem X, więc otrzymują wartość +10.

Ponieważ każdy z graczy dąży do maksymalizacji wyników, gracz X wybierze od razu opcje 2.

Wykorzystując ten algorytm możliwe jest stworzenie wszystkich możliwych zwycięskich kombinacji. Niestety wadą tego algorytmu jest jego wysoki koszt obliczeniowy, spowodowany koniecznością generowania wszystkich możliwych plansz. Możliwe jest jego optymalizowanie, na przykład przez ograniczenie ilości tur, w których gracze muszą osiągnąć zwycięstwo.

3 Test sieci

3.1 Przygotowanie zestawu danych

Na podstawie wcześniej przedstawionego algorytmu wygenerowano zestaw danych uczących oraz testujących.

Należy przy tym zauważyć, że generowanie tychże zbiorów było bardzo czasochłonne i wygenerowanie zestawu składającego się z 5000 przykładowych plansz oraz ich kolejnych ruchów trwało kilka godzin, dlatego przygotowano następujące zbiory danych

- Rozmiar zbioru uczącego: **5000**
- Rozmiar zbioru testowego: **100**

3.2 Test sieci

W ramach projektu przetestowano 2 rodzaje sieci:

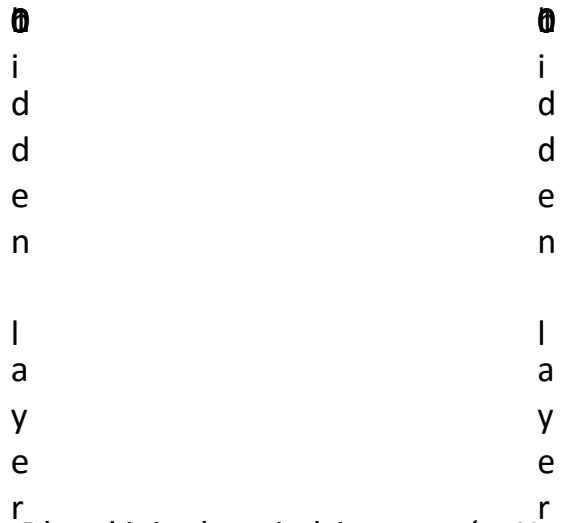
- Sieć jednokierunkowa wielowarstwowa
- Sieć RBF

Kolejnym krokiem był dobór wyjścia sieci spośród 3 następujących koncepcji:

1. W warstwie wyjściowej 2 neurony. Wyjście neuronu nr 1 określa współrzędną punktu X na planszy, z kolei wyjście neuronu nr 2 określa współrzędną Y. (problem aproksymacji).
   Np.

   Dla takiej odpowiedzi neuronów X = 2, Y = 3.

2. W warstwie wyjściowej 5 neuronów. Indeks pobudzonego neuronu w warstwie wyjściowej określa współrzędną. W takim wariancie potrzebujemy 2 sieci, gdzie pierwsza daje odpowiedź dla współrzędnej X, natomiast druga dla współrzędnej Y. (problem klasyfikacji).
   Np. sieć dla Wsp. X sieć dla WSP. Y



   Dla takiej odpowiedzi neuronów X = 5, Y = 2.

3. W warstwie wyjściowej 25 neuronów. Indeks pobudzonego neuronu określa obie współrzędne. Np. jeżeli pobudzony zostanie neuron nr. 20, oznacza to współrzędne: x = 4, y = 5.

