Sprawozdanie lab 2

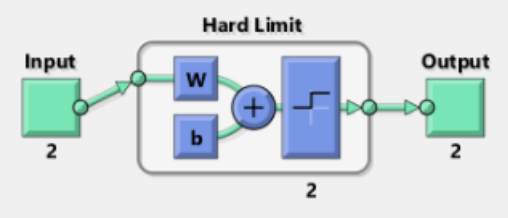
Metody inteligencji obliczeniowej – Informatyka Stosowana, WFIIS,
Jakub Salamon, II rok

Celem zajęć laboratoryjnych nr 2 była klasyfikacja danych przy pomocy neuronów liniowych

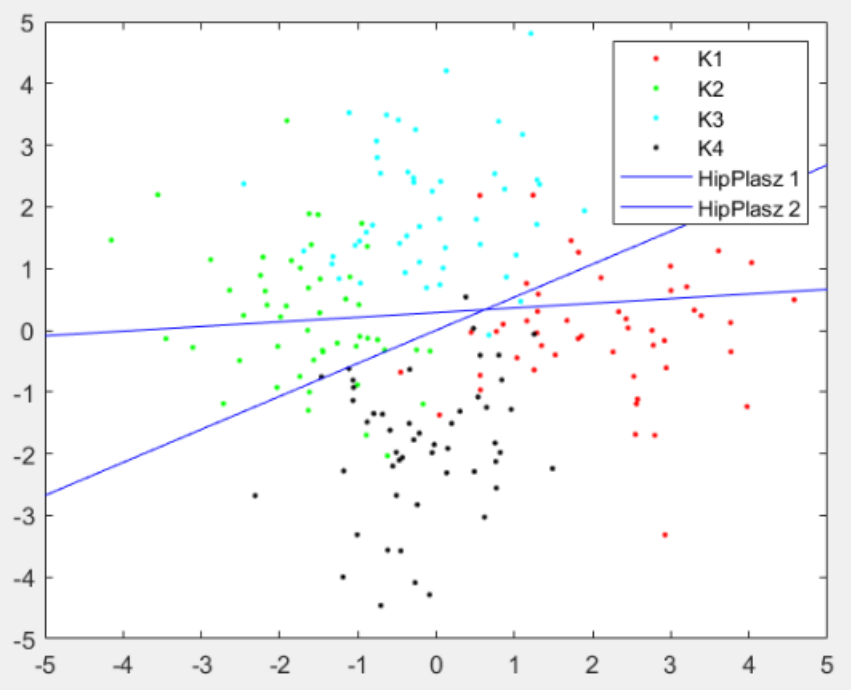
Dane losowe

Wygenerowaliśmy dane uczące i testujące z rozkładów N([2, 0], 1), N([-2, 0], 1), N([0, 2], 1) oraz N([0, -2], 1). Układ neuronowy oparty był na skokowej funkcji aktywacji. Dane wyjściowe były zakodowane w macierzy 2x(ilość elementów). Przynależność kolejno do klas 1, 2, 3, 4 była kodowana [0, 1; 1, 0; 0, 0; 1, 1] . Pozwala to na właściwe oddzielenie klas od siebie i otrzymanie dobrych wyników.

Schemat takiej sieci



Próba ucząca liczyła po 50 elementów, a testowa po 1000 dla każdej z klas.

