

# Sztuczne sieci neuronowe i ich zastosowania

Marcin Pluciński  
[mplucinski@wi.zut.edu.pl](mailto:mplucinski@wi.zut.edu.pl)

Zakład Metod Sztucznej Inteligencji  
Katedra Metod Sztucznej Inteligencji i Matematyki Stosowanej

Wydział Informatyki  
Zachodniopomorski Uniwersytet Technologiczny

- Stanisław Osowski:  
**Sieci neuronowe w ujęciu algorytmicznym**
- Józef Korbicz, Andrzej Obuchowicz, Dariusz Uciński:  
**Sztuczne sieci neuronowe, podstawy i zastosowania**
- Timothy Masters:  
**Sieci neuronowe w praktyce (programowanie w C++)**

# Historia sieci neuronowych

- 1943: McCulloch i Pitts – model sztucznego neuronu.
- 1949: Hebb – mechanizm pamiętania informacji przez neurony biologiczne.
- 1958: Rosenblatt – konstrukcja perceptronu.
- 1960: Widrow – sieć MADALINE.
- 1968: Minsky i Papert – głos krytyczny.

# Neuron biologiczny

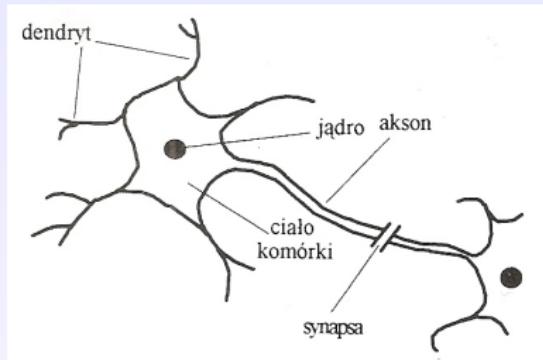
## Neuron

Jest podstawowym budulcem układu nerwowego. Jest komórką, która jest w stanie odbierać i przekazywać sygnały elektryczne.

## Działanie

Jeżeli wartość sygnału elektrycznego przekazywana do neuronu przekracza pewną wartość progową, neuron ulega depolaryzacji, powodując pobudzenie komórki nerwowej. Pobudzenie polega na wyładowaniu neuronu, a powstały sygnał przesyłany jest do innych neuronów. Siła wyładowania neuronu jest niezależna od bieżącej depolaryzacji, siły, która go spowodowała. Dopóki prąd przekracza określony próg, wielkość wyładowania pozostaje taka sama (w danym neuronie).

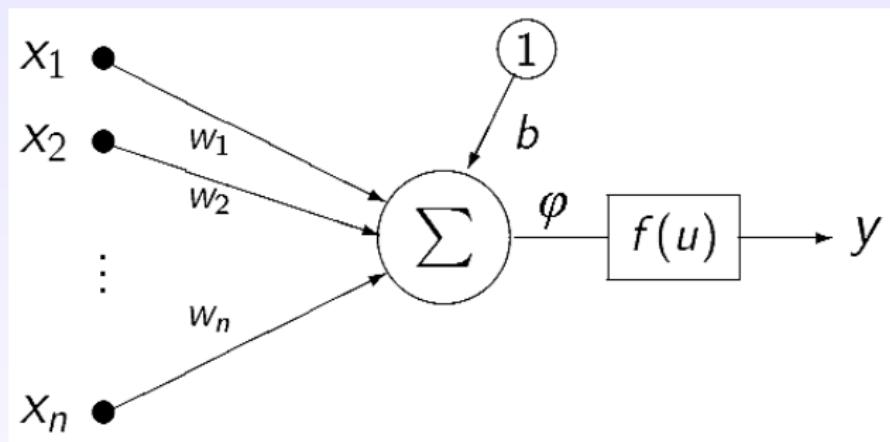
# Neuron biologiczny



Neuron składa się z następujących elementów.

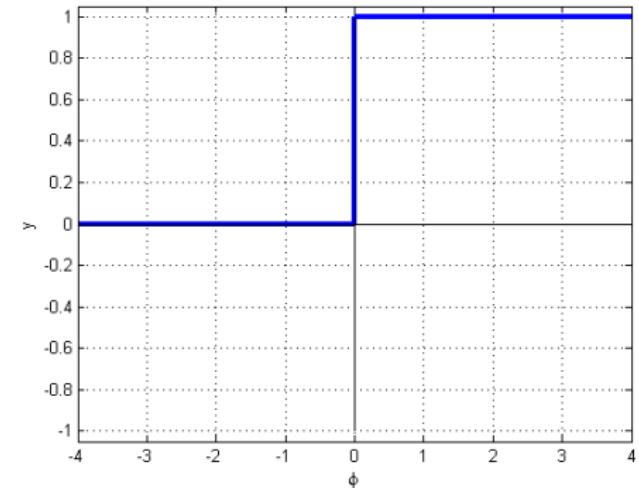
- 1 Wielu dendrytów, których celem jest pobieranie impulsów od innych neuronów.
- 2 Ciała komórk z jądrem.
- 3 Jednego aksonu, który przekazuje wyładowanie do kolejnych komórek.
- 4 Synaps – neuroprzekaźników osłabiających lub wzmacniających sygnał wyjściowy.

# Neuron sztuczny



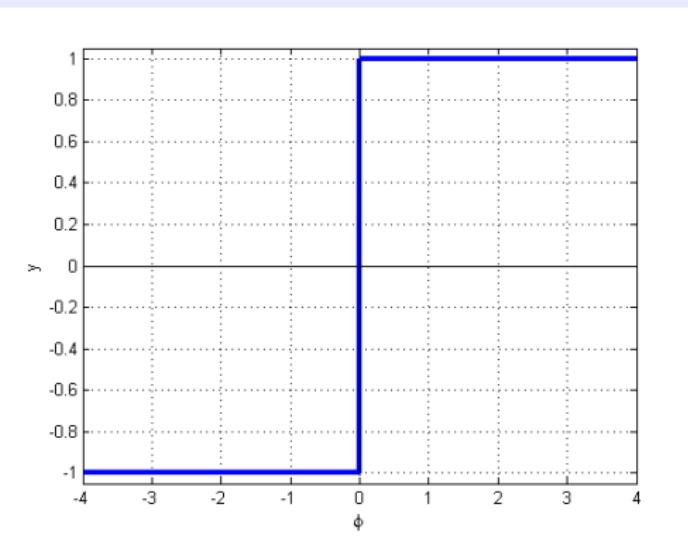
$$y(\mathbf{x}) = f\left(\sum_{i=1}^n w_i x_i + b\right) = f(\mathbf{w}^T \mathbf{x} + b)$$

# Funkcje aktywacji – progowa unipolarna



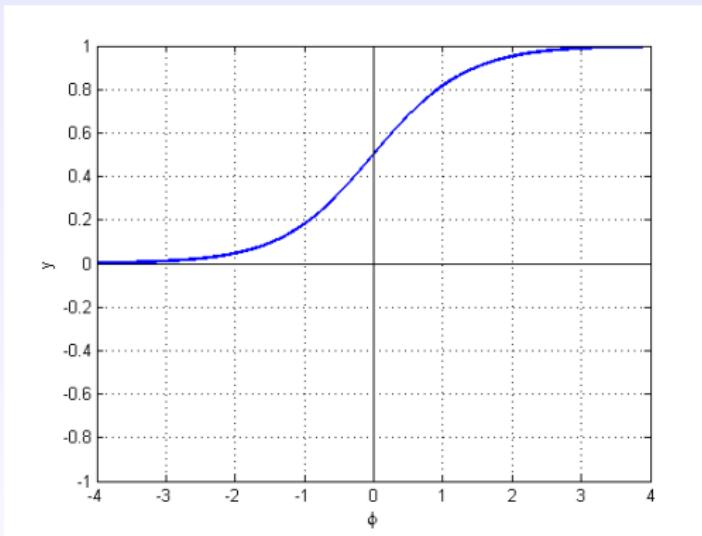
$$f(\phi) = \begin{cases} 1, & \text{dla } \phi > 0 \\ 0, & \text{dla } \phi \leq 0 \end{cases}$$

# Funkcje aktywacji – progowa bipolarna



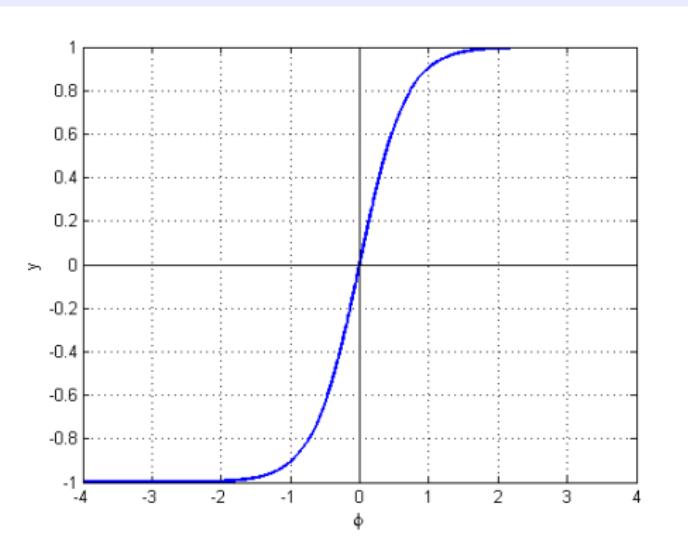
$$f(\phi) = \begin{cases} 1, & \text{dla } \phi > 0 \\ -1, & \text{dla } \phi \leq 0 \end{cases}$$

# Funkcje aktywacji – sigmoidalna



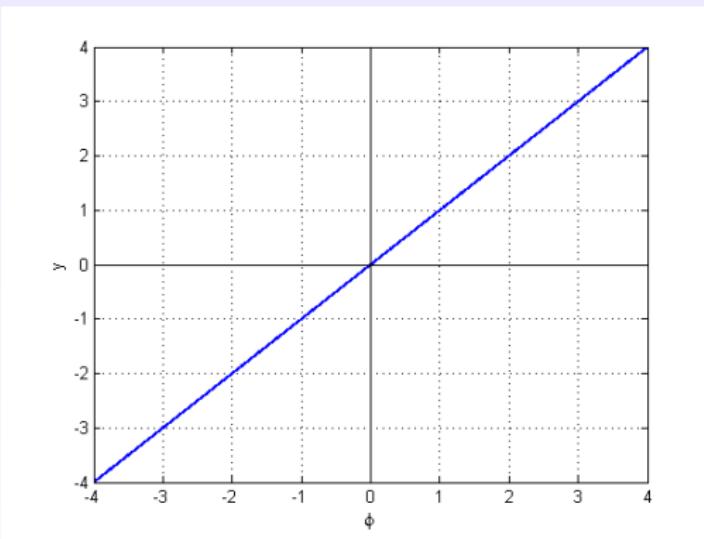
$$f(\phi) = \frac{1}{1 + e^{-\beta\phi}}$$

# Funkcje aktywacji – tangens hiperboliczny



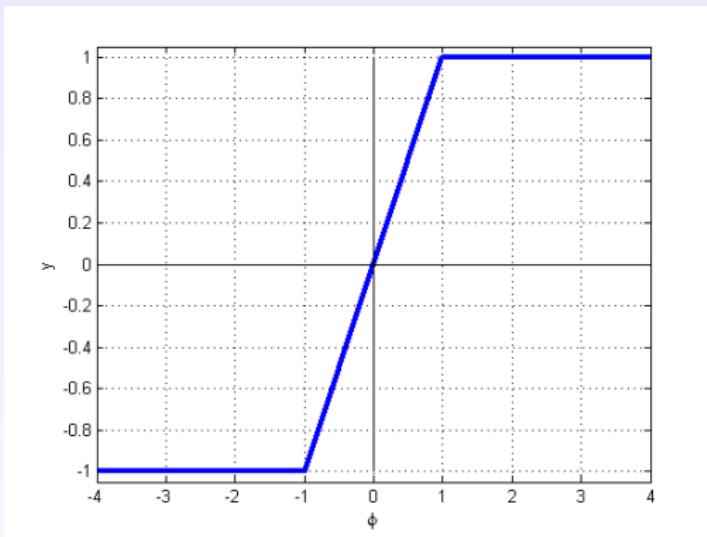
$$f(\phi) = \tanh(\phi) = \frac{e^{\beta\phi} - e^{-\beta\phi}}{e^{\beta\phi} + e^{-\beta\phi}}$$

# Funkcje aktywacji – liniowa



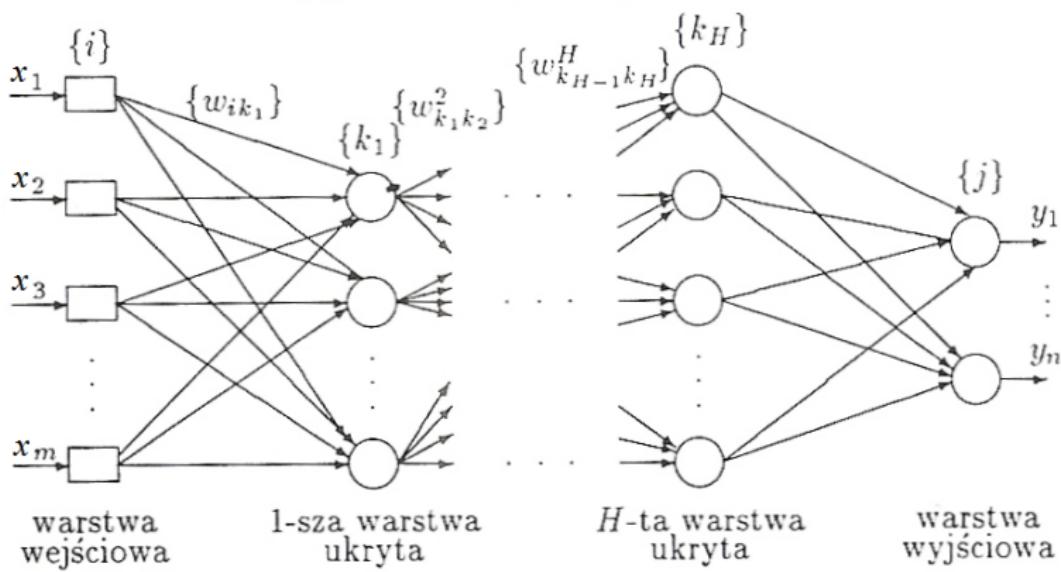
$$f(\phi) = \phi$$

# Funkcje aktywacji – liniowa z nasyceniem



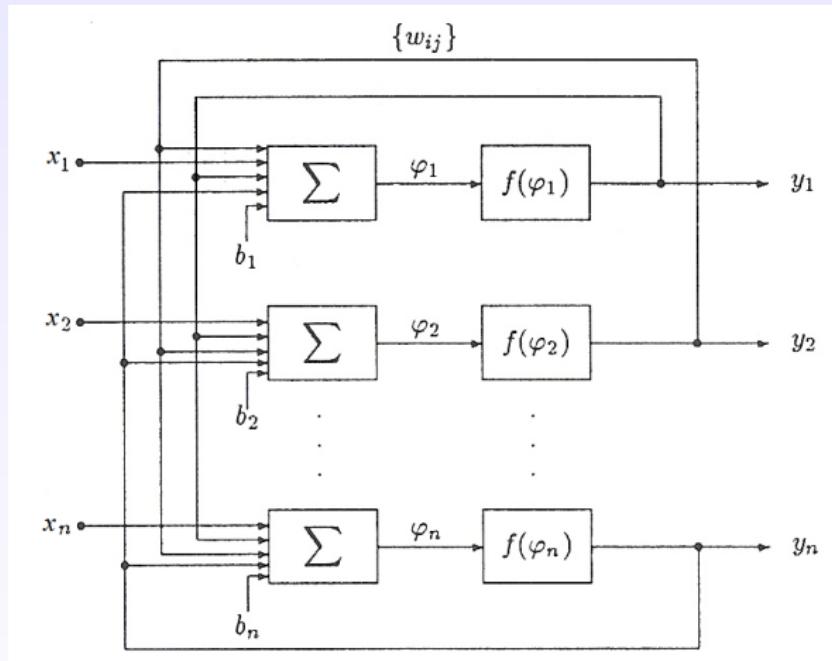
$$f(\phi) = \begin{cases} -1, & \text{dla } \phi < -1 \\ \phi, & \text{dla } -1 \leq \phi < 1 \\ 1, & \text{dla } \phi \geq 1 \end{cases}$$

# Typy sieci neuronowych



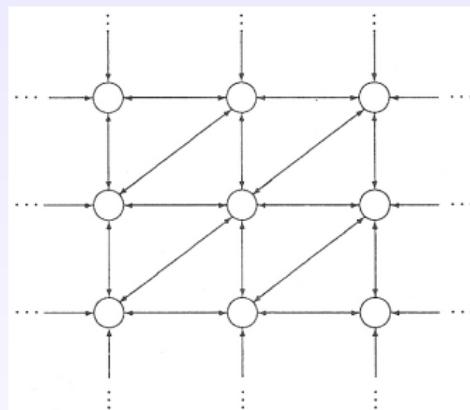
Jednokierunkowa, wielowarstwowa siec neuronowa.

# Typy sieci neuronowych



Rekurencyjna siec neuronowa.

# Typy sieci neuronowych



Komórkowa sieć neuronowa.