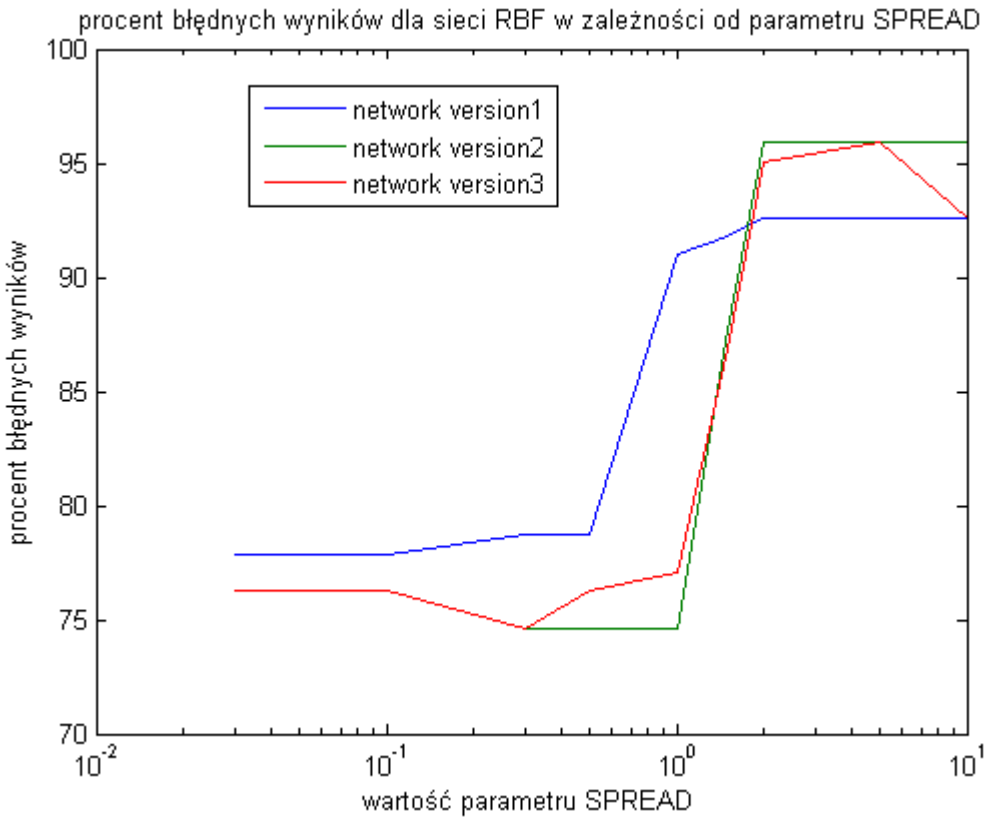
3.2.1 Sieć RBF

Test SPREAD

Dla tego typu sieci, przede wszystkim należało przetestować różne wartości parametru SPREAD.

W tym przypadku zastosowano zestaw 2000 danych uczących dla przyspieszenia obliczeń oraz zestaw 100 danych testowych.

W wyniku symulacji otrzymano następujący wykres przedstawiający zależność błędu sieci od parametru SPREAD.



Jak widać, optymalna wartość parametru SPREAD dla obu typów sieci RBF wynosi około 0.1.

Oprócz tego widzimy nieznaczną przewagę sieci z wariantem konfiguracji warstwy wyjściowej nr 2.

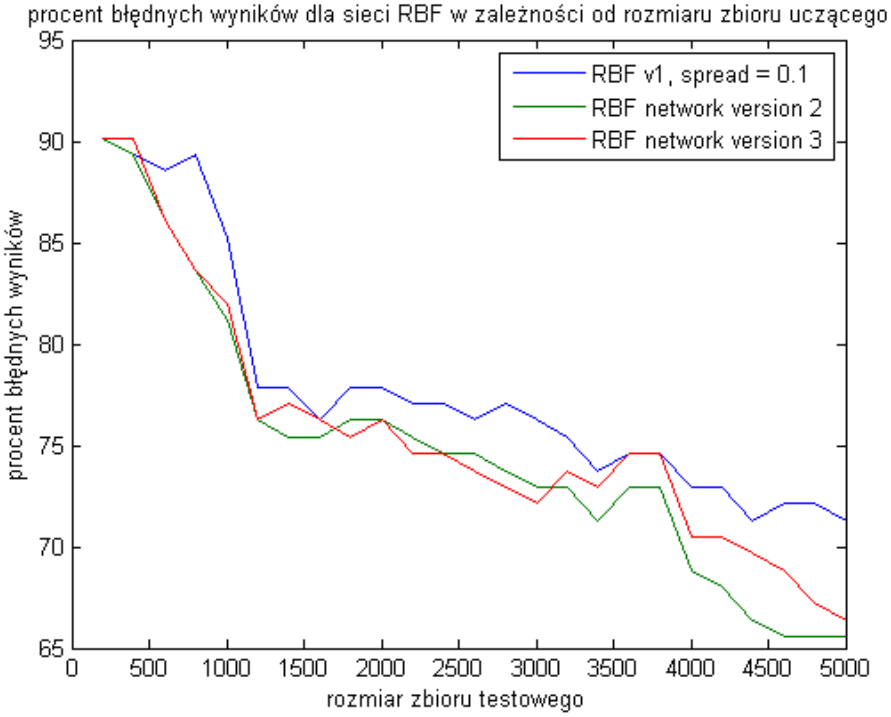
Analizując ten wykres możemy stwierdzić, że zastosowane wyjście w wersji 2 oraz 3 dają podobne rezultaty.