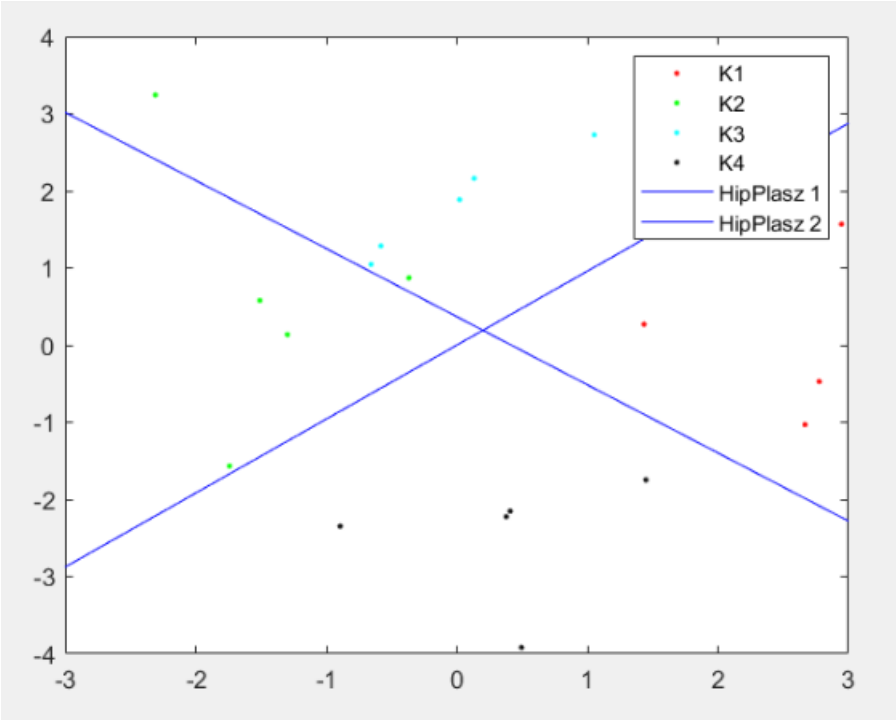


Hiperpłaszczyzna 1 y = 0.075257x + 0.28648
Hiperpłaszczyzna 2 y = 0.53561x + 0
Wydajność = 61.35%
Błąd całkowity = 38.65%

Widzimy, że hiperpłaszczyzny nie oddzielają danych od siebie idealnie. Hiperpłaszczyzna 2 dużo lepiej spełnia swoją rolę niż hiperpłaszczyzna 1. Jak wiemy wzory funkcji hiperpłaszczyzn są zależne od wag wygenerowanych przez sieć neuronową. Oznacza to, że sieć źle oddziela od siebie klasy kodowane pierwsza cyfrą 0 oraz 1.

Teraz zmieniamy wielkość próby uczącej i sprawdzamy działanie sieci.

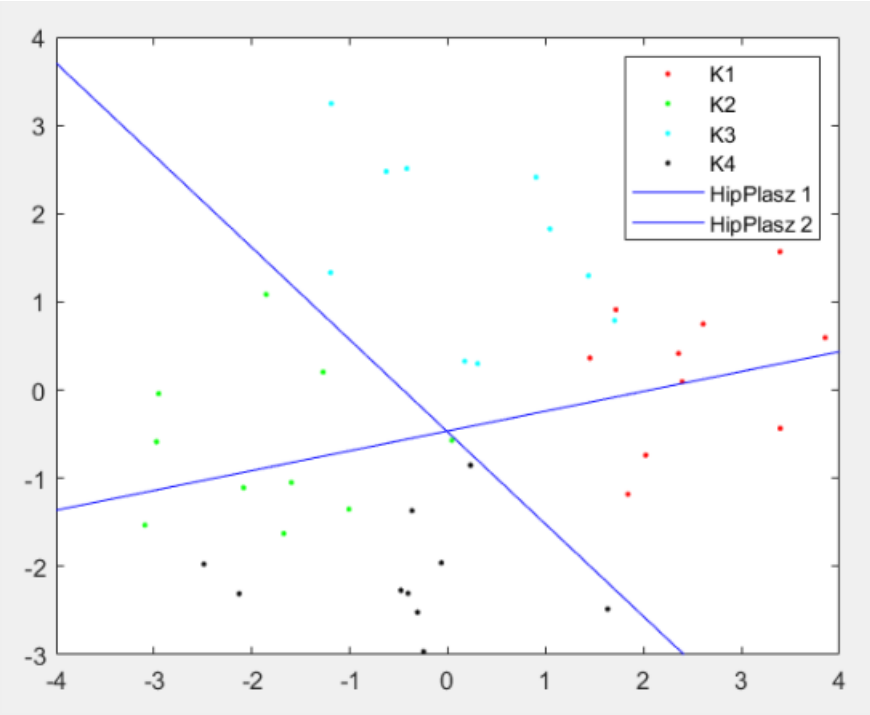
Próba ucząca 5 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -0.88312x + 0.36923
Hiperpłaszczyzna 2 y = 0.95892x + 0
Wydajność = 83.2%
Błąd całkowity = 16.8%

W tym przypadku otrzymaliśmy lepsze dopasowanie hiperpłaszczyzn. Również na podstawie wydajności możemy stwierdzić, że sieć działa dużo lepiej niż wcześniej.