# Zastosowania

Sieci neuronowe mogą realizować następujące zadania:

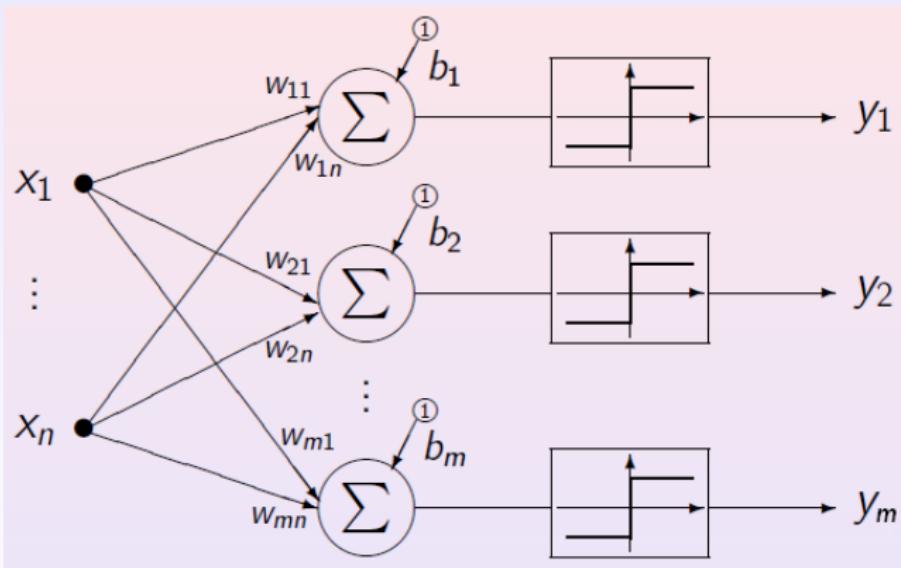
- 1 klasyfikacja,
- 2 aproksymacja (modelowanie zjawisk),
- 3 predykcja,
- 4 filtracja sygnałów,
- 5 analiza danych (np. wykrywanie skupisk, PCA),
- 6 optymalizacja,
- 7 inne (np. kompresja obrazów).

# Perceptron prosty

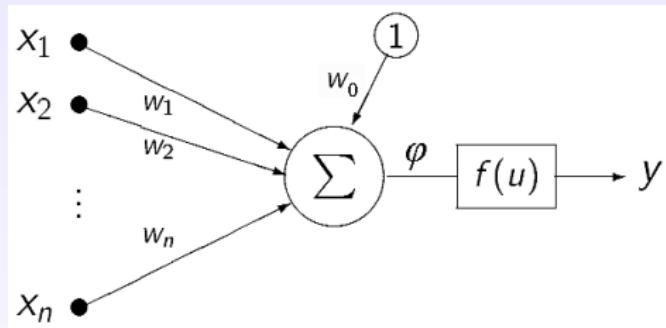
## Informacje ogólne

- Pojedyncza warstwa neuronów
- Bipolarna lub unipolarna progowa funkcja aktywacji
- Sieć uczona pod nadzorem
- Algorytm uczenia – reguła delty
- Zastosowanie – klasyfikacja

# Perceptron prosty



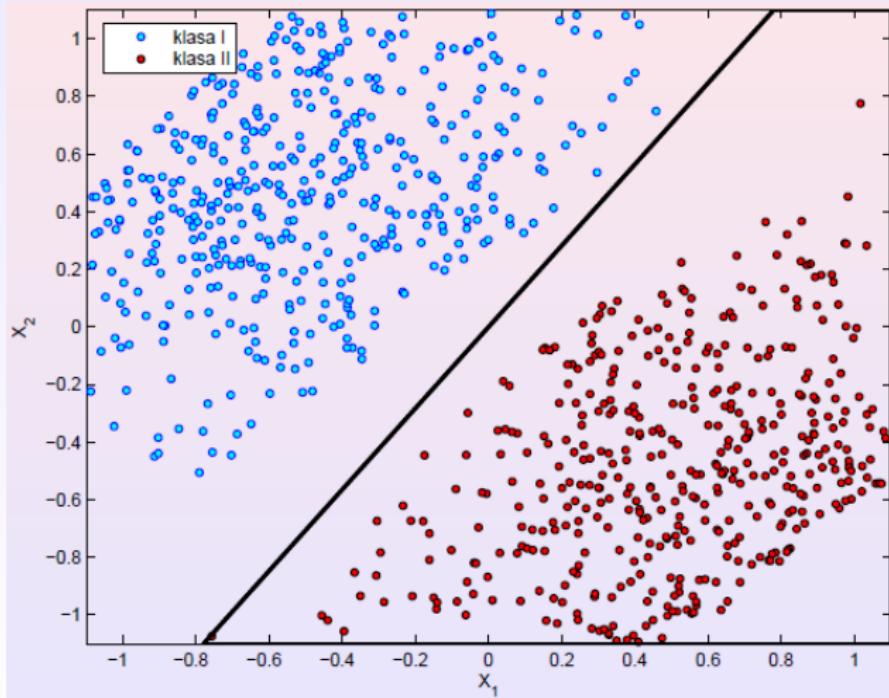
# Perceptron prosty



$$y(\mathbf{x}) = f\left(\sum_{i=0}^n w_i x_i\right) = f(\mathbf{w}^T \mathbf{x})$$

# Perceptron prosty

Pojedynczy neuron separuje przestrzeń wejścia na dwie części.  
Jego podstawowe zadanie to klasyfikacja binarna.



# Perceptron prosty

## Klasyfikacja binarna

Klasyfikacja polega na przydzieleniu obiektu do pewnej klasy na podstawie jego atrybutów (danych wejściowych). W klasyfikacji binarnej wyjście jest dwuwartościowe (możliwe są dwie klasy na wyjściu).

Niech dany będzie zbiór uczący (zbiór par):

$$(\mathbf{x}_i, d_i), \quad i = 1 \dots L,$$

gdzie  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in} \in \mathbb{R}^n)$  są danymi wejściowymi a  $d_i \in \{0, 1\}$  są skojarzonymi z nimi klasami (wyjściami).

# Perceptron prosty

Równanie hiper-płaszczyzny w przestrzeni  $\mathbb{R}^n$ :

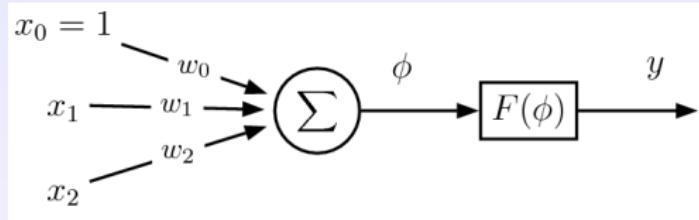
$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n = 0$$

$$\mathbf{w}^T \mathbf{x} = 0$$

Uczenie neuronu polega na takim doborze współczynników wagowych, aby spełnione było:

- $\mathbf{w}^T \mathbf{x} > 0$  dla próbek, których wyjście zadane  $d$  jest równe 1,
- $\mathbf{w}^T \mathbf{x} \leq 0$  dla próbek, których wyjście zadane  $d$  jest równe 0 lub  $-1$ .

# Perceptron prosty



Przykładowo dla neuronu z dwoma wejściami:

$$\phi = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$$

Prosta separująca opisana jest równaniem:

$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0$$

lub:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

## Cel uczenia

- Sieć uczymy na podstawie danych uczących.
- Dane uczące – reprezentują informację o pożądanym zachowaniu sieci.
- Cel uczenia – taki dobór współczynników wagowych, aby sieć realizowała odpowiednie zadanie.

# Metody uczenia sieci

## Uczenie nadzorowane

Zbiór uczący to zbiór par:

$$(\mathbf{x}_i, d_i), \quad i = 1 \dots L,$$

gdzie  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in} \in \mathbb{R}^n)$  są danymi wejściowymi a  $d_i \in \{0, 1\}$  lub  $d_i \in \mathbb{R}$  są skojarzonymi z nimi zadanymi wyjściami.

# Metody uczenia sieci

## Uczenie nadzorowane

Zbiór uczący to zbiór par:

$$(\mathbf{x}_i, d_i), \quad i = 1 \dots L,$$

gdzie  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in} \in \mathbb{R}^n)$  są danymi wejściowymi a  $d_i \in \{0, 1\}$  lub  $d_i \in \mathbb{R}$  są skojarzonymi z nimi zadanymi wyjściami.

## Uczenie nienadzorowane

Zbiór uczący to zbiór:

$$(\mathbf{x}_i), \quad i = 1 \dots L,$$

gdzie  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in} \in \mathbb{R}^n)$  są danymi wejściowymi.

# Reguła delty

Po prezentacji kolejnej próbki nr  $p$ , wagi modyfikujemy wg wzoru:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$$

gdzie:

$$\delta_p = d_p - y_p = \begin{cases} -1 \\ 0 \\ 1 \end{cases}$$

$d_p$  – wyjście zadane dla próbki nr  $p$

$y_p$  – wyjście obliczone przez neuron dla próbki  $p$

$\mathbf{x}_p$  – wejście dla próbki nr  $p$

$\eta$  – współczynnik szybkości uczenia

$k$  – krok uczenia

# Algorytm uczenia

- 1 Inicjacja wag początkowych
- 2  $n = 1$  (ustawiamy licznik stopu)
- 3 **while**  $n > 0$  (sprawdzamy czy wszystkie próbki sklasyfikowano poprawnie)
  - $n = 0$  (zerujemy licznik stopu)
  - Mieszamy losowo próbki w zbiorze uczącym
  - Dla kolejnych próbek:
    - Obliczamy wyjście  $y_p$  neuronu dla próbki  $p$
    - Obliczamy błąd  $\delta_p = d_p - y_p$
    - **if**  $\delta_p \neq 0$   
 $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$   
 $n++$  (zwiększamy licznik stopu)

# Cechy algorytmu uczenia

- Zawsze kończy swoje działanie jeżeli zbiór wzorców jest liniowo separowalny.
- Nie zatrzyma się gdy wzorce nie są liniowo separowalne.
- Margines separacji większy lub równy zero.

W praktyce chcemy aby margines separacji był jak największy.

# Cechy algorytmu uczenia

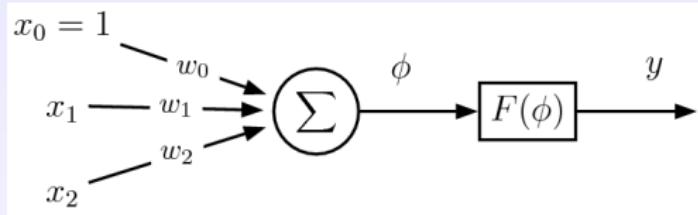
## Twierdzenie Novikoffa

Jeżeli zbiór wzorców jest liniowo separowalny tzn. istnieje prosta zdefiniowana przez współczynniki wagowe  $\mathbf{w}$  neuronu, gwarantująca poprawną klasyfikację binarną, wtedy liczba aktualnień wektora wag  $K$  jest skończona.

Maksymalna wartość  $K$  zależy od marginesu separacji  $\gamma$  ( $K_{max} = f(\frac{1}{\gamma^2})$ ).

# Przykład

Dany jest neuron ze skokową unipolarną funkcją aktywacji.



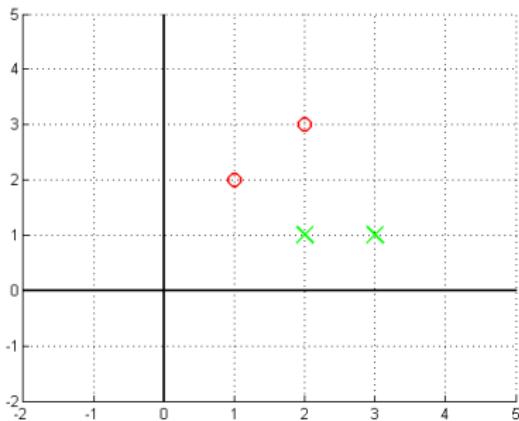
Wykorzystując regułę delty, dobrać współczynniki wagowe neuronu tak, aby prawidłowo klasyfikował próbki uczące podane w tabeli poniżej.

$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

Współczynnik szybkości uczenia:  $\eta = 0.2$ .

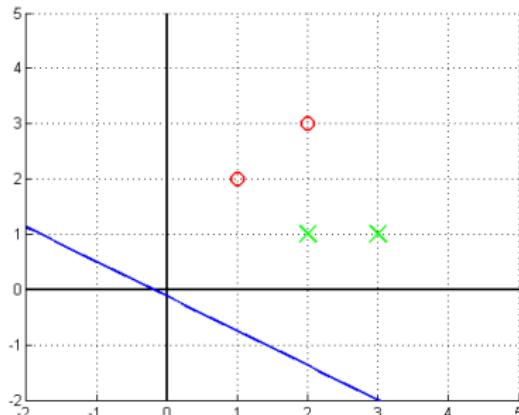
Wagi początkowe:  $\mathbf{w}(0) = [w_0 \ w_1 \ w_2]^T = [0.1 \ 0.5 \ 0.8]^T$

# Przykład



$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

# Przykład



Wagi:

$$\mathbf{w}(0) = [w_0 \ w_1 \ w_2]^T = [0.1 \ 0.5 \ 0.8]^T$$

Prosta separująca:

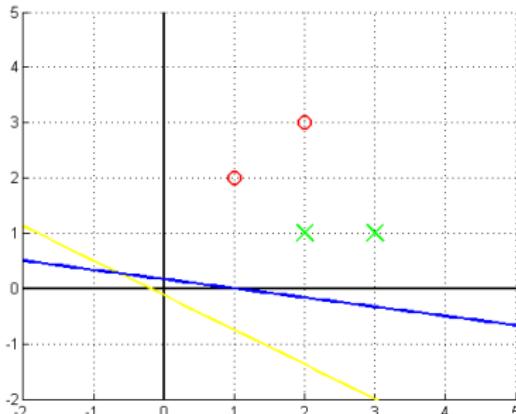
$$w_0 + w_1x_1 + w_2x_2 = 0$$

lub inaczej:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2} = -\frac{5}{8}x_1 - \frac{1}{8}$$

$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

# Przykład



Wagi:

$$\mathbf{w} = [w_0 \ w_1 \ w_2]^T = [-0.1 \ 0.1 \ 0.6]^T$$

Prosta separująca:

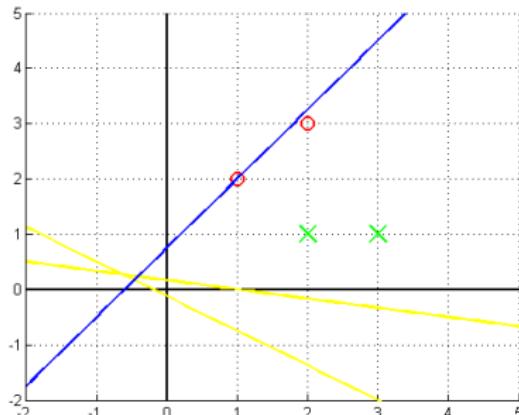
$$w_0 + w_1x_1 + w_2x_2 = 0$$

lub inaczej:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2} = -\frac{1}{6}x_1 + \frac{1}{6}$$

$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

# Przykład



Wagi:

$$\mathbf{w} = [w_0 \ w_1 \ w_2]^T = [-0.3 \ -0.5 \ 0.4]^T$$

Prosta separująca:

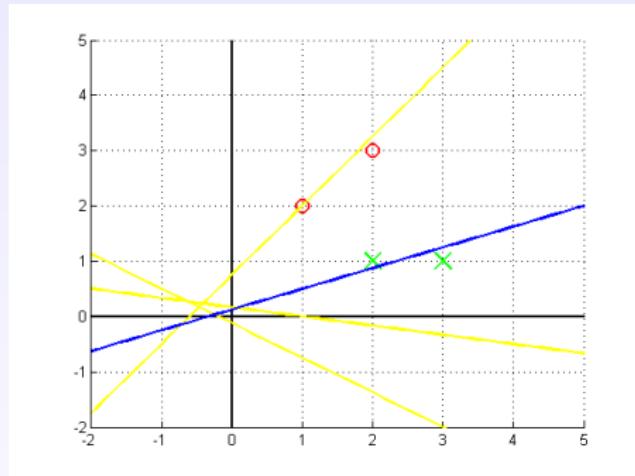
$$w_0 + w_1x_1 + w_2x_2 = 0$$

lub inaczej:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2} = \frac{5}{4}x_1 + \frac{3}{4}$$

$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

# Przykład



Wagi:

$$\mathbf{w} = [w_0 \ w_1 \ w_2]^T = [-0.1 \ -0.3 \ 0.8]^T$$

Prosta separująca:

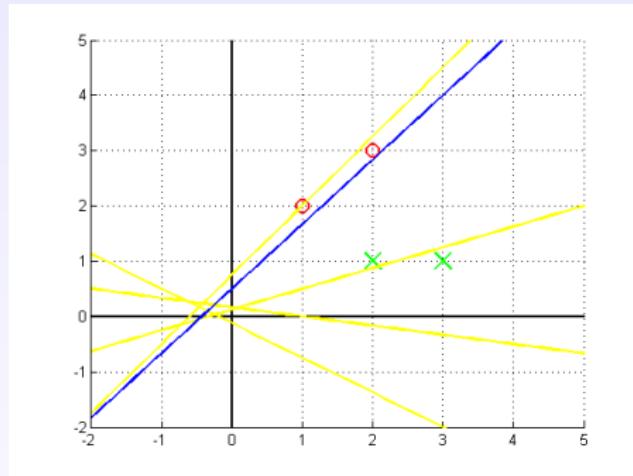
$$w_0 + w_1x_1 + w_2x_2 = 0$$

lub inaczej:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2} = \frac{3}{8}x_1 + \frac{1}{8}$$

$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

# Przykład



Wagi:

$$\mathbf{w} = [w_0 \ w_1 \ w_2]^T = [-0.3 \ -0.7 \ 0.6]^T$$

Prosta separująca:

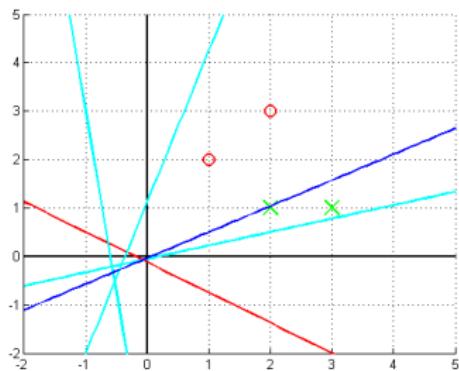
$$w_0 + w_1x_1 + w_2x_2 = 0$$

lub inaczej:

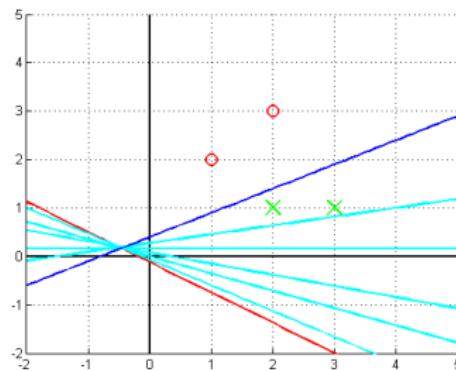
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2} = \frac{7}{6}x_1 + \frac{3}{6}$$

$x_1$	$x_2$	$d$
1	2	1
2	3	1
2	1	0
3	1	0

# Przykład – wpływ $\eta$ na przebieg uczenia

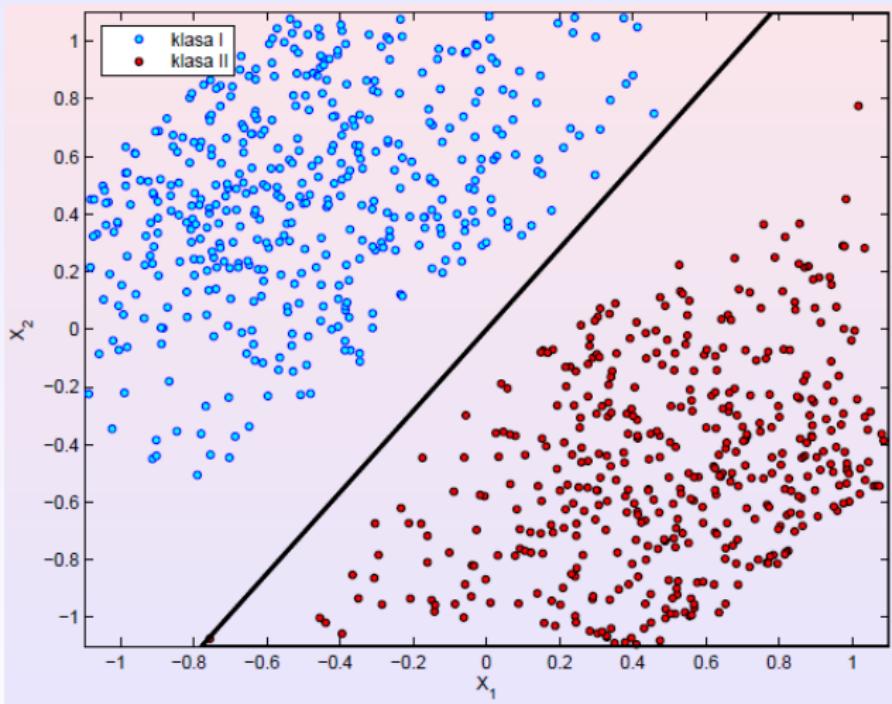


$$\eta = 1$$

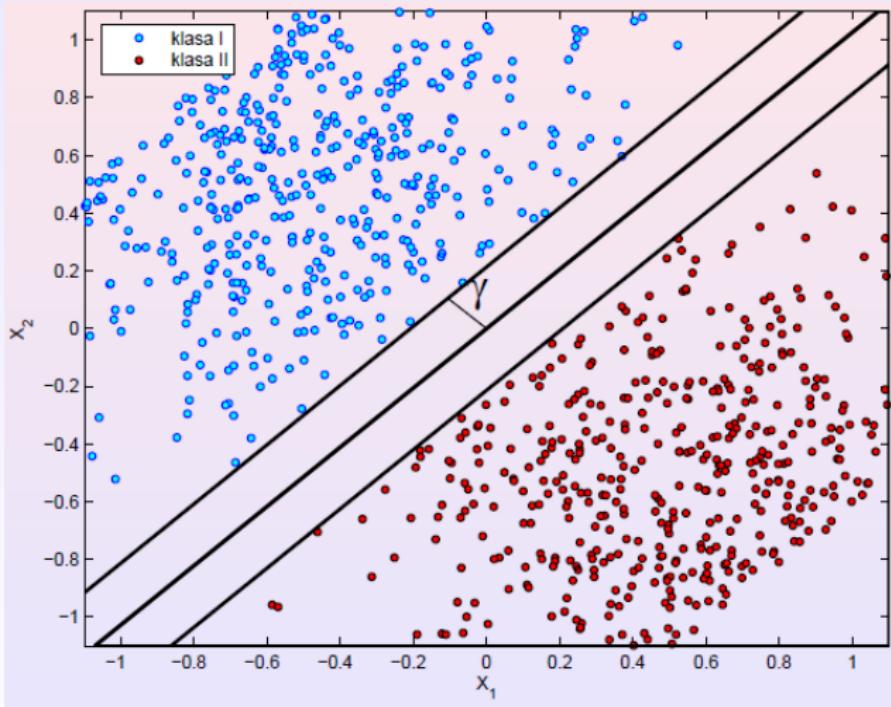


$$\eta = 0.05$$

# Uczenie perceptronu z marginesem separacji $\gamma$



# Uczenie perceptronu z marginesem separacji $\gamma$



1 Inicjacja wag początkowych

2  $n = 1$  (ustawiamy licznik stopu)

3 **while**  $n > 0$  (sprawdzamy czy wszystkie próbki sklasyfikowano poprawnie)

- $n = 0$  (zerujemy licznik stopu)

- Mieszamy losowo próbki w zbiorze uczącym

- Dla kolejnych próbek:

- Obliczamy wyjście  $y_p$  neuronu dla próbki  $p$

- Obliczamy błąd  $\delta_p = d_p - y_p$

- Obl. odległość próbki od prostej separacji  $I_p = \frac{\phi_p}{||w||}$

- **if**  $\delta_p \neq 0$

- $w(k+1) = w(k) + \eta \cdot \delta_p \cdot x_p$

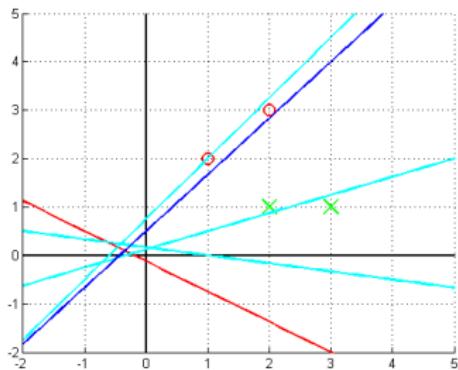
- $n++$  (zwiększamy licznik stopu)

- **elseif**  $\text{abs}(I_p) < \gamma_{\min} - tol$  ( $tol$  – mała wartość, np. 0.01)

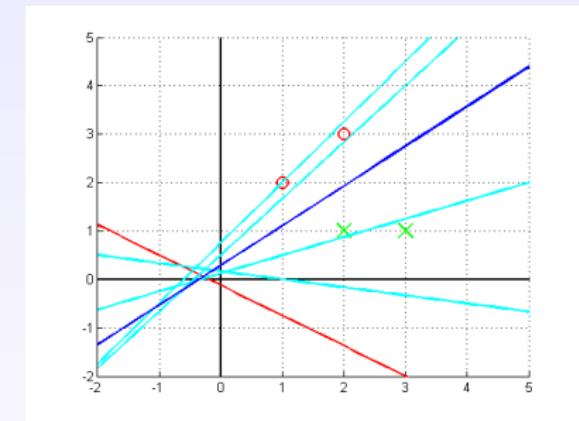
- $w(k+1) = w(k) + \eta \cdot (\text{sgn}(I_p) \cdot \gamma_{\min} - I_p) \cdot x_p$

- $n++$  (zwiększamy licznik stopu)

# Przykład – uczenie perceptronu z marginesem separacji $\gamma$



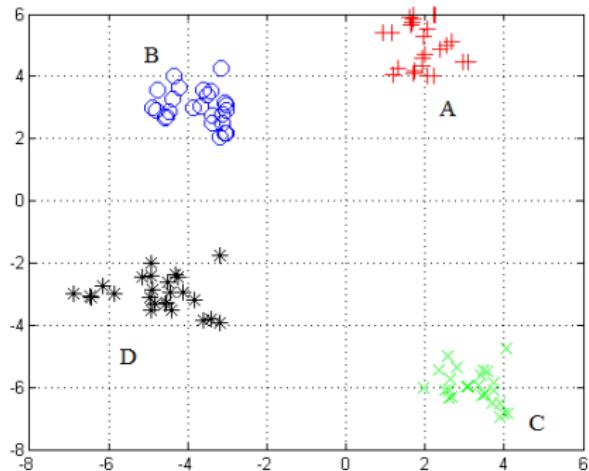
Uczenie bez zadanego marginesu  
separacji



$$\gamma_{min} = 0.6$$

# Przykład – klasyfikacja wielowartościowa

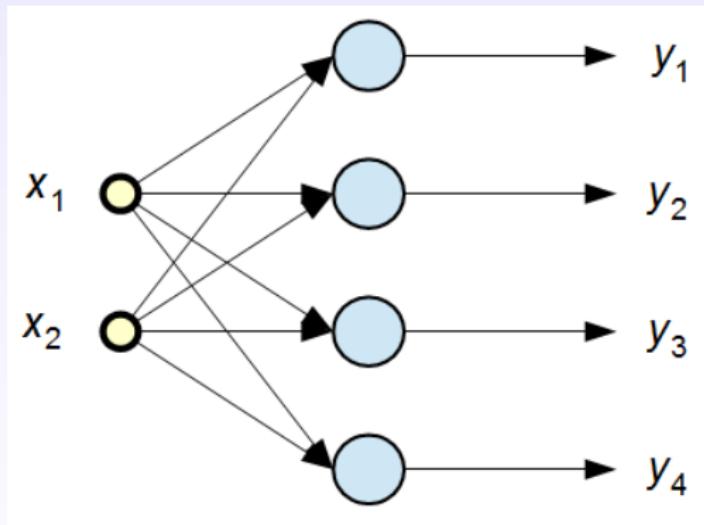
Próbki należą do 4 klas:



2.04	5.54	A
1.92	4.58	A
⋮	⋮	⋮
-3.67	3.04	B
-3.02	3.05	B
⋮	⋮	⋮
3.08	-5.95	C
3.34	-5.94	C
⋮	⋮	⋮
-4.13	-2.95	D
-4.44	-2.94	D
⋮	⋮	⋮

## Przykład – klasyfikacja wielowartościowa

Tworzymy jednowarstwową sieć z 4 neuronami.



# Przykład – klasyfikacja wielowartościowa

Dane po kodowaniu:

Wyjścia zadane  
kodujemy następująco:

A → 1000

B → 0100

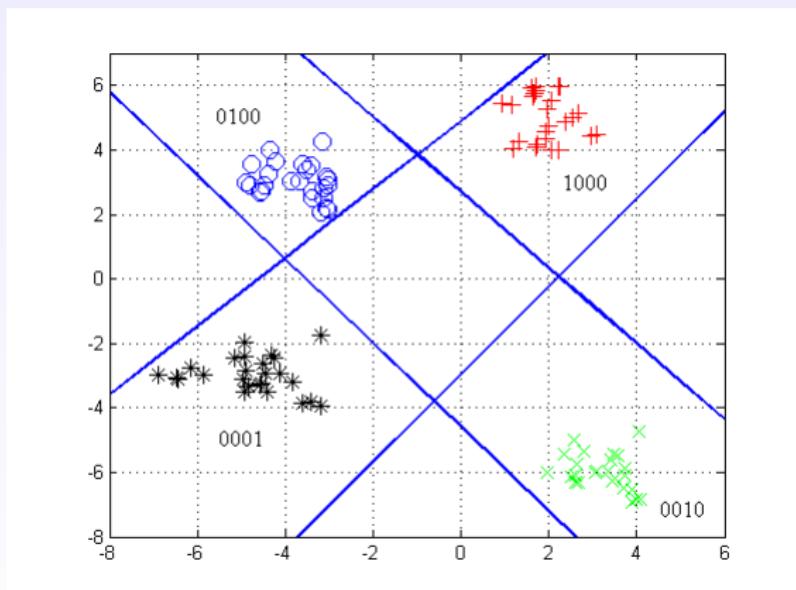
C → 0010

D → 0001

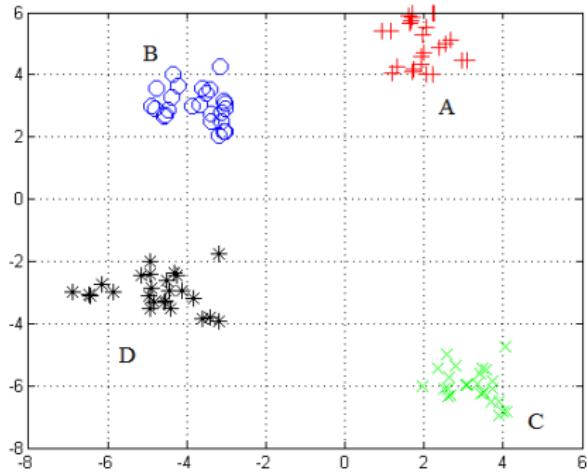
$x_1$	$x_2$	Klasa	$d_1$	$d_2$	$d_3$	$d_4$
2.04	5.54	A	1	0	0	0
1.92	4.58	A	1	0	0	0
⋮						
-3.67	3.04	B	0	1	0	0
-3.02	3.05	B	0	1	0	0
⋮						
3.08	-5.95	C	0	0	1	0
3.34	-5.94	C	0	0	1	0
⋮						
-4.13	-2.95	D	0	0	0	1
-4.44	-2.94	D	0	0	0	1
⋮						

## Przykład – klasyfikacja wielowartościowa

Każdy neuron możemy uczyć osobno, biorąc jako wyjścia zadane kolejno:  $d_1$ ,  $d_2$ ,  $d_3$  i  $d_4$ . Efekt uczenia (z zadanym marginesem separacji):



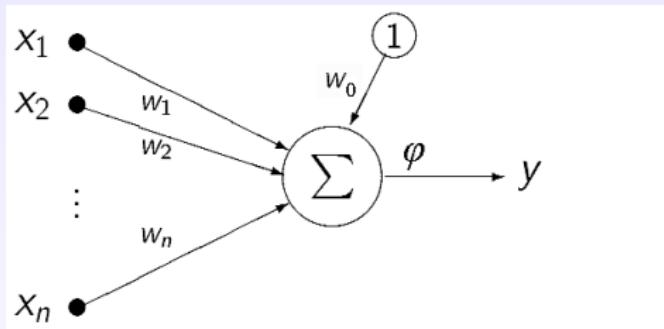
# Przykład – klasyfikacja wielowartościowa



Czy możliwe byłoby zastosowanie sieci z 2 wyjściami i przyjęcie kodowania:

- A → 00
- B → 10
- C → 11
- D → 01

# Neuron typu ADALINE (Adaptive Linear Neuron)



$$y(\mathbf{x}) = \phi = \sum_{i=0}^n w_i x_i = \mathbf{w}^T \mathbf{x}$$

# Reguła delty

Po prezentacji kolejnej próbki nr  $p$ , wagi modyfikujemy wg wzoru:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$$

gdzie:

$$\delta_p = d_p - y_p$$

$d_p$  – wyjście zadane dla próbki nr  $p$

$y_p$  – wyjście obliczone przez neuron dla próbki  $p$

$\mathbf{x}_p$  – wejście dla próbki nr  $p$

$\eta$  – współczynnik szybkości uczenia

$k$  – krok uczenia

# Reguła delty

Po prezentacji kolejnej próbki nr  $p$ , wagi modyfikujemy wg wzoru:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$$

gdzie:

$$\delta_p = d_p - y_p$$

$d_p$  – wyjście zadane dla próbki nr  $p$

$y_p$  – wyjście obliczone przez neuron dla próbki  $p$

$\mathbf{x}_p$  – wejście dla próbki nr  $p$

$\eta$  – współczynnik szybkości uczenia

$k$  – krok uczenia

## Poprawka skumulowana

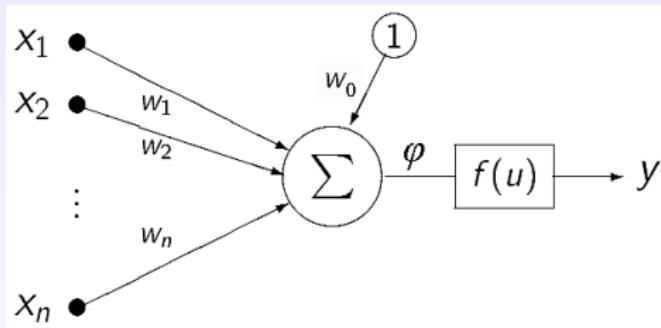
Po prezentacji wszystkich próbek ze zbioru uczącego liczymy:

$$\Delta \mathbf{w} = \frac{1}{L} \eta \sum_{p=1}^L \delta_p \cdot \mathbf{x}_p$$