TEST LICZEBNOSCI ZBIORU UCZĄCEGO

W tym kroku zbadano zależność błędu sieci od liczebności zbioru uczącego. Jako parametr SPREAD dla sieci zastosowano wartość równą 0.1.

Test przeprowadzono dla sieci z 3 wariantami warstwy wyjściowej.

W wyniku symulacji otrzymano następujący wykres:



3.2.1.1 Wykres dokładności sieci RBF od rozmiaru zbioru uczącego

Jak widać na podstawie wykresu błąd sieci maleje wraz ze wzrostem liczebności zbioru uczącego. Największą różnicę widać w przedziale 1000 a 1500, gdzie obserwujemy zmniejszenie błędu sieci o około 10%.

W przedziale 1500 – 5000 różnica wynosi około 5%.

Porównując warianty warstwy wyjściowej widzimy znowu, że sieci w wariancie 2 i 3 dają bardzo zbliżone wyniki z lekką przewagą wariantu nr. 2. Wariant nr 1 również w tym teście daje gorsze rezultaty.

Dodatkowo możemy wywnioskować, że kolejne zwiększanie rozmiaru zbioru uczącego przyniesie lepsze rezultaty, aczkolwiek ze względu na ograniczenia sprzętowe nie jesteśmy w stanie znacząco zwiększyć tego zbioru.

3.2.2 Sieć Feedforward (jednokierunkowa wielowarstwowa).

TEST LICZBY NEURONÓW W WARSTWIE UKRYTEJ

W tego typu sieci należało przetestować różne liczby neuronów w warstwie ukrytej.

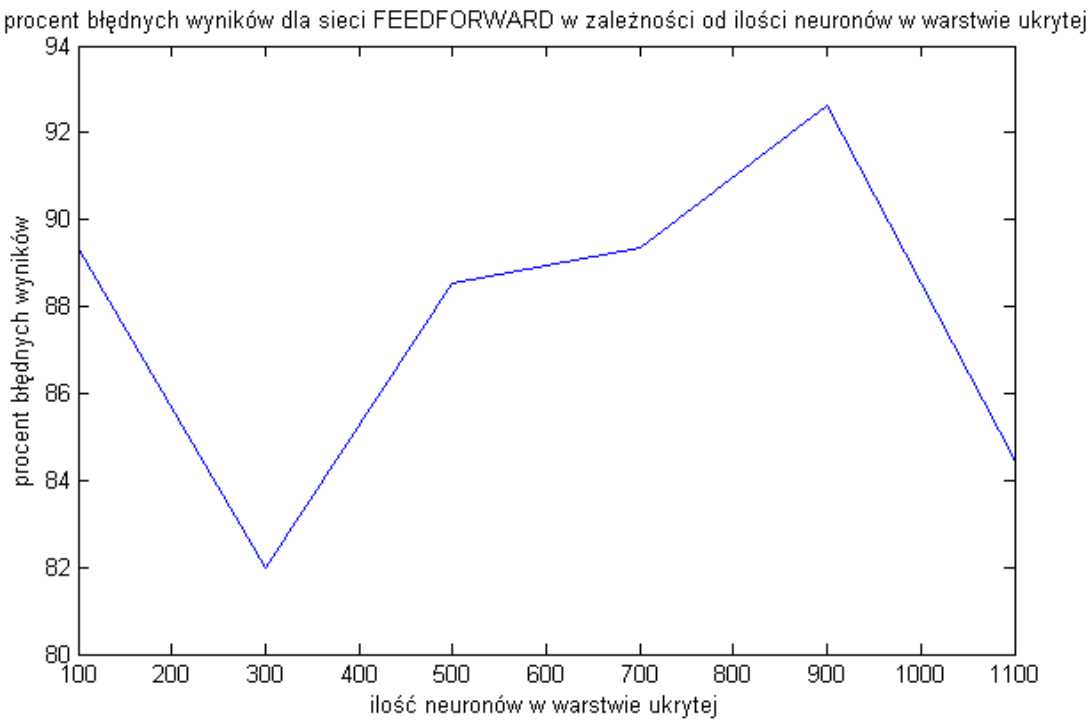
Zastosowano zbiór uczący równy 2000, oraz przetestowano 3 konfiguracje warstwy wyjściowej tak jak w poprzednim przykładzie.