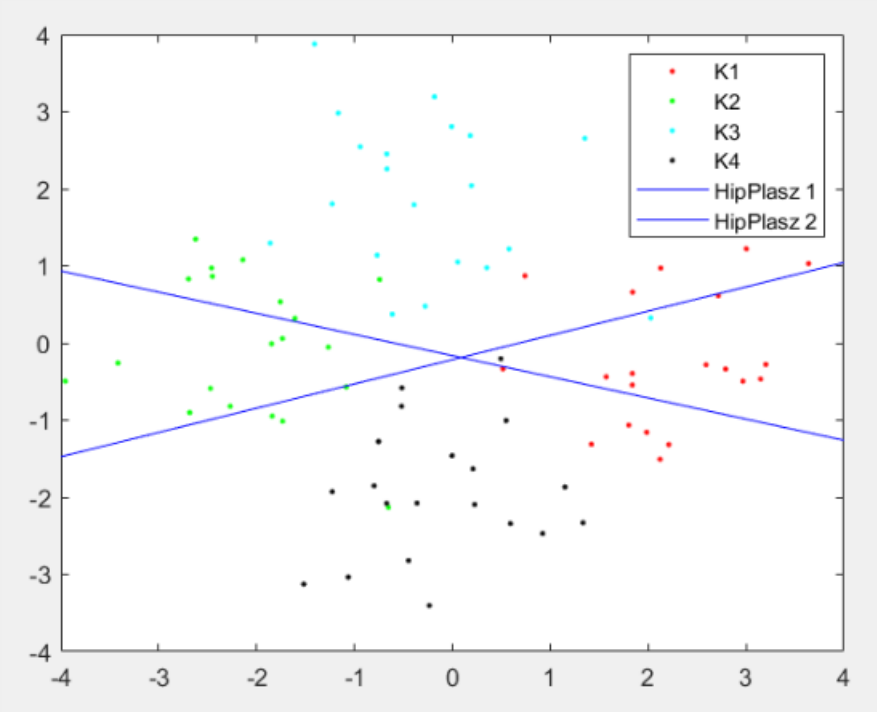
Próba ucząca 10 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -1.0455x -0.47703
Hiperpłaszczyzna 2 y = 0.22473x -0.46427
Wydajność = 72.125%
Błąd całkowity = 27.875%

Sieć nie zawsze dokładnie oddziela klasy kodowane drugą cyfrą jako 0 oraz 1. Obserwujemy to na wykresie na podstawie hiperpłaszczyzny 2.

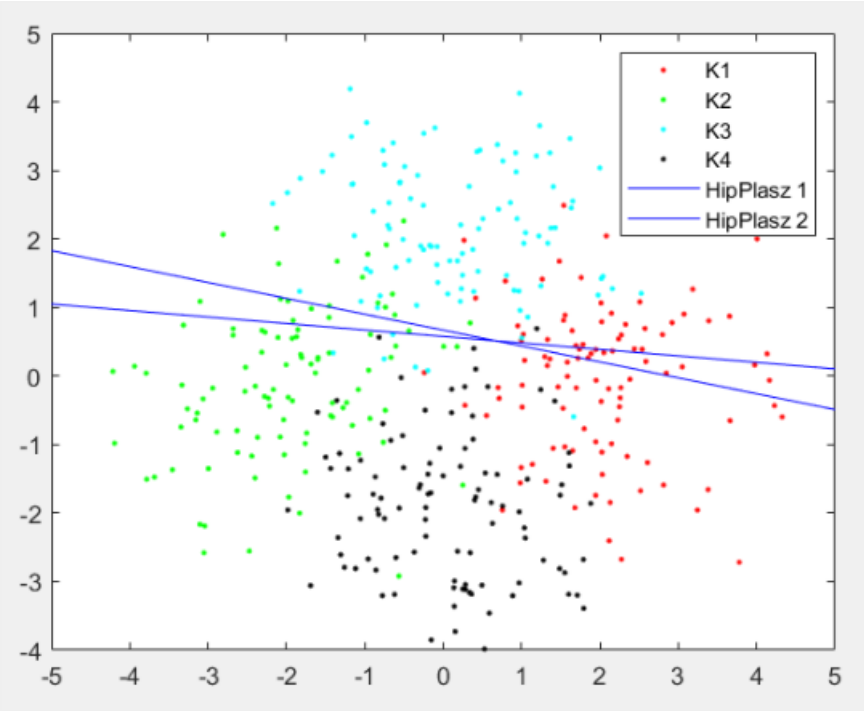
Próba ucząca 20 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -0.27437x -0.16305
Hiperpłaszczyzna 2 y = 0.31459x -0.2153
Wydajność = 68.475%
Błąd całkowity = 31.525%