# Algorytm uczenia

- 1 Inicjacja wag początkowych
- 2  $Q = \text{Real\_max\_value}$  (ustawiamy dużą wartość początkową błędu)
- 3 **while**  $Q > Q_{min}$  (sprawdzamy czy błąd jest mniejszy od zadanego)
  - $Q = 0$  (zerujemy błąd)
  - Mieszamy losowo próbki w zbiorze uczącym
  - Dla kolejnych próbek:
    - Obliczamy wyjście  $y_p$  neuronu dla próbki  $p$
    - Obliczamy błąd  $\delta_p = d_p - y_p$
    - Poprawiamy wagi  $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$
    - $Q = Q + \delta_p^2$
  - Liczymy średni błąd  $Q = Q/L$  ( $L$  – ilość próbek)

# Neuron z nieliniową funkcją aktywacji



$$y(\mathbf{x}) = f\left(\sum_{i=0}^n w_i x_i\right) = f(\mathbf{w}^T \mathbf{x})$$

# Uogólniona reguła delty

Po prezentacji kolejnej próbki nr  $p$ , wagi modyfikujemy wg wzoru:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot f'(\phi_p) \cdot \mathbf{x}_p$$

gdzie:

$$\delta_p = d_p - y_p$$

$d_p$  – wyjście zadane dla próbki nr  $p$

$y_p$  – wyjście obliczone przez neuron dla próbki  $p$

$\mathbf{x}_p$  – wejście dla próbki nr  $p$

$\eta$  – współczynnik szybkości uczenia

$k$  – krok uczenia

# Uogólniona reguła delty

Po prezentacji kolejnej próbki nr  $p$ , wagi modyfikujemy wg wzoru:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot f'(\phi_p) \cdot \mathbf{x}_p$$

gdzie:

$$\delta_p = d_p - y_p$$

$d_p$  – wyjście zadane dla próbki nr  $p$

$y_p$  – wyjście obliczone przez neuron dla próbki  $p$

$\mathbf{x}_p$  – wejście dla próbki nr  $p$

$\eta$  – współczynnik szybkości uczenia

$k$  – krok uczenia

Błąd uogólniony

$$\delta'_p = \delta_p \cdot f'(\phi_p)$$

# Pochodne funkcji aktywacji

Pochodna sigmoidalnej funkcji aktywacji:

$$f(\phi) = \frac{1}{1 + e^{-\beta\phi}}$$

może być obliczona za pomocą wzoru:

$$f'(\phi) = \beta \cdot f(\phi) \cdot (1 - f(\phi))$$

# Pochodne funkcji aktywacji

Pochodna sigmoidalnej funkcji aktywacji:

$$f(\phi) = \frac{1}{1 + e^{-\beta\phi}}$$

może być obliczona za pomocą wzoru:

$$f'(\phi) = \beta \cdot f(\phi) \cdot (1 - f(\phi))$$

Pochodna funkcji aktywacji typu tangens hiperboliczny:

$$f(\phi) = \tanh(\phi) = \frac{e^{\beta\phi} - e^{-\beta\phi}}{e^{\beta\phi} + e^{-\beta\phi}}$$

może być obliczona za pomocą wzoru:

$$f'(\phi) = \beta \cdot (1 - f^2(\phi))$$

# Uogólniona reguła delty

- Algorytm jest zbieżny do najbliższego minimum lokalnego funkcji błędu  $Q(\mathbf{w})$ .
- Algorytm nie gwarantuje znalezienia minimum globalnego!
- Konieczność uczenia z różnymi, losowymi wagami początkowymi.

# Składnik momentum

Po prezentacji kolejnej próbki nr  $p$ , wagi modyfikujemy wg wzoru:

$$\Delta\mathbf{w}(k + 1) = \eta \cdot \delta_p \cdot f'(\phi_p) \cdot \mathbf{x}_p + \alpha \cdot \Delta\mathbf{w}(k)$$

$$\mathbf{w}(k + 1) = \mathbf{w}(k) + \Delta\mathbf{w}(k + 1)$$

gdzie:

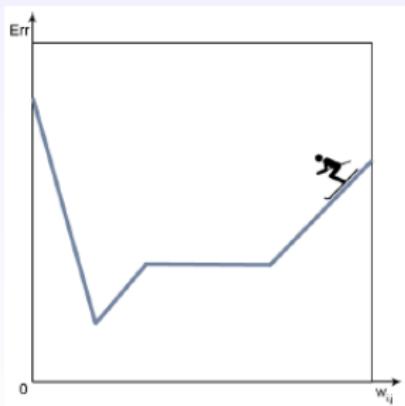
$\alpha$  – współczynnik pędu (momentum)

$k$  – krok uczenia

# Składnik momentum

Składnik momentum:

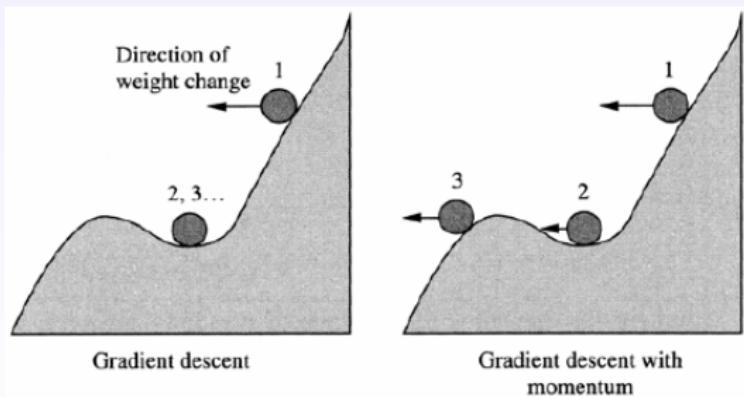
- przyspiesza proces uczenia sieci,
- pozwala na stosowanie większych wartości współczynnika szybkości uczenia  $\eta$ ,
- umożliwia przeskakiwanie lokalnych minimów błędu,
- wygasza oscylacje wag w trakcie uczenia.



# Składnik momentum

Składnik momentum:

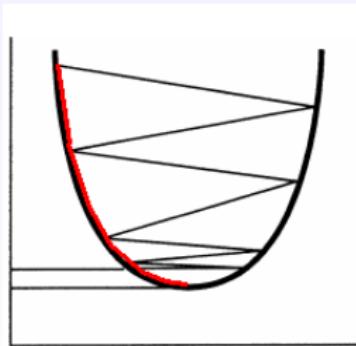
- przyspiesza proces uczenia sieci,
- pozwala na stosowanie większych wartości współczynnika szybkości uczenia  $\eta$ ,
- umożliwia przeskakiwanie lokalnych minimów błędu,
- wygasza oscylacje wag w trakcie uczenia.



# Składnik momentum

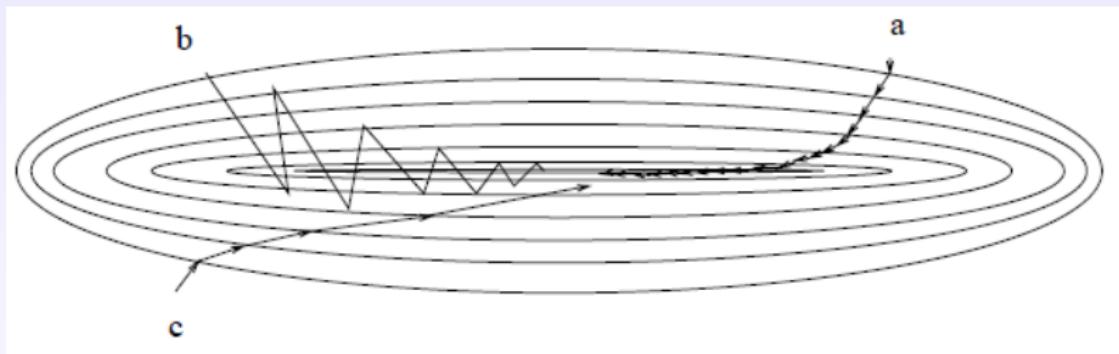
Składnik momentum:

- przyspiesza proces uczenia sieci,
- pozwala na stosowanie większych wartości współczynnika szybkości uczenia  $\eta$ ,
- umożliwia przeskakiwanie lokalnych minimów błędu,
- wygasza oscylacje wag w trakcie uczenia.



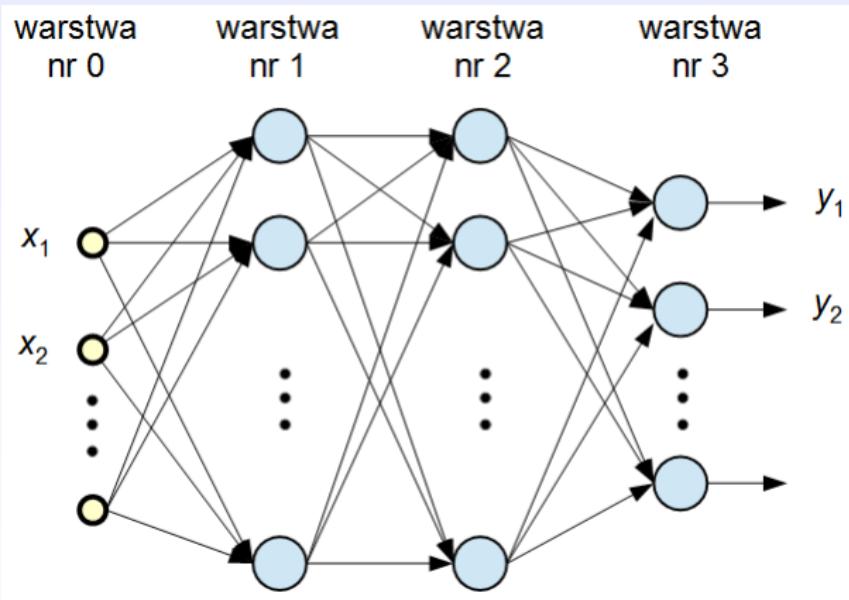
# Składnik momentum

Przykładowy przebieg uczenia:

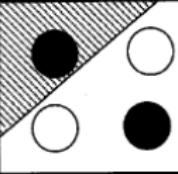
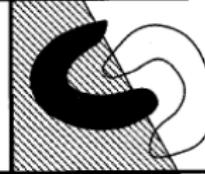
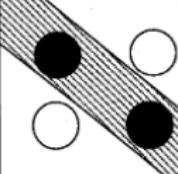
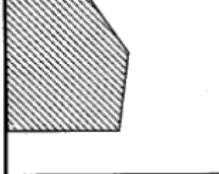
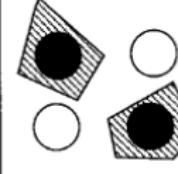
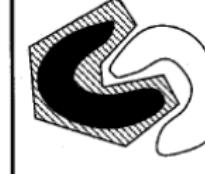
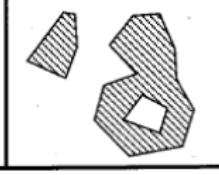


- a) mały współczynnik  $\eta$ ,
- b) duży współczynnik  $\eta$ ,
- c) duży współczynnik  $\eta$  z dodanym składnikiem pędu.

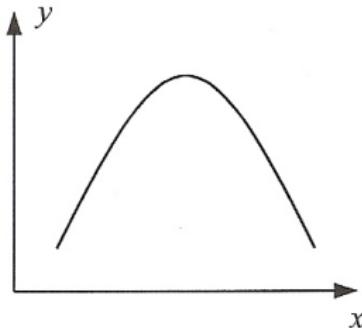
# Sieci jednokierunkowe, wielowarstwowe



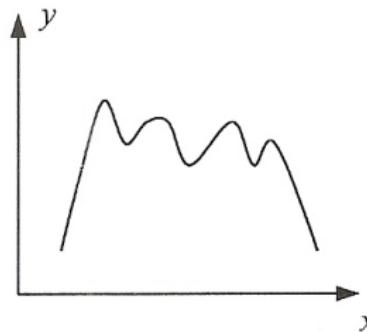
# Dobór struktury sieci

Structure	Description of decision regions	Exclusive-OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

# Dobór struktury sieci

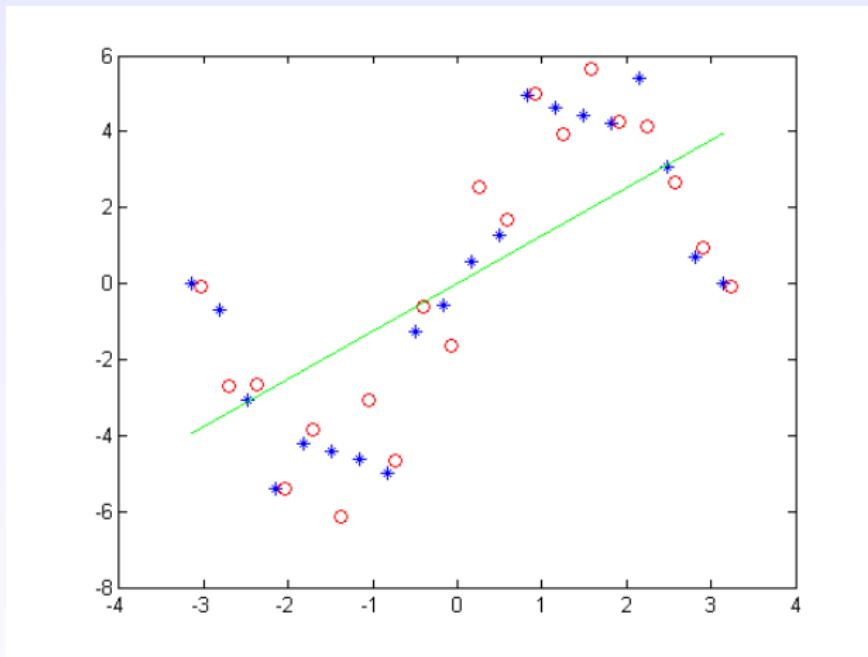


a) nieskomplikowana powierzchnia:  
mniej neuronów w  
warstwie pośredniej



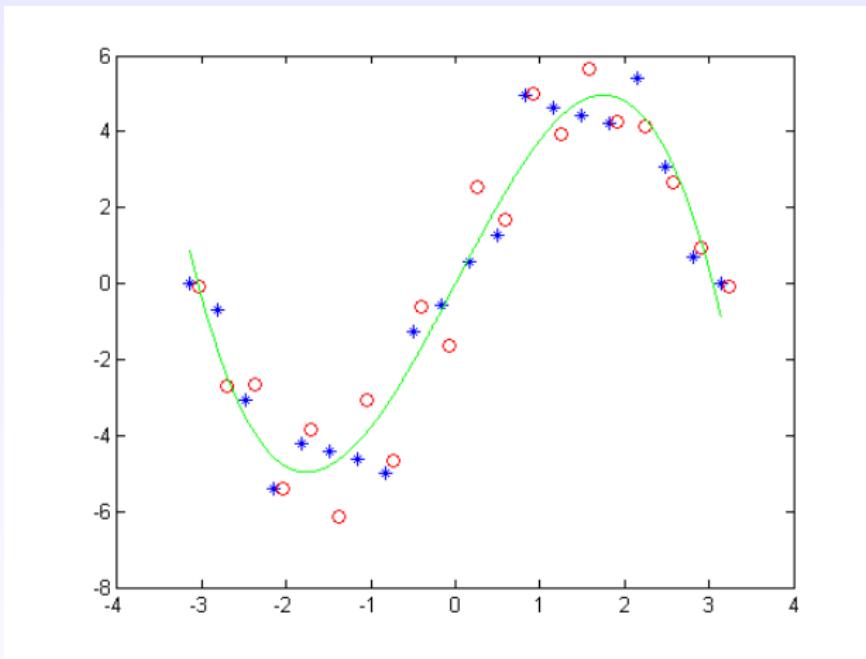
b) skomplikowana powierzchnia:  
więcej neuronów w  
warstwie pośredniej

# Aproksymacja wielomianem rzędu 1



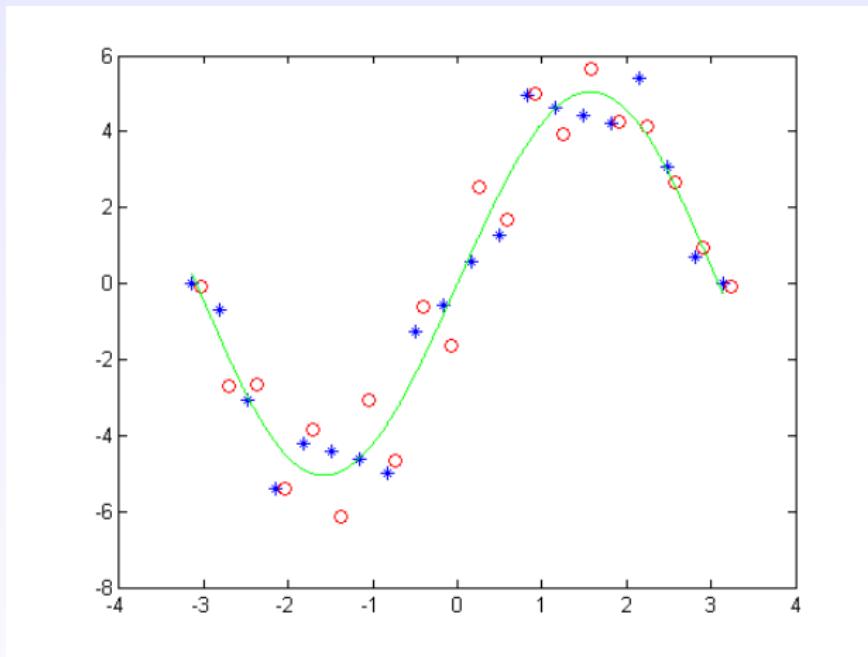
Błąd danych uczących (niebieski) = 44.30;  
Błąd danych testujących (czerwony) = 44.37