W wyniku symulacji otrzymano następujący wykres:

Jak widać na wykresie, zastosowanie większej liczby neuronów dało pozytywny skutek jedynie dla sieci z konfiguracją warstwy wyjściowej w wersji nr 2.

Różnica w zastosowaniu 50 neuronów a 300 neuronów wynosi około 10%. Mimo wszystko widzimy, że błąd sieci wciąż jest niezadowalający.

Na podstawie wykresu można również zauważyć, iż sieć w wariancie nr 2 daje coraz lepsze rezultaty dla większych ilości neuronów w warstwie ukrytej, dlatego postanowiono przetestować tą sieć z większą liczbą neuronów, co przedstawia z kolei następujący wykres:



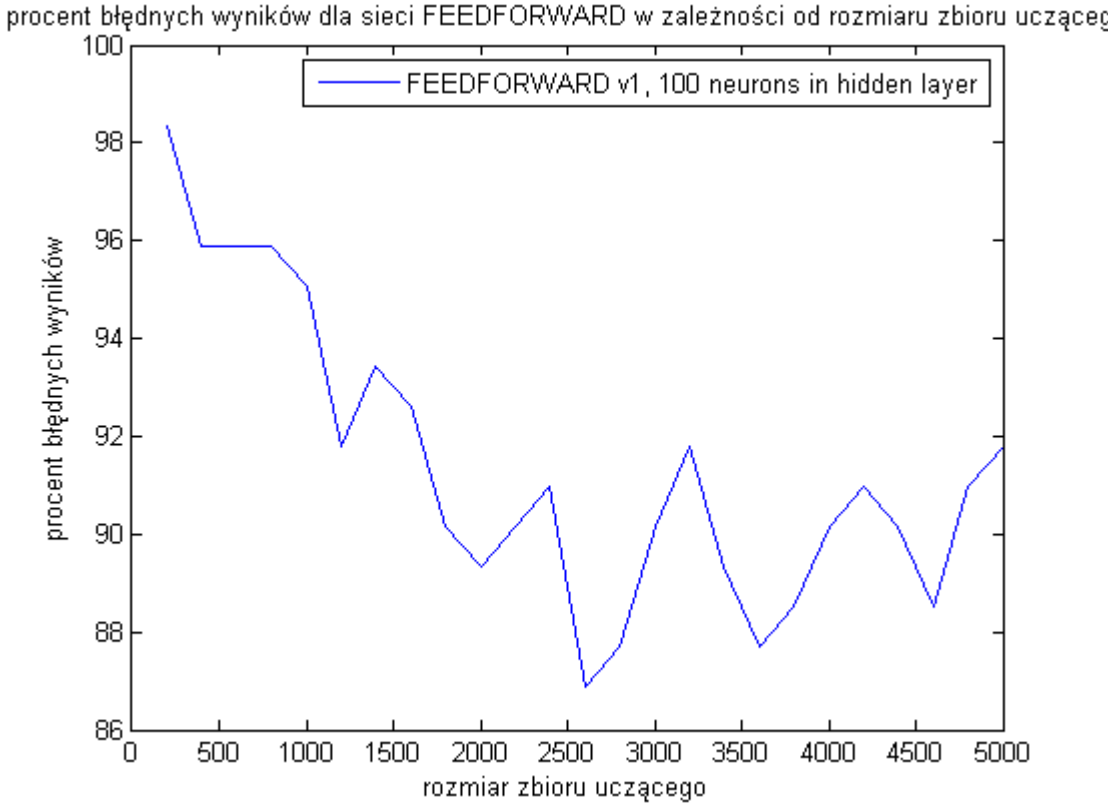
Analizując wykres widzimy jednak, że dokładność sieci nie zwiększa się w nieskończoność wraz z liczbą neuronów w warstwie ukrytej.

Można stwierdzić, że najbardziej optymalna liczba neuronów w warstwie ukrytej wynosi około 300.

Podsumowując test należy dodać, że stosowanie kolejnych warstw ukrytych z różną liczbą neuronów przynosiło jeszcze gorsze efekty.

TEST LICZEBNOSCI ZBIORU UCZĄCEGO.

Dla tego testu zastosowano sieć jednokierunkową z konfiguracją warstwy wyjściowej w wersji nr 2, która dała lepsze rezultaty w poprzednim teście, oraz liczbę neuronów w warstwie ukrytej ustalono na 100 (w celu przyspieszenia obliczeń).