W tym przypadku żadna z hiperpłaszczyzn nie jest dobrze dopasowana. Również wydajność się zmniejszyła.

Próba ucząca 100 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -0.094575x + 0.57715
Hiperpłaszczyzna 2 y = -0.23198x + 0.66923
Wydajność = 47.825%
Błąd całkowity = 52.175%

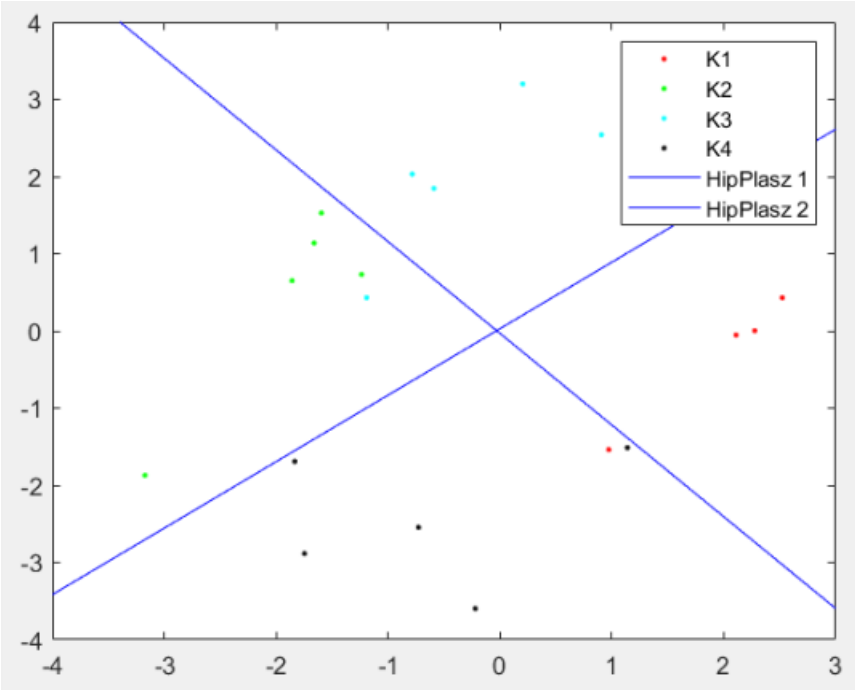
Dla 100 elementów uczących wydajność jest mniejsza niż 50%. Oznacza to, że lepiej odwrócić dane wyjściowe i te błędne uznać za właściwe. Algorytm nie działa dobrze dla takiej ilości danych i nie spełnia swojej roli.

Podsumowując algorytm zmniejsza swoją efektywność wraz we wzrostem ilości danych wejściowych jako danych uczących. Następuje przeuczenie przez co rośnie ilość błędów, które popełnia. Wagi są źle dopasowywane, co obserwujemy patrząc na hiperpłaszczyzny na wykresach.

Dane rzeczywiste

Teraz będziemy korzystać z metody uczenia za pomocą reguły Widrowa-Hoffa oraz neuronów liniowych. Wymaga to zastosowania innego kodowania macierzy wyjścia. Kodujemy je za pomocą -1 oraz 1. Dane generowane są w taki sam sposób jak poprzednio.