# Aproksymacja wielomianem rzędu 3



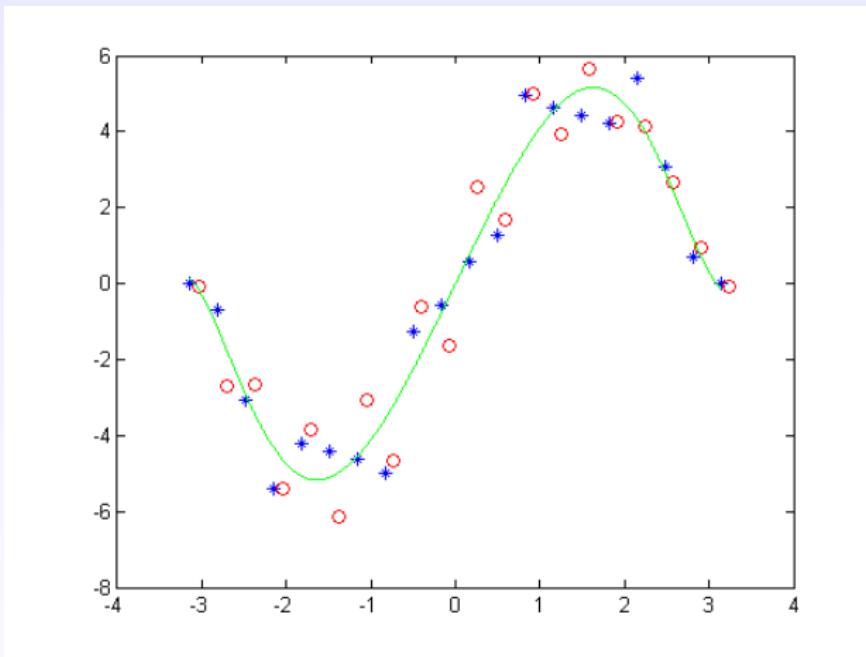
Błąd danych uczących (niebieski) = 14.69;  
Błąd danych testujących (czerwony) = 17.96

# Aproksymacja wielomianem rzędu 5



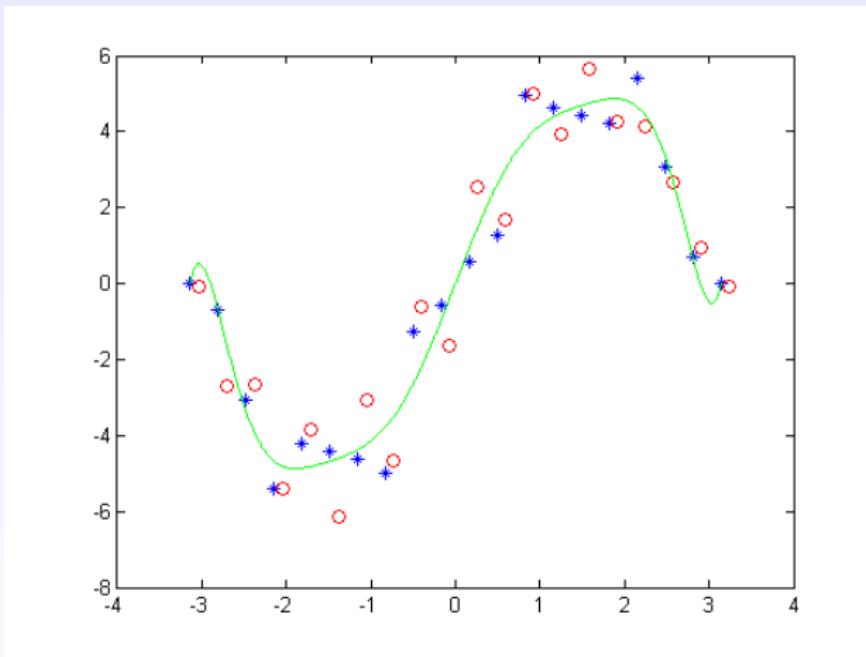
Błąd danych uczących (niebieski) = 12.37;  
Błąd danych testujących (czerwony) = 16.61

# Aproksymacja wielomianem rzędu 7



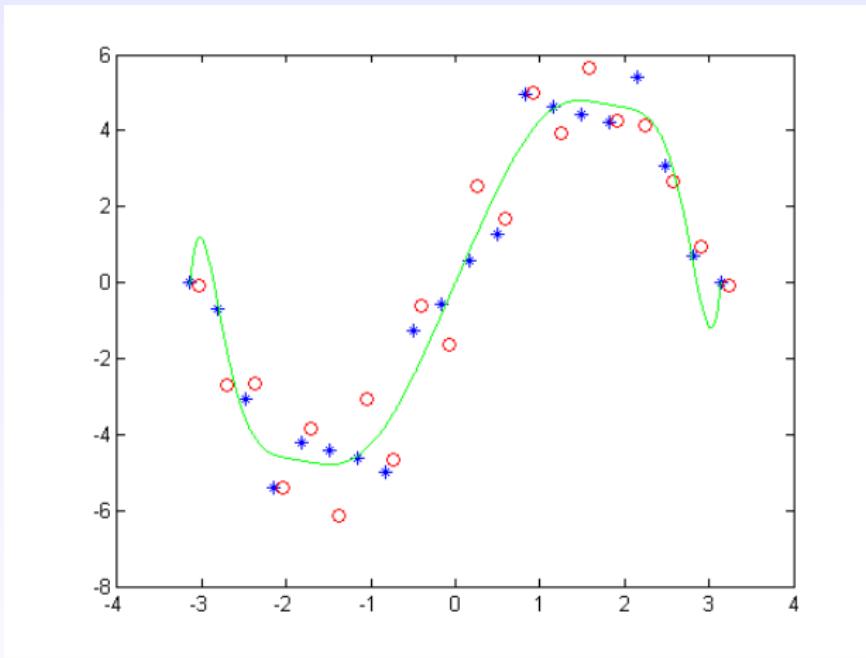
Błąd danych uczących (niebieski) = 11.99;  
Błąd danych testujących (czerwony) = 16.54

# Aproksymacja wielomianem rzędu 9



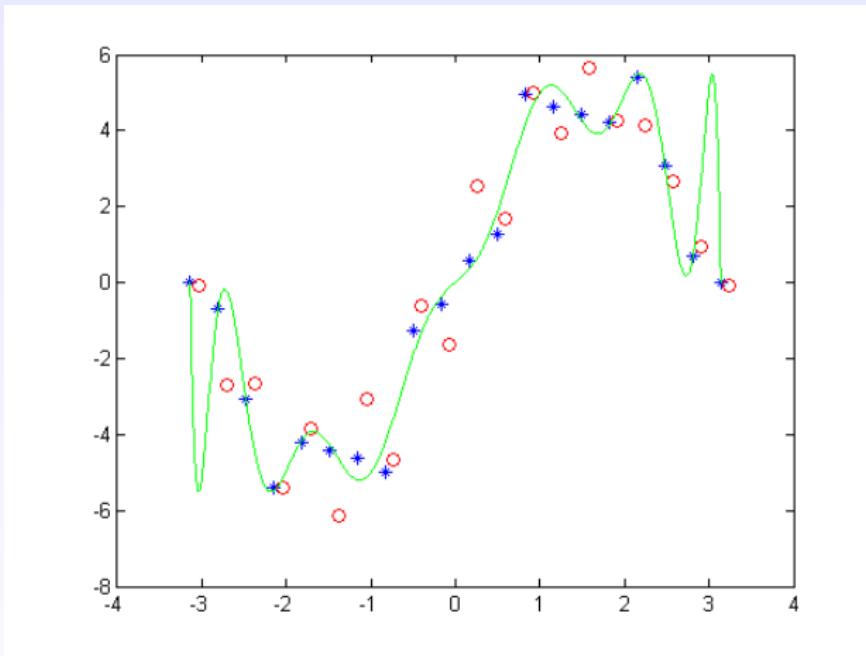
Błąd danych uczących (niebieski) = 10.21;  
Błąd danych testujących (czerwony) = 20.01

# Aproksymacja wielomianem rzędu 11



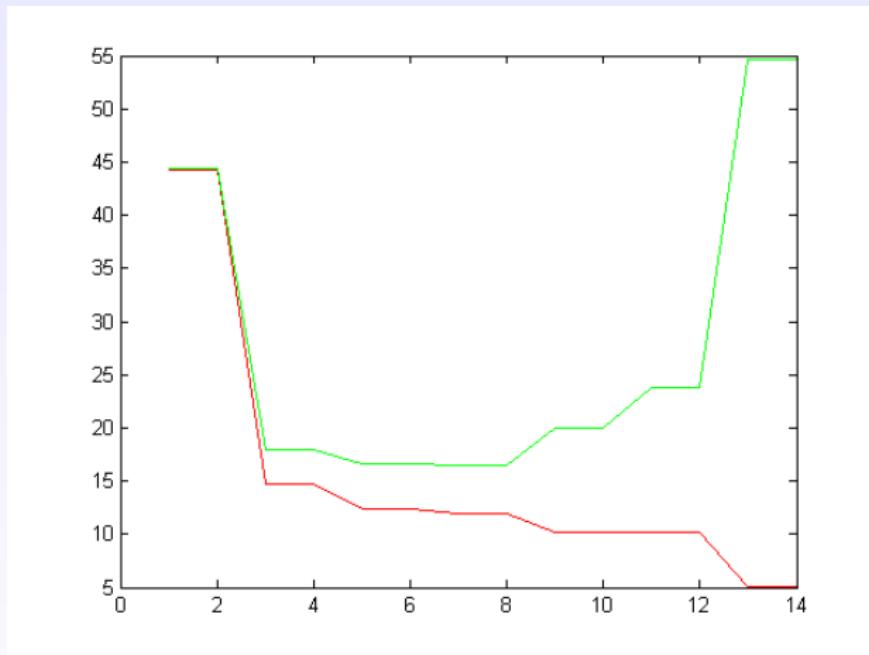
Błąd danych uczących (niebieski) = 10.19;  
Błąd danych testujących (czerwony) = 23.72

# Aproksymacja wielomianem rzędu 13



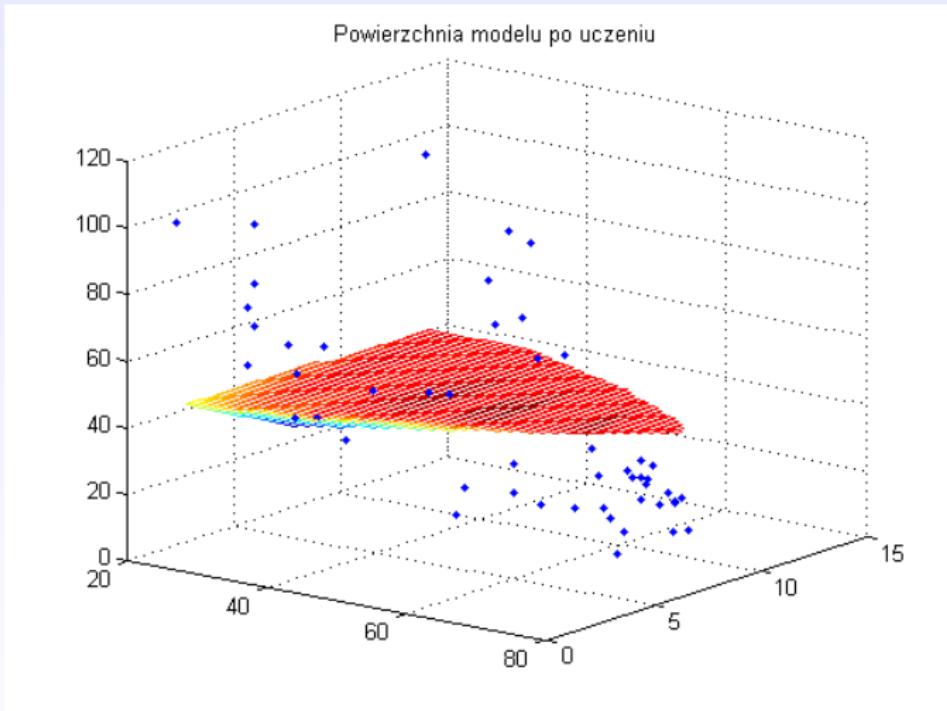
Błąd danych uczących (niebieski) = 5.02;  
Błąd danych testujących (czerwony) = 54.61

# Przeuczenie i niedouczenie sieci



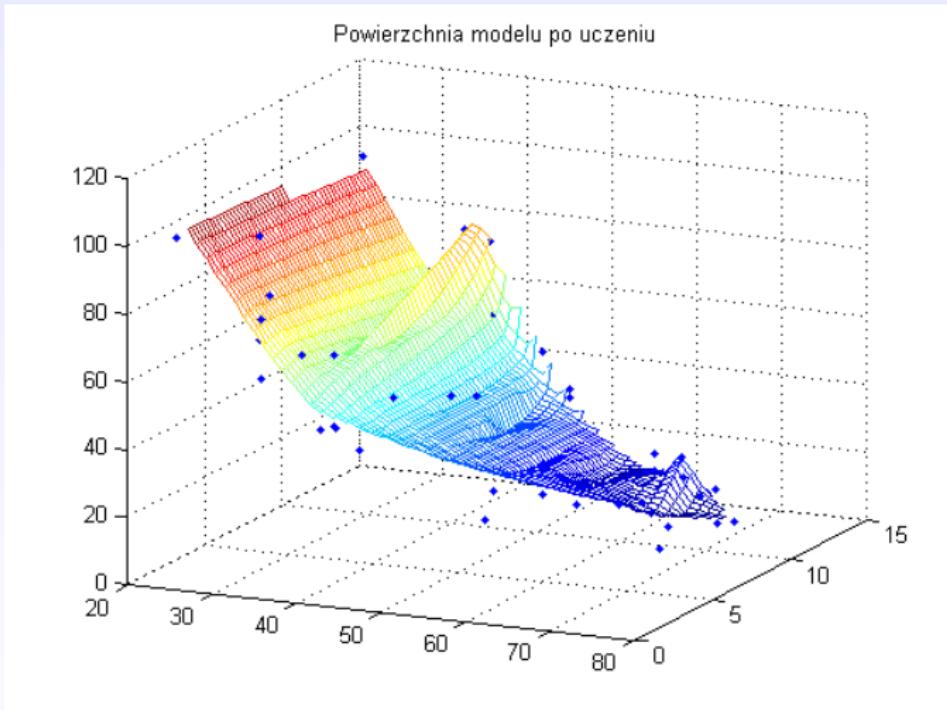
Błąd danych uczących (czerwony) i błąd danych testujących (zielony) w zależności od rzędu wielomianu.

# Przeuczenie i niedouczenie sieci



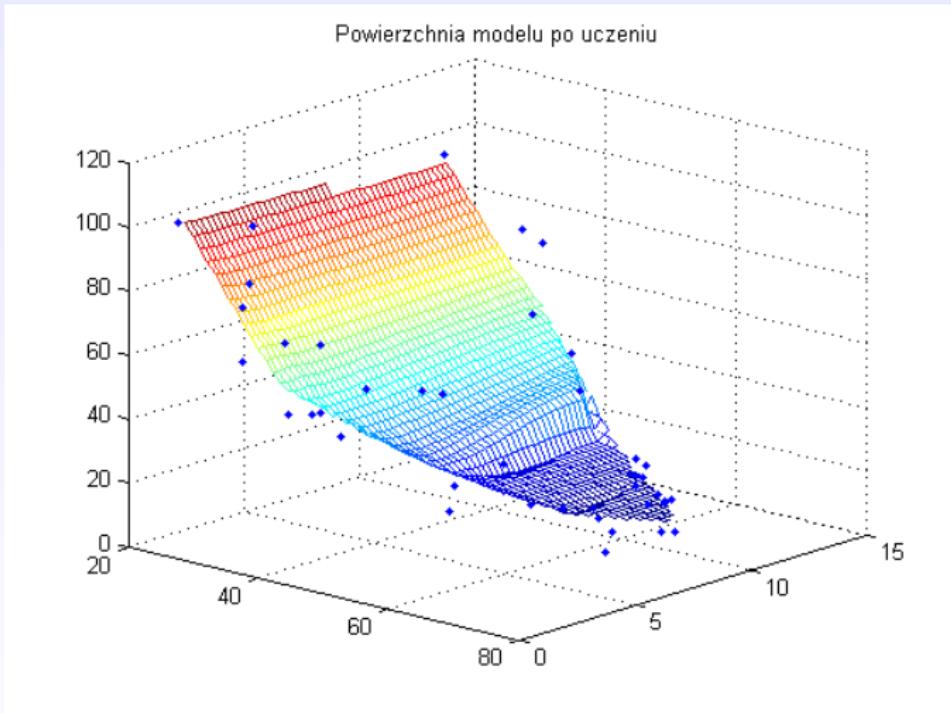
Sieć niedouczona (1 neuron na warstwie ukrytej).