Analizując wykres zależności dokładności sieci od rozmiaru zbioru uczącego można zauważyć, że zwiększanie zbioru daje pozytywny efekt jedynie do rozmiaru około 2500, natomiast dalsze zwiększanie tegoż zbioru o dziwo nie przynosi żadnych korzyści.

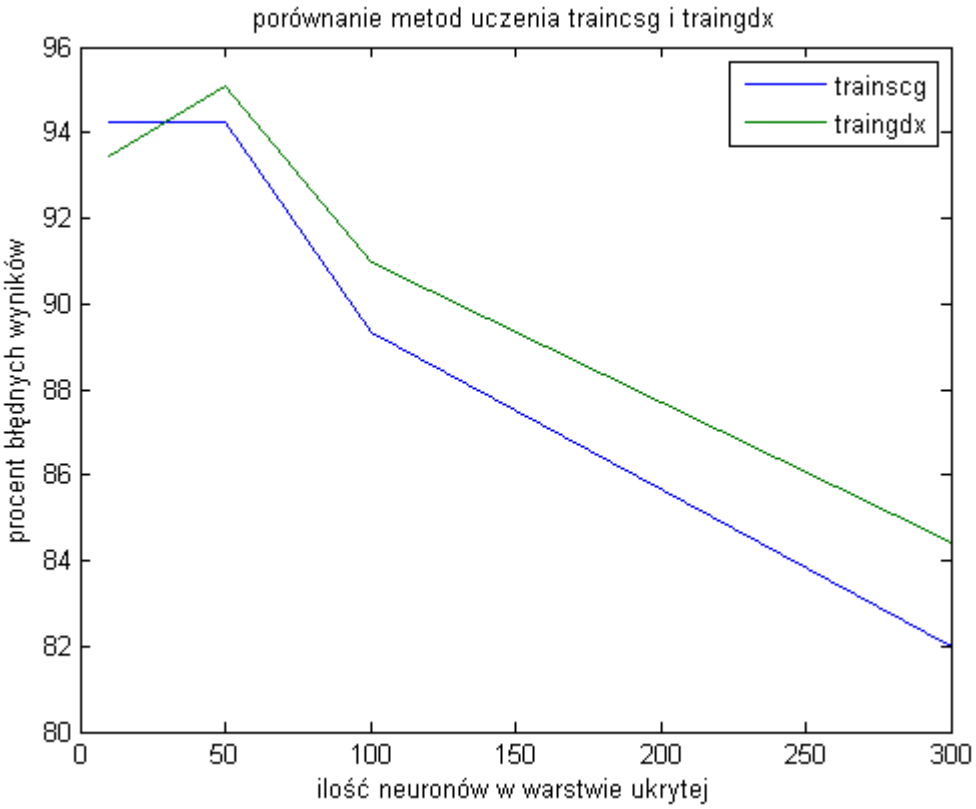
TEST ALGORYTMÓW UCZENIA

W tym kroku porównano 2 metody uczenia, które według literatury dają najlepsze efekty:

- Trainscg - Scaled conjugate gradient backpropagation
- Traingdx - Gradient descent with momentum and adaptive learning rate backpropagation

Dla obu tych metod stworzono sieci oraz przetestowano ich dokładność dla różnych ilości neuronów w warstwie ukrytej.

Wyniki przeprowadzonego testu znajdują się na poniższym rysunku.



Analizując porównanie obu metod można stwierdzić, że nie ma większej różnicy w dokładności. Można przyjąć, że metoda trainscg daje rezultaty nieznacznie lepsze (o około 2%).

3.3 Podsumowanie testów sieci.

Podsumowując przeprowadzone testy dla sieci RBF oraz FeedForward możemy stwierdzić następujące fakty:

- Dla obu sieci lepszy rezultat przyniosło zastosowanie konfiguracji warstwy wyjściowej nr 2.
- Poniższa tabelka przedstawia zestawienie najlepszych wariantów obu sieci pod kątem dokładności