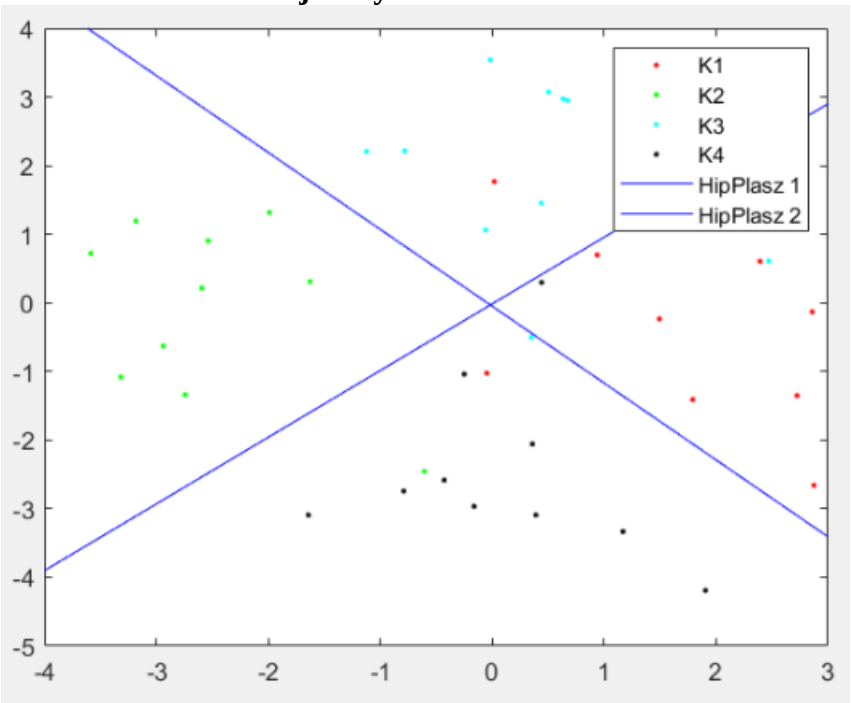
Próba ucząca 5 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -1.188x -0.031095
Hiperpłaszczyzna 2 y = 0.86141x + 0.029095
Wydajność = 84.925%
Błąd całkowity = 15.075%

Płaszczyzny są wyznaczone dobrze, również wydajność jest dobra.

Próba ucząca 10 elementów dla każdej klasy

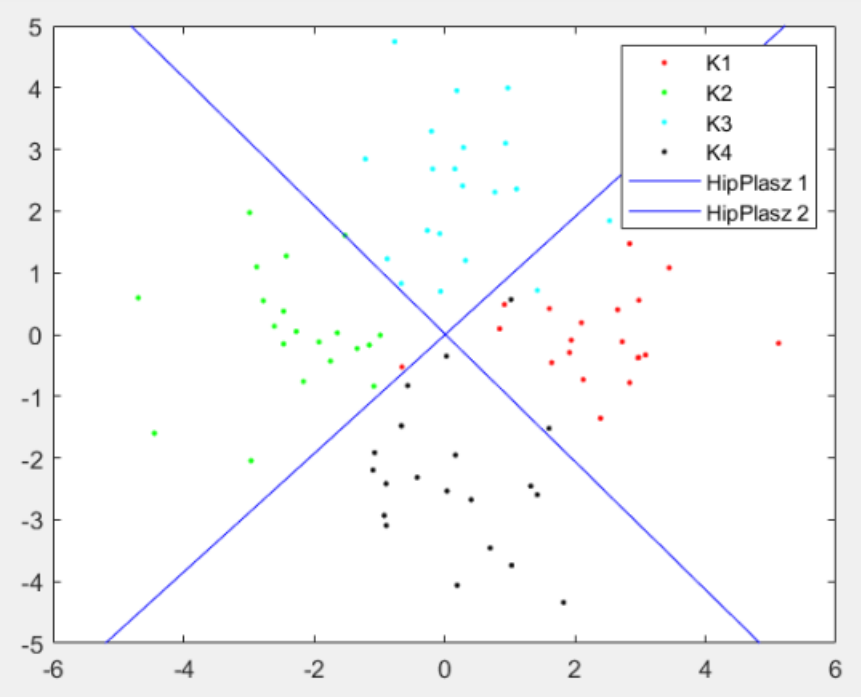


Hiperpłaszczyzna 1 y = -1.1204x -0.049079
Hiperpłaszczyzna 2 y = 0.97215x -0.014763

Wydajność = 84.675%
Błąd całkowity = 15.325%

Pomimo zastosowania większej populacji uczącej sieć nie zmieniła swojej wydajności.