# Przeuczenie i niedouczenie sieci



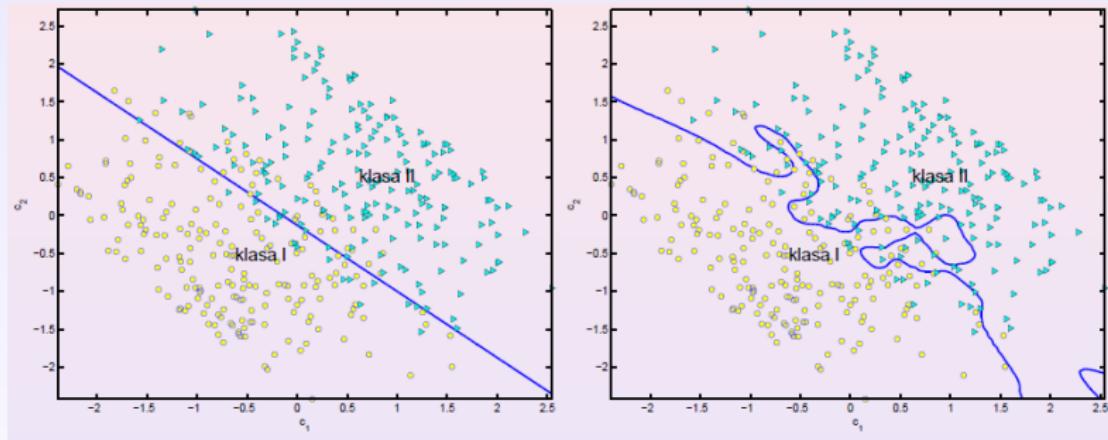
Sieć przeuczona (10 neuronów na warstwie ukrytej).

# Przeuczenie i niedouczenie sieci



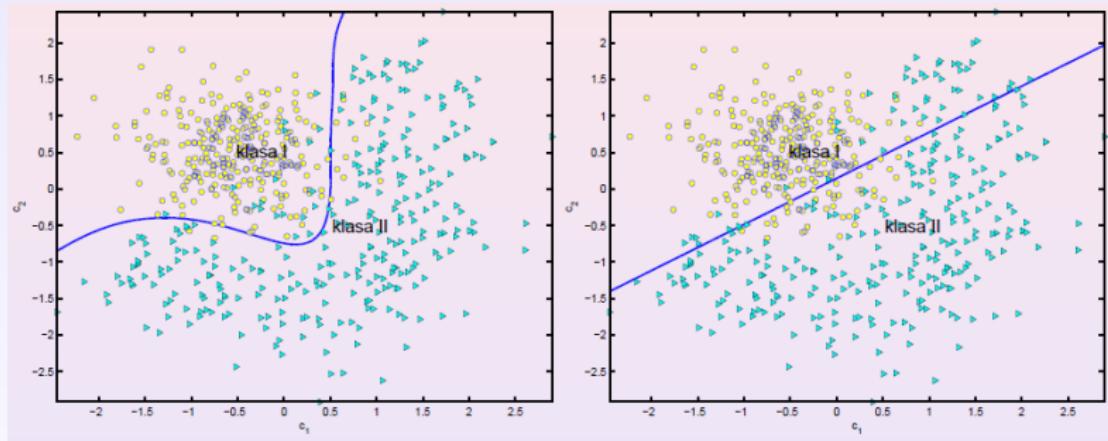
Sieć nauczona prawidłowo (3 neurony na warstwie ukrytej).

# Przeuczenie i niedouczenie sieci



Sieć nauczona prawidłowo (po lewej) i przeuczona (po prawej).

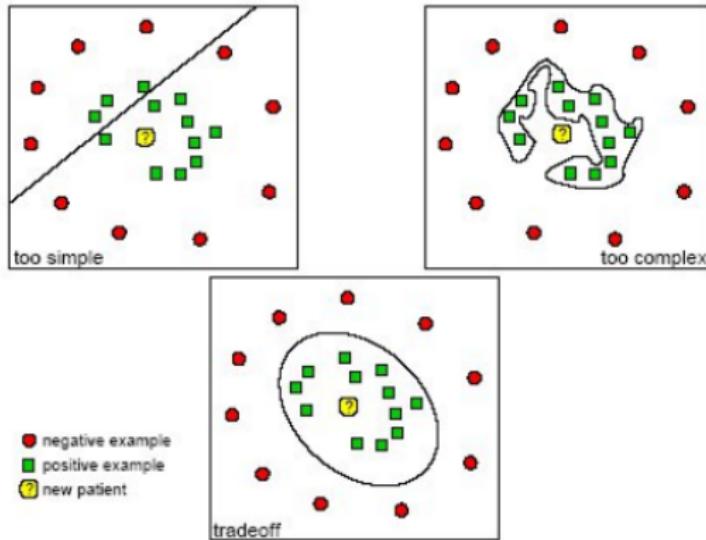
# Przeuczenie i niedouczenie sieci



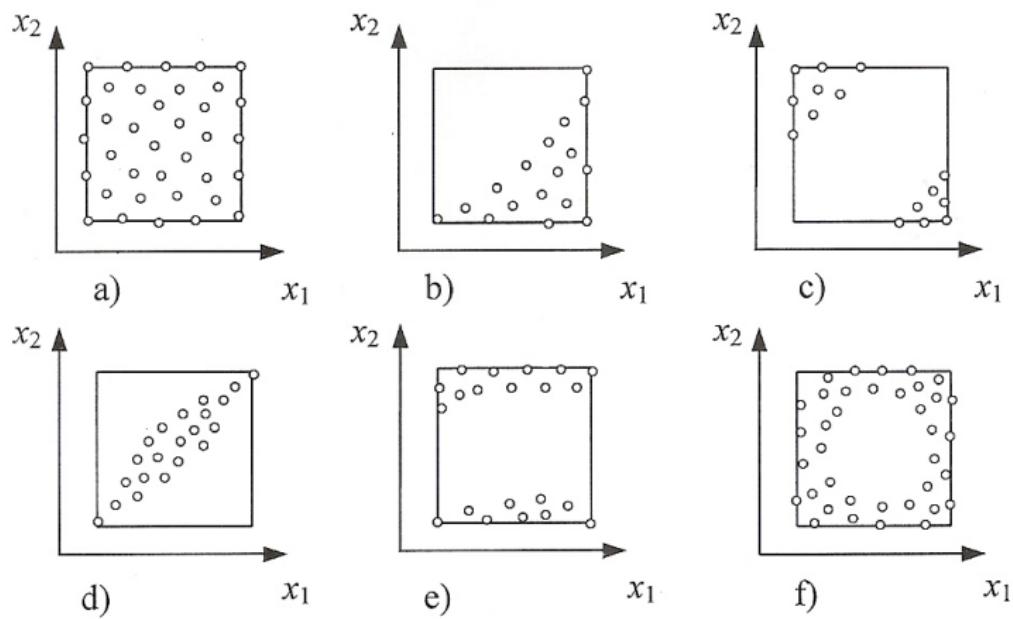
Sieć nauczona prawidłowo (po lewej) i niedouczona (po prawej).

# Przeuczenie i niedouczenie sieci

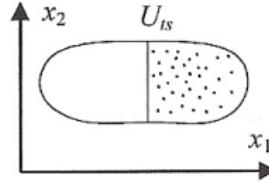
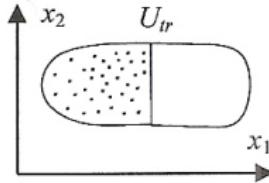
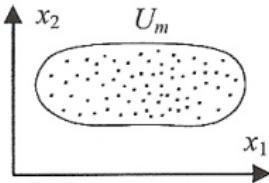
## Underfitting and Overfitting



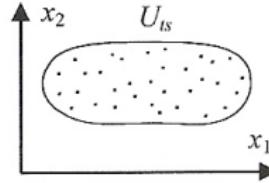
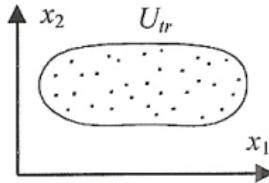
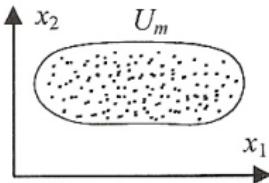
# Dane uczące



# Dane uczące

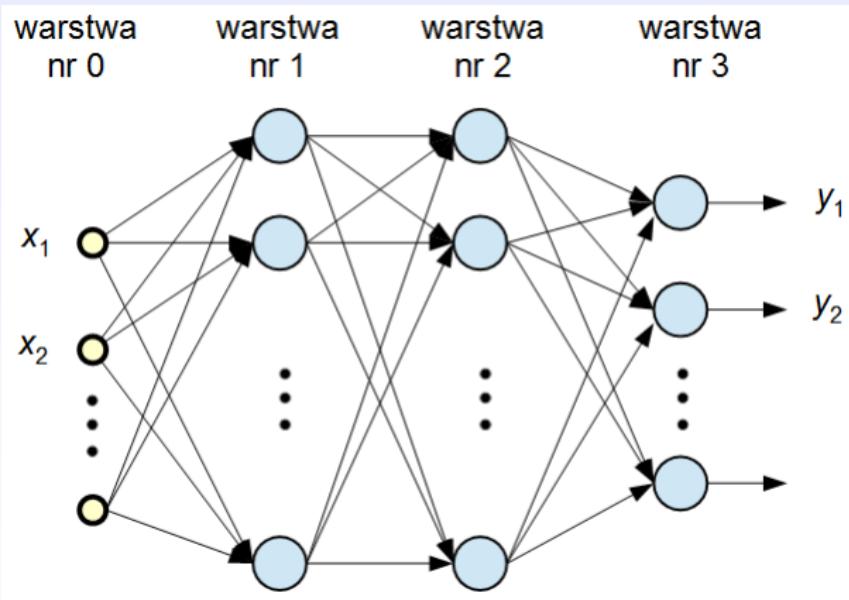


- a) zły podział zbioru pomierowego  $U_m$  na zbiór uczący  $U_{tr}$  i zbiór testujący  $U_{is}$



- b) prawidłowy podział zbioru pomierowego  $U_m$  na zbiór uczący  $U_{tr}$  i zbiór testujący  $U_{is}$

# Sieci jednokierunkowe, wielowarstwowe



# Sieci jednokierunkowe, wielowarstwowe

Rozważamy  $N$ -warstwową sieć o jednakowych funkcjach aktywacji  $f(\phi)$ . Wprowadźmy następujące oznaczenia:

- $n = 0, \dots, N$  – nr warstwy
- $p = 1, \dots, L$  – numer próbki ( $L$  – ilość próbek w zbiorze uczącym)
- $i, j = 1, \dots, t_n$  – nr neuronu na warstwie  $n$  ( $t_n$  – ilość neuronów na warstwie  $n$ )

# Sieci jednokierunkowe, wielowarstwowe

Wyjście neuronu  $j$  na warstwie nr  $n$  można obliczyć jako:

$$v_j^{np} = f(\phi_j^{np}) = f \left( \sum_{i=0}^{t_{n-1}} w_{ji}^n \cdot v_i^{(n-1)p} \right) = f \left( \mathbf{w}_j^{nT} \cdot \mathbf{v}^{(n-1)p} \right) \quad (1)$$

gdzie:  $w_{ji}^n$  – waga na wejściu nr  $i$ .

$$y_j^p = v_j^{Np} \quad x_i^p = v_i^{0p}$$

# Sieci jednokierunkowe, wielowarstwowe

Wyjście neuronu  $j$  na warstwie nr  $n$  można obliczyć jako:

$$v_j^{np} = f(\phi_j^{np}) = f \left( \sum_{i=0}^{t_{n-1}} w_{ji}^n \cdot v_i^{(n-1)p} \right) = f \left( \mathbf{w}_j^{nT} \cdot \mathbf{v}^{(n-1)p} \right) \quad (1)$$

gdzie:  $w_{ji}^n$  – waga na wejściu nr  $i$ .

$$y_j^p = v_j^{Np} \quad x_i^p = v_i^{0p}$$

Dla warstwy wyjściowej możemy policzyć **uogólniony** błąd sieci:

$$\delta'_j^{Np} = f'(\phi_j^{Np}) \cdot \delta_j^p = f'(\phi_j^{Np}) \cdot (d_j^p - y_j^p) \quad (2)$$

# Sieci jednokierunkowe, wielowarstwowe

Błąd wstecznie propagujemy na warstwy ukryte:

$$\delta_j^{(np)} = f'(\phi_j^{np}) \sum_{k=1}^{t_{n+1}} \delta_k^{(n+1)p} \cdot w_{kj}^{(n+1)} \quad (3)$$

# Sieci jednokierunkowe, wielowarstwowe

Błąd wstępnie propagujemy na warstwy ukryte:

$$\delta_j'^{np} = f'(\phi_j^{np}) \sum_{k=1}^{t_{n+1}} \delta_k'^{(n+1)p} \cdot w_{kj}^{(n+1)} \quad (3)$$

Po wyznaczeniu uogólnionych błędów dla wszystkich neuronów w sieci, obliczamy poprawki wag za pomocą uogólnionej reguły delta:

$$\Delta w_{ji}^{np} = \eta \cdot \delta_j'^{np} \cdot v_i^{(n-1)p} \quad (4)$$

$$\Delta \mathbf{w}_j^{np} = \eta \cdot \delta_j'^{np} \cdot \mathbf{v}^{(n-1)p}$$

# Algorytm wstecznej propagacji błędów

Dla próbki uczącej nr  $p$ :

- 1 Podaj wektor  $\mathbf{x}^p$  na wejście sieci.
- 2 Wyznacz wartość wyjścia  $v_j^{np}$  każdego neuronu dla kolejnych warstw sieci, od pierwszej warstwy ukrytej do wyjściowej (wzór nr 1).
- 3 Oblicz wartości błędów uogólnionych dla warstwy wyjściowej (wzór nr 2).
- 4 Dokonaj wstecznej propagacji uogólnionego błędu wyjściowego do poszczególnych neuronów warstw ukrytych (wzór nr 3).
- 5 Oblicz poprawki wag (wzór nr 4) i zmodyfikuj wagi w sieci.

# Algorytm wstecznej propagacji błędów

Dla próbki uczącej nr  $p$ :

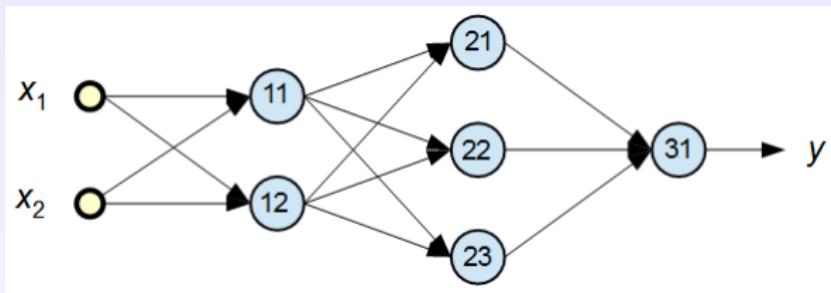
- 1 Podaj wektor  $x^p$  na wejście sieci.
- 2 Wyznacz wartość wyjścia  $v_j^{np}$  każdego neuronu dla kolejnych warstw sieci, od pierwszej warstwy ukrytej do wyjściowej (wzór nr 1).
- 3 Oblicz wartości błędów uogólnionych dla warstwy wyjściowej (wzór nr 2).
- 4 Dokonaj wstecznej propagacji uogólnionego błędu wyjściowego do poszczególnych neuronów warstw ukrytych (wzór nr 3).
- 5 Oblicz poprawki wag (wzór nr 4) i zmodyfikuj wagi w sieci.

3A: Po kroku nr 3, należy obliczyć sumę kwadratów błędów

$$Q^p = \sum_{j=1}^N (\delta_j^{np})^2 \text{ i dodać ją do całkowitego błędu danych uczących } Q.$$

3B: Jeśli była to ostatnia próbka w zbiorze, możemy sprawdzić czy błąd  $Q$  spadł poniżej zadanego progu. Jeśli tak, przerywamy uczenie.

# Sieć neuronów liniowych – przykład



$x_1$	$x_2$	$d$
1	2	1
0	1	2
2	0	3
:	:	:

$$\mathbf{w}^{11}(0) = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.1 \end{bmatrix} \quad \mathbf{w}^{12}(0) = \begin{bmatrix} 0.2 \\ -0.2 \\ 0.2 \end{bmatrix}$$

$$\mathbf{w}^{21}(0) = \begin{bmatrix} -0.1 \\ 0.2 \\ 0.3 \end{bmatrix} \quad \mathbf{w}^{22}(0) = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.2 \end{bmatrix} \quad \mathbf{w}^{23}(0) = \begin{bmatrix} -0.2 \\ 0.1 \\ 0.2 \end{bmatrix} \quad \mathbf{w}^{31}(0) = \begin{bmatrix} -0.1 \\ 0.3 \\ -0.2 \\ 0.1 \end{bmatrix}$$

# Sieci neuronowe typu RBF

## RBF – Radial Basis Function

Neurony na warstwie ukrytej realizują funkcję zmieniającą się radialnie wokół pewnego punktu  $\mathbf{c}$  zwanego centrum neuronu.

Funkcja ma ogólną postać:

$$f(\mathbf{x}) = \phi(||\mathbf{x} - \mathbf{c}||)$$

gdzie:  $||\mathbf{x} - \mathbf{c}||$  – odległość między próbką  $\mathbf{x}$  i centrum funkcji  $\mathbf{c}$ .