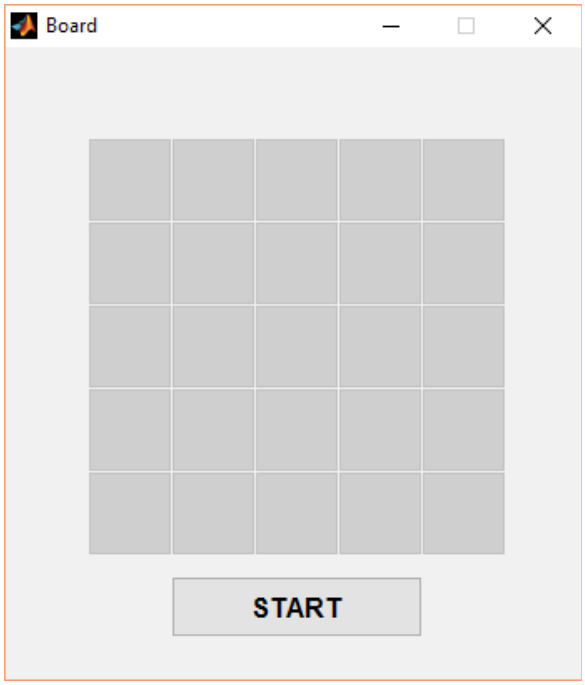
Sieć	**RBF**	**FEEDFORWARD**	**WYBÓR LOSOWY**
% poprawnych odp.	**~ 30%**	**~ 20%**	**4%**

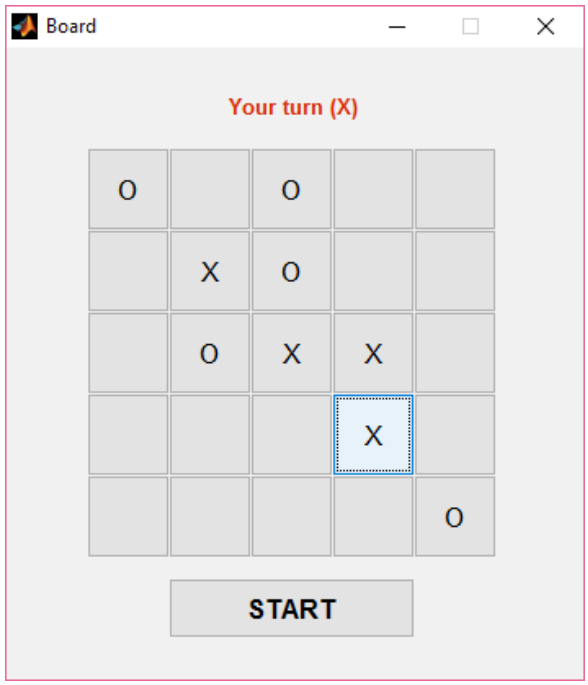
W tabelce zawarto również wynik dla losowych wyborów. Dla planszy 5x5 prawdopodobieństwo wyboru właściwego pola wynosi 1/25 co daje nam 4%.

Oczywiście otrzymane rezultaty nie są zbytnio zadowalające.

Jak widzimy najlepszą dokładność możemy uzyskać poprzez zastosowanie sieci RBF. Dokładność dla tej sieci możemy nieco zwiększyć poprzez zwiększenie zbiory uczącego. Analizując otrzymany wykres (3.2.1.1) możemy domniemywać iż zwiększenie zbioru uczącego do 20 000 mogłoby dać w rezultacie dokładność około **40%**, aczkolwiek trzeba mieć świadomość ograniczeń wynikających z długiego czasu generowania takiej liczby danych oraz ograniczenia pamięci komputera.

4 GUI





Interfejs graficzny jest bardzo prosty w użyciu co usprawnia użytkownikowi grę.

Jak widzimy na powyższych obrazkach, gra rozpoczyna się dopiero po naciśnięciu przycisku START, inaczej klikanie po tabeli nie daje żadnych rezultatów. Nie jest to jedyny przycisk w oknie. Cała tabela do gry złożona jest z przycisków. Dodatkowo, komunikat wyświetlany na czerwono sugeruje nam nasz ruch, bądź ruch komputera a także informuje o wygranej.

5 Wnioski

W wyniku projektu została napisana w Matlabie aplikacja umożliwiająca grę z komputerem. Niestety z powodu niedostatecznej mocy obliczeniowej poziom trudności nie jest zbyt wysoki. Według przeprowadzonych badan nad skutecznością sieci można wnioskować że wygenerowanie odpowiednio dużego zbioru danych jest kluczowe do nauczenia sieci. Jest to dosyć logiczne, biorąc pod uwagę jak wiele różnych kombinacji planszy i możliwych ruchów można wygenerować.