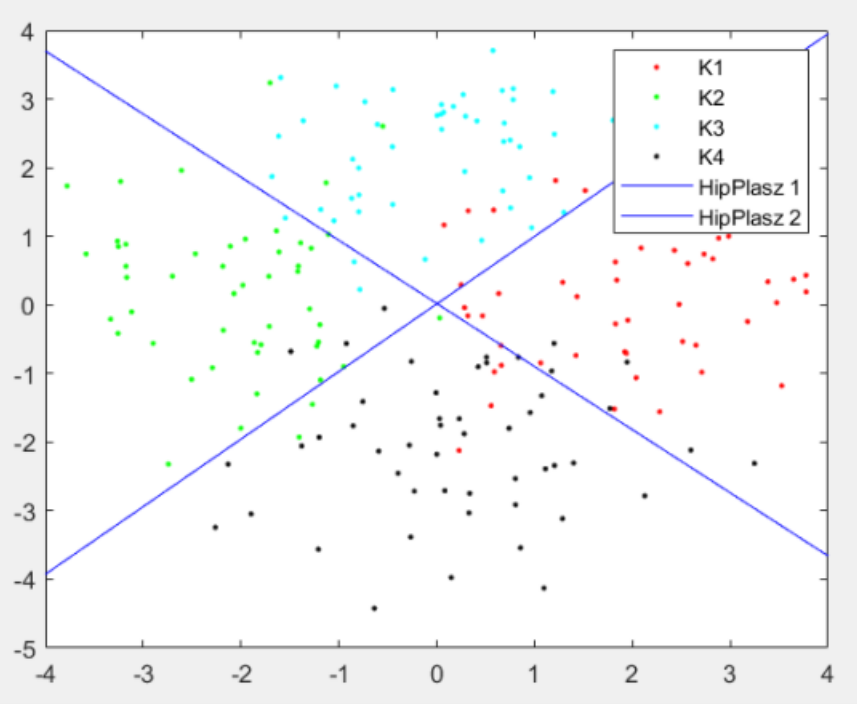
Próba ucząca 20 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -1.0379x + 0.018974
Hiperpłaszczyzna 2 y = 0.9602x -0.012404
Wydajność = 84.5%
Błąd całkowity = 15.5%

Liczba danych uczących dalej rośnie, ale wydajność nie zmniejsza się.

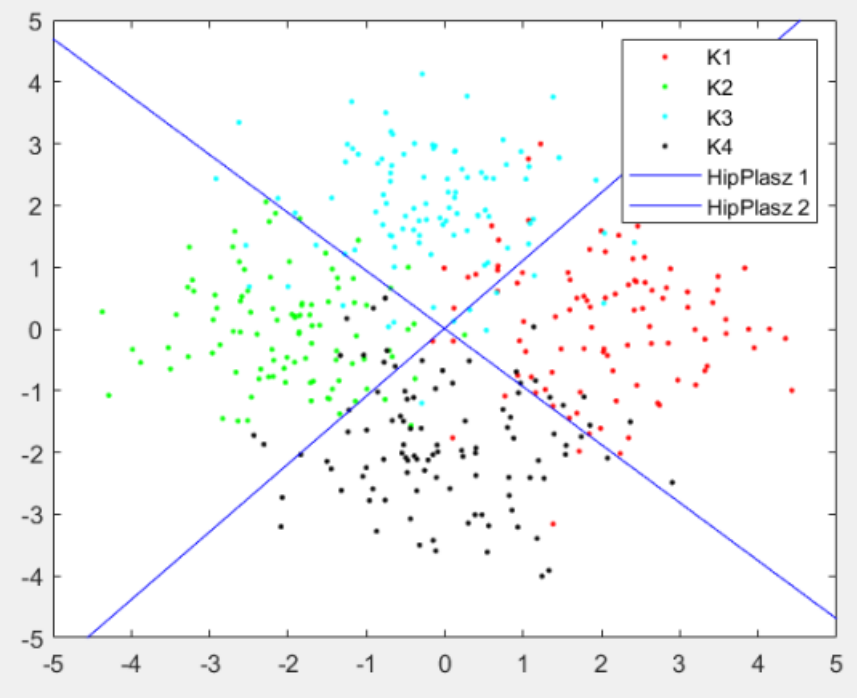
Próba ucząca 50 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -0.91984x + 0.02123
Hiperpłaszczyzna 2 y = 0.98452x + 0.01294
Wydajność = 83.975%
Błąd całkowity = 16.025%

Wydajność jest dalej bardzo zbliżona do poprzednich.