# Sieci neuronowe typu RBF

Jedną z najpopularniejszych funkcji RBF jest funkcja Gaussa:

$$\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2\sigma^2}\right)$$

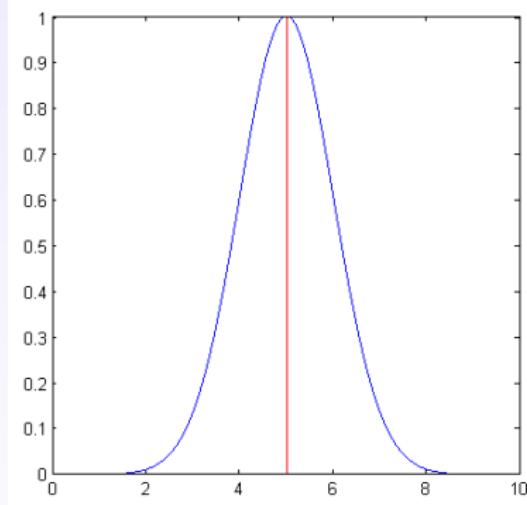
gdzie:

$$\|\mathbf{x} - \mathbf{c}\| = \sqrt{\sum_{i=1}^n (x_i - c_i)^2}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

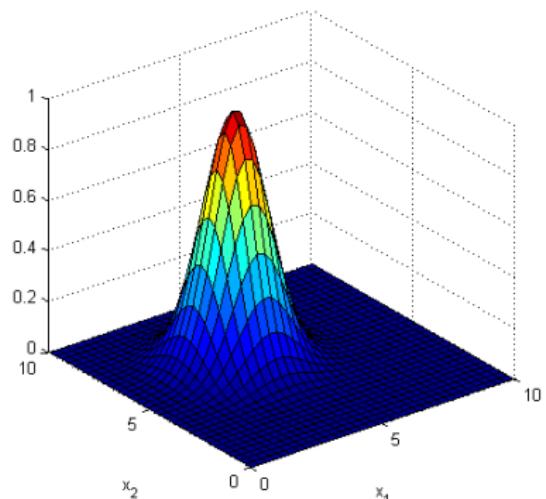
Parametr  $\sigma$  określa szerokość funkcji RBF.

# Funkcja Gaussa

$$\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2\sigma^2}\right)$$



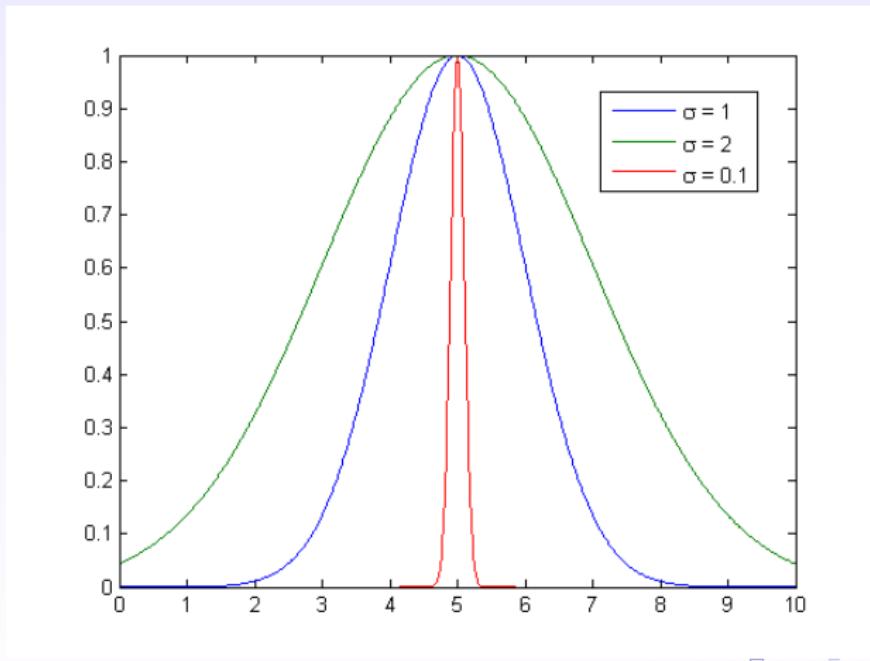
$$c = 5, \quad \sigma = 1$$



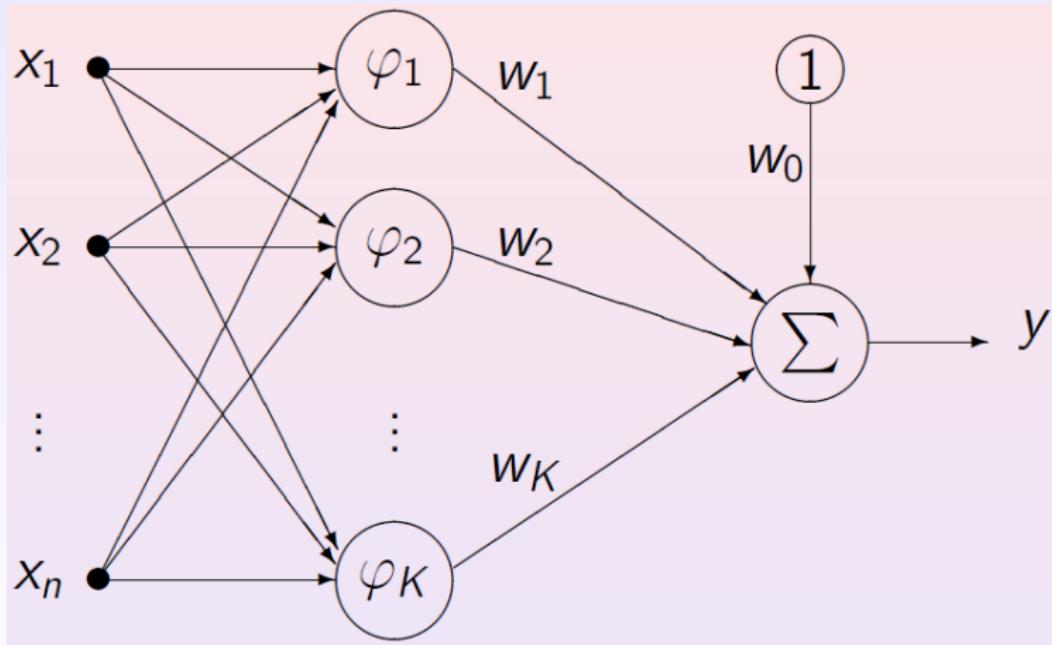
$$\mathbf{c} = [4 \ 6]^T, \quad \sigma = 1$$

# Funkcja Gaussa

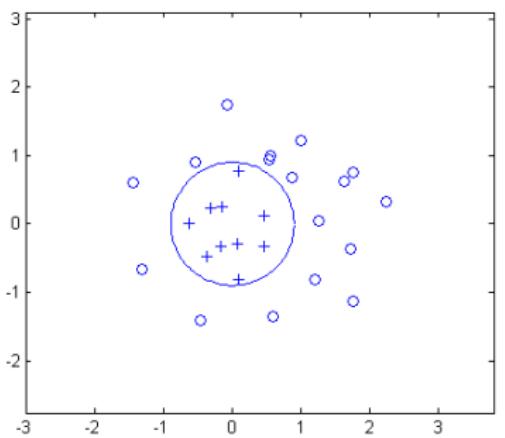
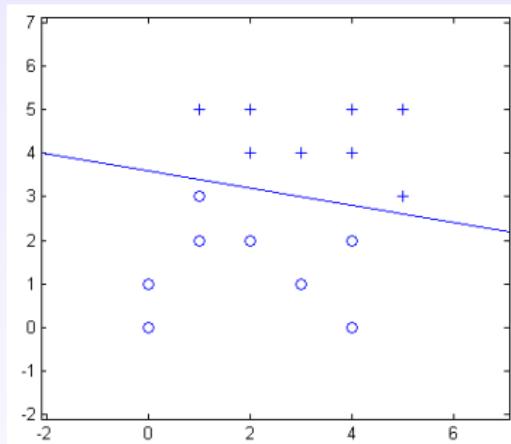
$$\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2\sigma^2}\right)$$



# Budowa sieci RBF



# Działanie sieci RBF



W zadaniach klasyfikacji neuron RBF dzieli przestrzeń wejścia za pomocą okręgu (dla 2 wejść), sfery (dla 3 wejść) bądź hipersfery.

# Uczenie sieci RBF

Sieci RBF są uczone pod nadzorem. Przed uczeniem, część wejściowa danych uczących powinna być znormalizowana.

- 1 Przyjmujemy ilość  $K$  neuronów radialnych na warstwie ukrytej.
- 2 Dobieramy środki  $\mathbf{c}_i$  i szerokości  $\sigma_i$  neuronów radialnych.
- 3 Dobieramy wagi  $w_j$  neuronów na warstwie wyjściowej.

# Dobór środków $\mathbf{c}_i$ i szerokości $\sigma_i$

W najprostszym przypadku środki neuronów radialnych mogą być wybrane losowo (pożądany jest ich równomierny rozkład w przestrzeni wejścia).

- Szerokości neuronów można przyjąć jednakowe i równe:

$$\sigma = \frac{d}{\sqrt{2K}}$$

gdzie:  $d$  – maksymalna odległość między centrami.

- Inny sposób to przyjęcie  $\sigma_i$  równego średniemu odchyleniu standardowemu odległości próbek sąsiednich od centrum.
- Można też za wartość  $\sigma_i$  przyjąć odległość centrum  $\mathbf{c}_i$  od najbliższego centrum sąsiedniego.
- Najlepszy efekt daje podział danych na uczące i walidujące i zastosowanie walidacji.

# Dobór środków $\mathbf{c}_i$ – klasteryzacja

Algorytm K-środków:

- 1 Losowo generujemy  $K$  punktów w przestrzeni wejścia (pożądany jest równomierny rozkład punktów w przestrzeni wejścia). Punkty te są początkowymi środkami klastrów  $\mathbf{c}_i$ .
- 2 Każdą próbę przyporządkowujemy do najbliższego środka  $\mathbf{c}_i$ .
- 3 Obliczamy nowe środki klastrów:

$$\mathbf{c}_i^{new} = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{x}_j^{(i)}$$

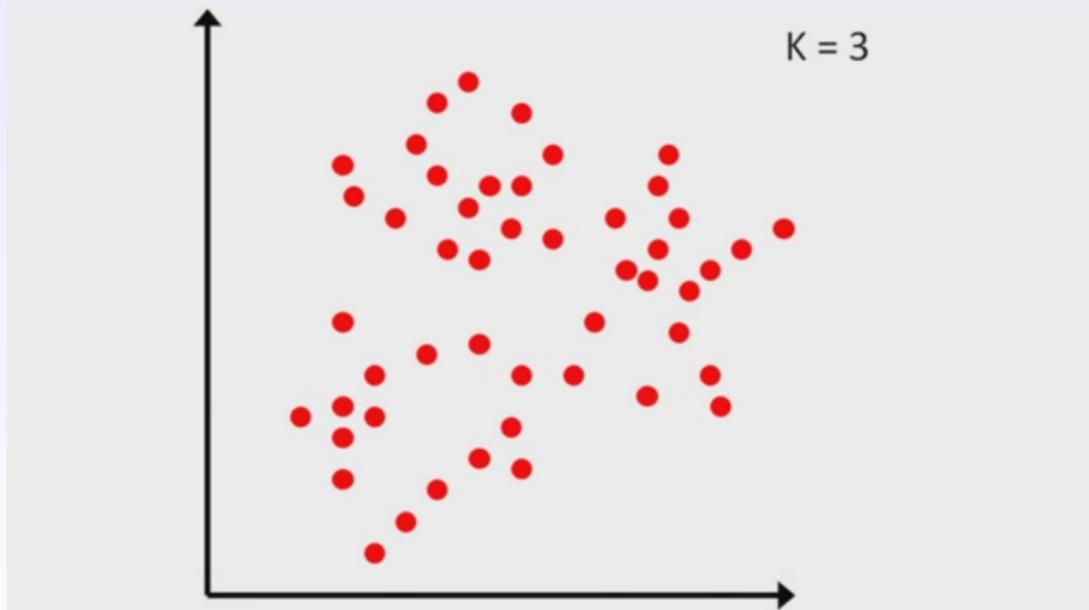
gdzie:  $\mathbf{x}_j^{(i)}$  – próbka nr  $j$  należąca do klastra  $i$ ,  $N_i$  – ilość próbek należących do klastra  $i$ .

- 4 Sprawdzamy przemieszczenia środków:

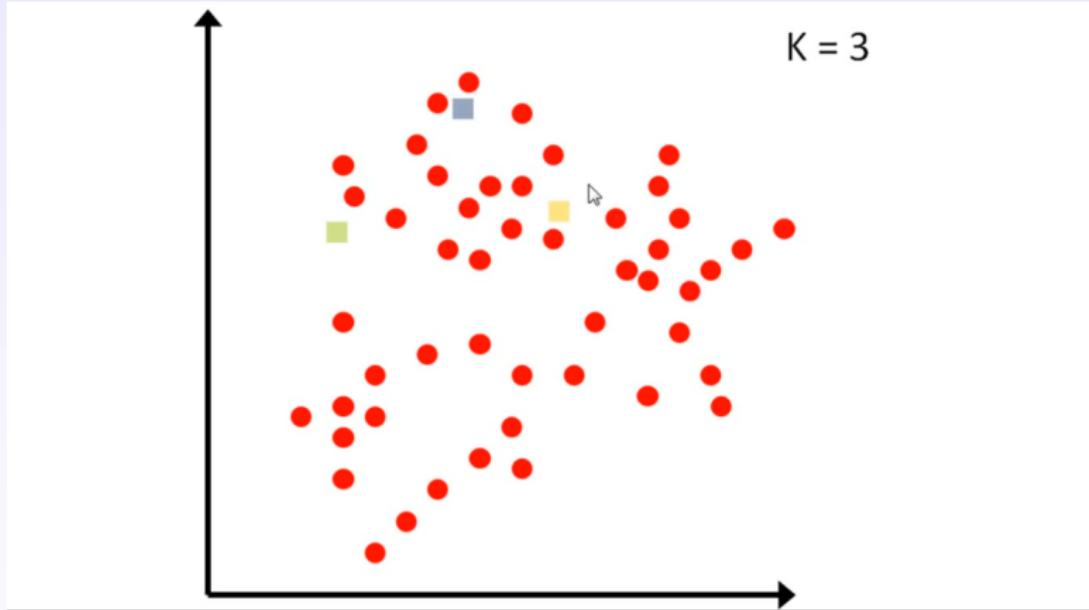
$$\Delta_i = \|\mathbf{c}_i^{new} - \mathbf{c}_i\|$$

- 5 Jeśli  $\max \Delta_i > \epsilon$  wracamy do punktu 2, w przeciwnym wypadku algorytm kończymy.