Próba ucząca 100 elementów dla każdej klasy



Hiperpłaszczyzna 1 y = -0.93944x + 0.0025168

Hiperpłaszczyzna 2 y = 1.0987x + 0.011321
Wydajność = 84.85%
Błąd całkowity = 15.15%

W tym przypadku nie występuje zmniejszanie wydajności sieci wraz ze wzrostem ilości danych uczących.

Porównanie metody z ćwiczenia pierwszego do metody z ćwiczenia drugiego

Próba ucząca (dla klasy)	Sieć oparta o perceptron		Neurony liniowe z metodą Widrow-Hoffa	
	Wydajność	Błąd całkowity	Wydajność	Błąd całkowity
5	83.2%	16.8%	84.925%	15.075%
10	72.125%	27.875%	84.675%	15.325%
20	68.475%	31.525%	84.5%	15.5%
50	61.35%	38.65%	83.975%	16.025%
100	47.825%	52.125%	84.85%	15.15%

Z otrzymanych wyników można jasno wywnioskować, że sieci z neuronami liniowymi z metodą uczenia reguła Widrowa-Hoffa są lepsze dla tego problemu. Jednak warto się zastanowić czy ich potrzebujemy, ponieważ dla 5 elementów dla każdej z klas uczących otrzymujemy bardzo zbliżoną wydajność, więc okazuję się, że nie ważne którą z metod wybierzemy otrzymamy satysfakcjonujące nas rozwiązanie.

Dane rzeczywiste – irysy

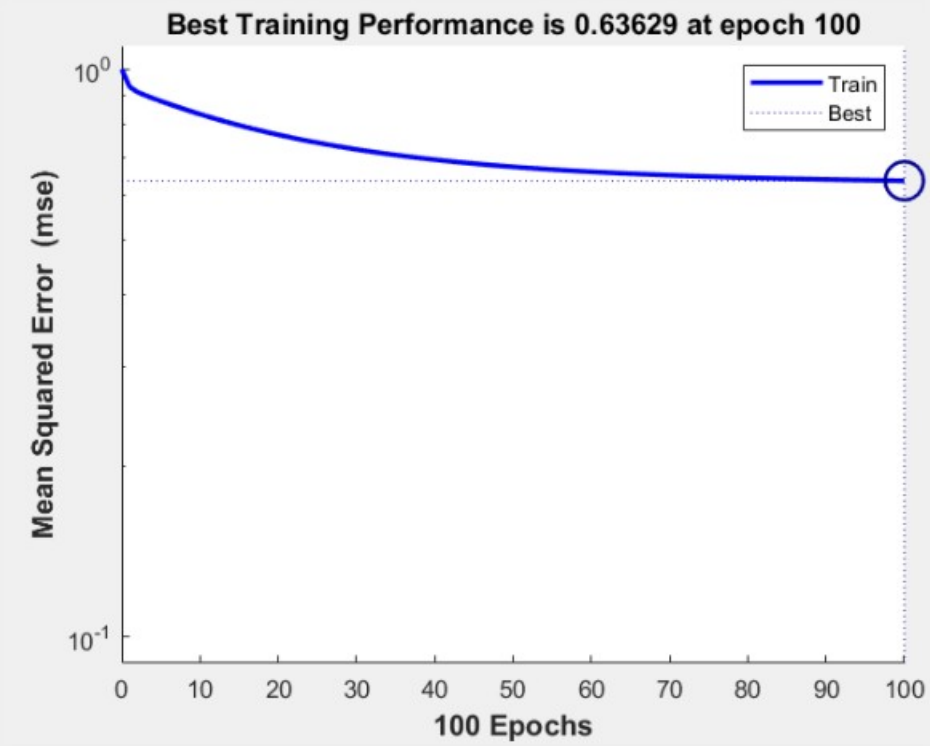
Dane na temat kwiatów irysa są dostępne w programie MATLAB, jednak wymagają odpowiedniego wybrania spośród nich danych uczących i testujących. Użyłem do tego metody walidacji krzyżowej, a konkretnie K-Fold Cross Validation. Dzięki temu dane były dzielone bardzo dobrze zawsze na 135 elementów uczących i 15 testujących. Wydajność była różna w kilku uruchomieniach algorytmu. Liczba epok wynosiła 100 i nie zmieniała się.

Uruchomienie	1	2	3	4	5	Średnia wydajność
Wydajność neurony dyskretne	53.3%	66.7%	66.7%	66.7%	66.7%	64%
Wydajność neurony liniowe	66.7%	66.7%	66.7%	66.7%	66.7%	66.7%

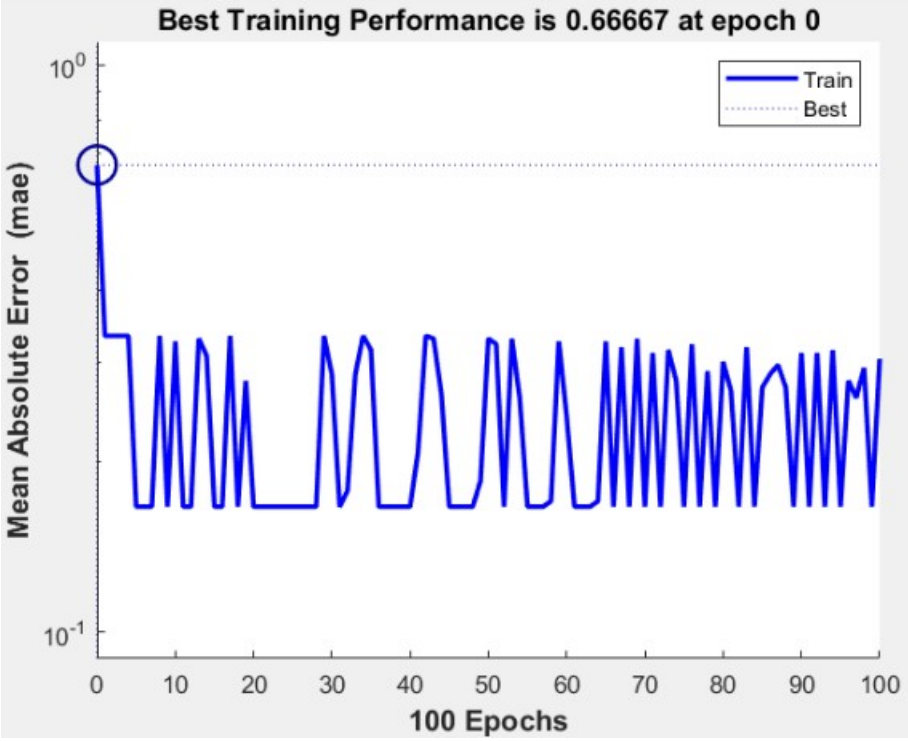
Warto zauważyć, że sieci są bardzo wrażliwe na macierz danych wyjściowych. Przykładowo zmiana z macierzy [0, 0; 0, 1; 1, 0] na macierz [0, 1; 1, 0; 0, 0] spowodowała zmniejszenie

wydajności do 48% w przypadku neuronów dyskretnych, a dla neuronów liniowych wynik analogicznej zmiany przyniósł jeszcze gorsze rezultaty, bo tylko 33.3% wydajności.

Neurony liniowe nie są podatne na duże odchylenia rozwiązań, łagodnie zmierzają do właściwego rozwiązania.



Natomiast neurony dyskretne już tak.



Podsumowując implementując sieć neuronową warto zwrócić uwagę na kodowanie macierzy wyniku, bo ma ona bardzo duży wpływ na jakość naszego rozwiązania. Warto też przeanalizować jak dużo elementów uczących musimy użyć oraz jaki typ neuronów zastosować.

Jakub Salamon, 297914
Informatyka Stosowana, WFIIS, II rok