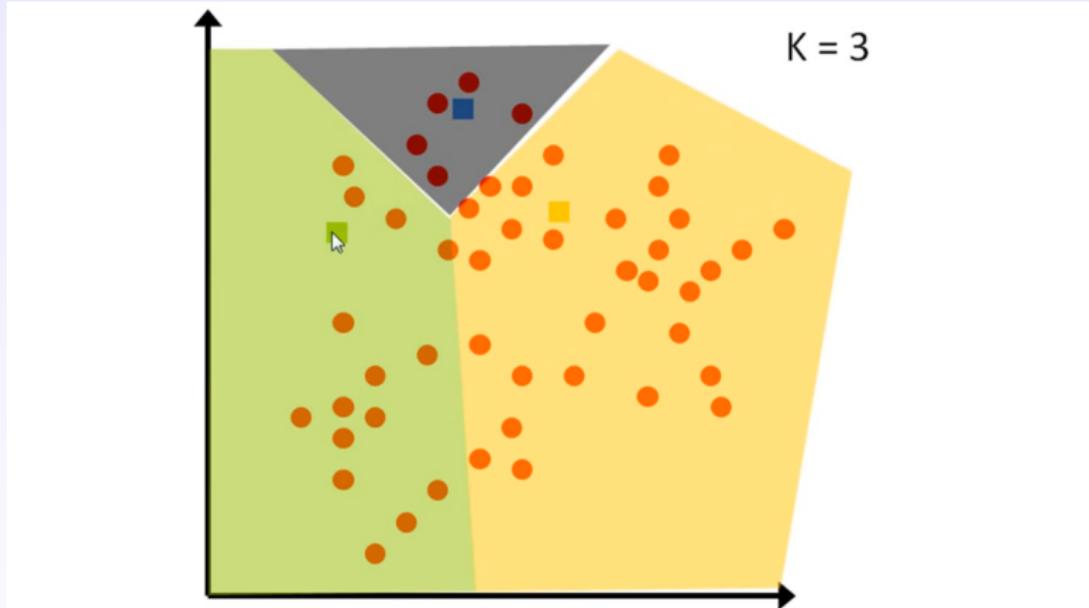
# Algorytm K-środków – przykład



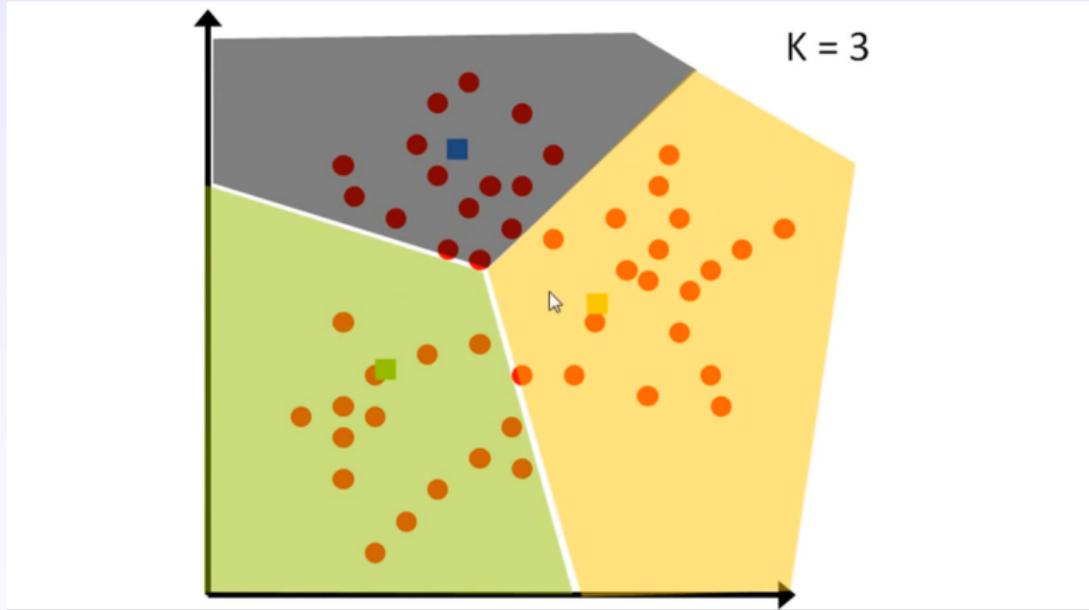
# Algorytm K-środków – przykład



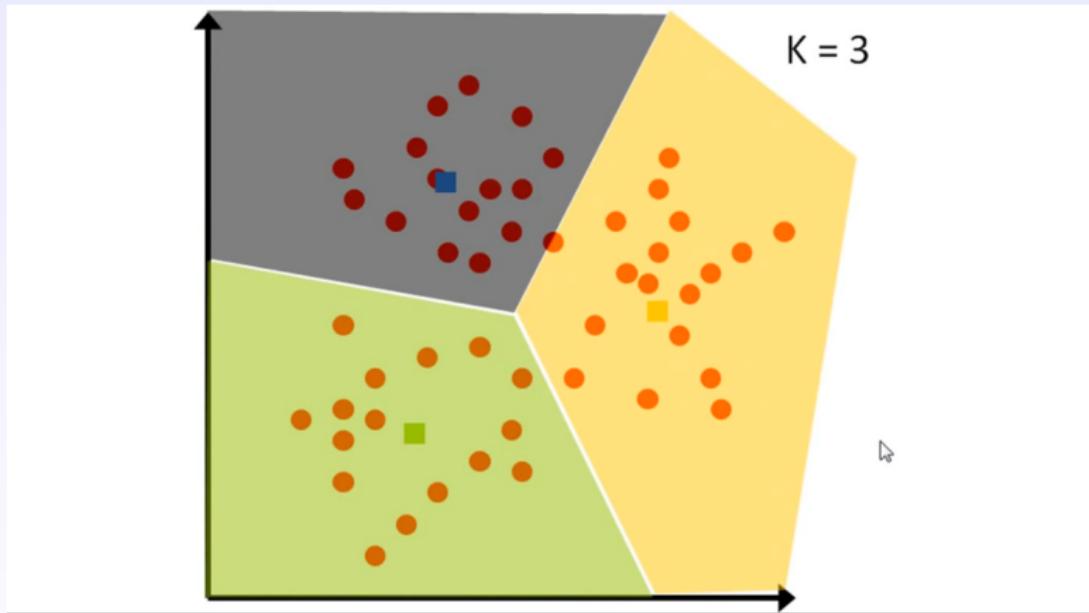
# Algorytm K-środków – przykład



# Algorytm K-środków – przykład



# Algorytm K-środków – przykład



# Dobór wag na warstwie wyjściowej

- 1 Mając dobrane środki i szerokości neuronów radialnych wagi można nauczyć w oparciu o regułę delta.
- 2 Wagi można też obliczyć w taki sposób aby minimalizować średni błąd kwadratowy modelu.

# Obliczenia wag

Dla każdej próbki  $p$  możemy utworzyć równanie:

$$w_0 + w_1 \cdot \phi(||\mathbf{x}^{(p)} - \mathbf{c}_1||) + \dots + w_K \cdot \phi(||\mathbf{x}^{(p)} - \mathbf{c}_K||) = d^{(p)}$$

Zakładając, że mamy  $L$  próbek, otrzymujemy w ten sposób układ  $L$  równań, w którym występuje  $K + 1$  niewiadomych  $w_i$ .

Układ ten można zapisać jako:

$$\mathbf{G} \cdot \mathbf{w} = \mathbf{d}$$

gdzie:

$$\mathbf{G} = \begin{bmatrix} 1 & \phi(||\mathbf{x}^{(1)} - \mathbf{c}_1||) & \dots & \phi(||\mathbf{x}^{(1)} - \mathbf{c}_K||) \\ \vdots & & & \\ 1 & \phi(||\mathbf{x}^{(p)} - \mathbf{c}_1||) & \dots & \phi(||\mathbf{x}^{(p)} - \mathbf{c}_K||) \\ \vdots & & & \\ 1 & \phi(||\mathbf{x}^{(L)} - \mathbf{c}_1||) & \dots & \phi(||\mathbf{x}^{(L)} - \mathbf{c}_K||) \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_K \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} d^{(1)} \\ \vdots \\ d^{(p)} \\ \vdots \\ d^{(L)} \end{bmatrix}$$

# Obliczenia wag

Rozwiążanie minimalizujące średni błąd kwadratowy modelu wyznaczamy jako:

$$\mathbf{w} = \mathbf{G}^+ \cdot \mathbf{d} = (\mathbf{G}^T \cdot \mathbf{G})^{-1} \cdot \mathbf{G}^T \cdot \mathbf{d}$$

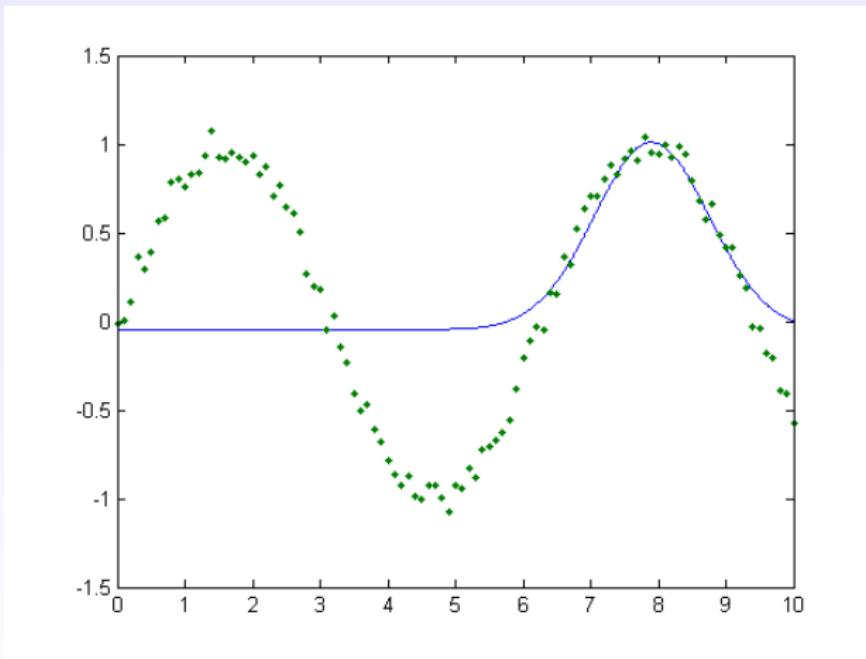
# Dobór ilości neuronów radialnych

- 1 Dzielimy dane na część uczącą i walidującą.
- 2 Tworzymy sieci neuronowe kolejno z 1 neuronem radialnym, potem z 2 neuronami i tak dalej.
- 3 Obserwujemy błąd zbioru uczącego i walidującego.
- 4 Jeśli błąd dla zbioru walidującego zaczyna rosnąć, przerywamy dodawanie neuronów.

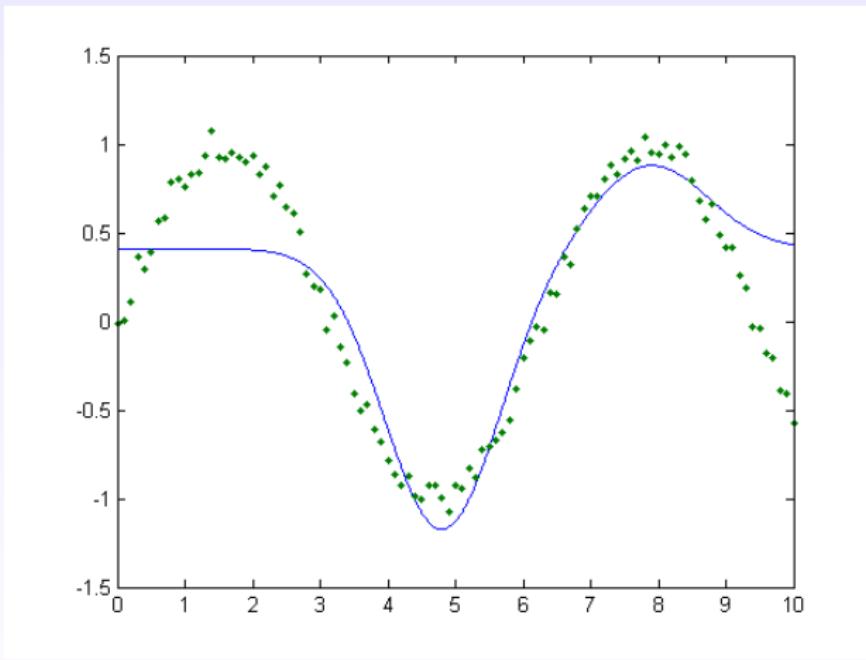
# Metoda newrb

- 1 Zakładamy szerokość neuronów RBF (jednakową dla wszystkich).
- 2 Zakładamy minimalny średni kwadratowy błąd modelu.
- 3 Tworzymy sieć z jednym neuronem RBF.
- 4 Wyznaczamy błąd dla każdej próbki.
- 5 Liczymy średni błąd kwadratowy (MSE) modelu.
- 6 Jeśli MSE jest większy niż założone minimum, dodajemy kolejny neuron RBF umieszczając jego centrum w próbce powodującej największy błąd. Wyznaczamy wagę na wyjściu i wracamy do punktu 4.  
W przeciwnym przypadku kończymy uczenie.

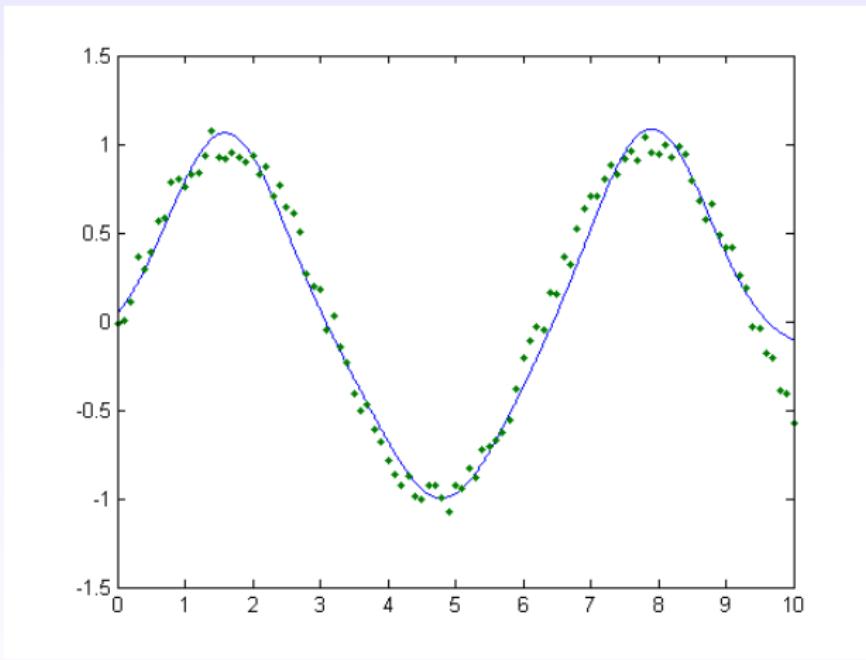
# Metoda newrb – przykład: $\sigma = 1$



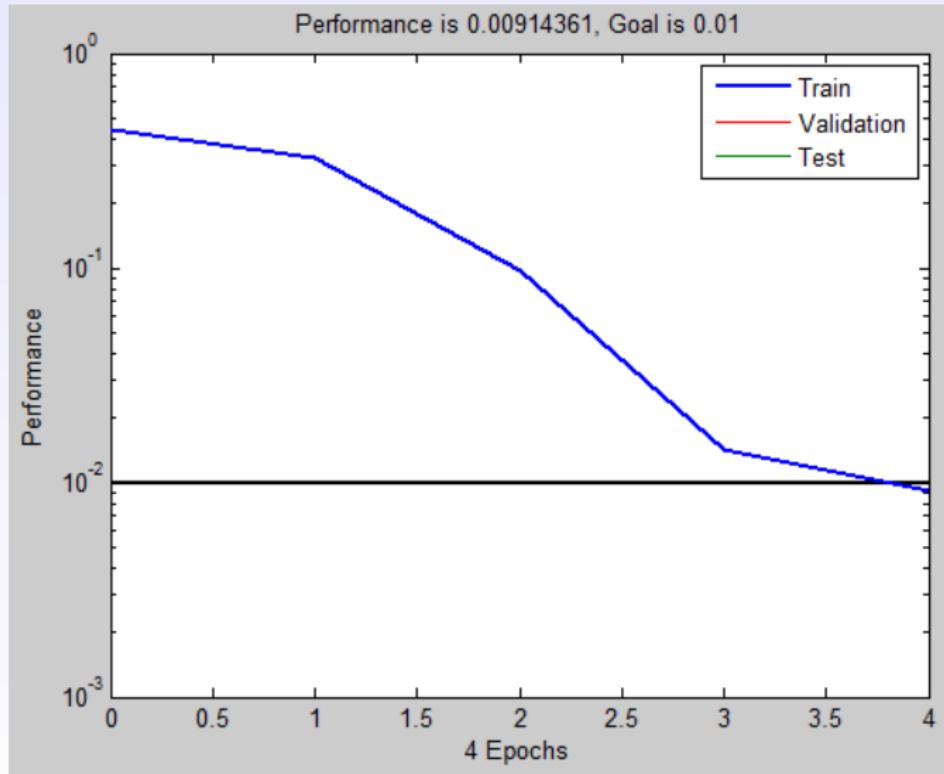
# Metoda newrb – przykład: $\sigma = 1$



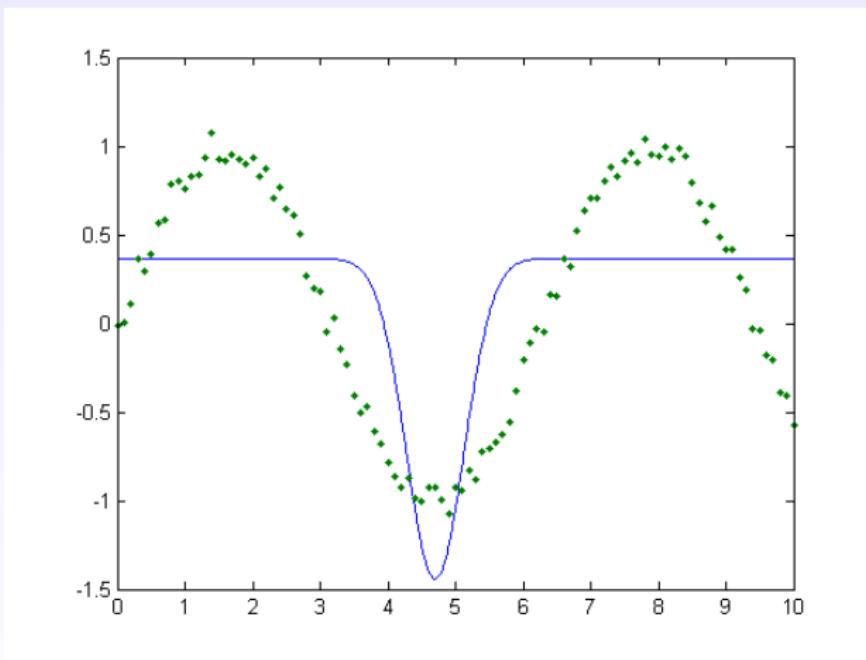
# Metoda newrb – przykład: $\sigma = 1$



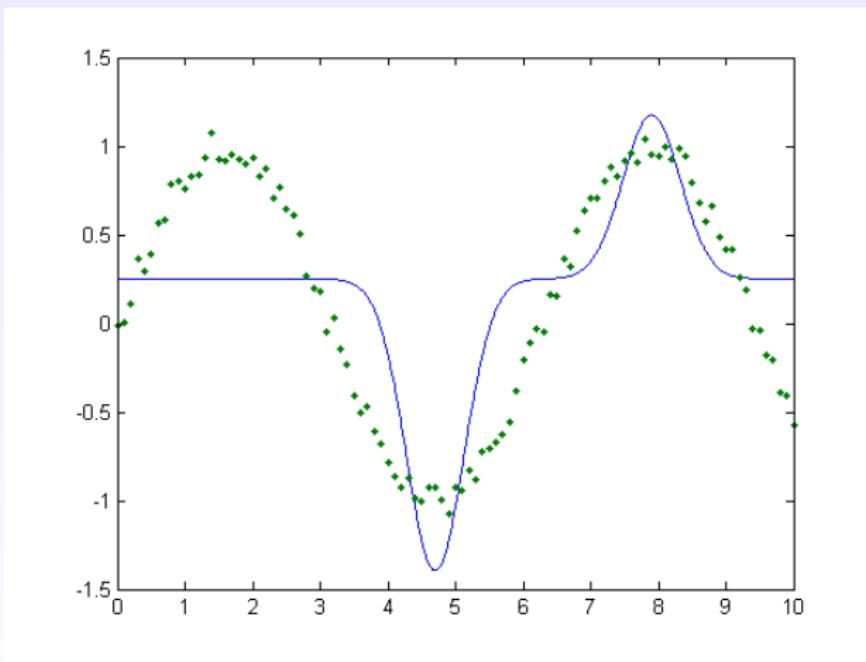
# Metoda newrb – przykład: $\sigma = 1$



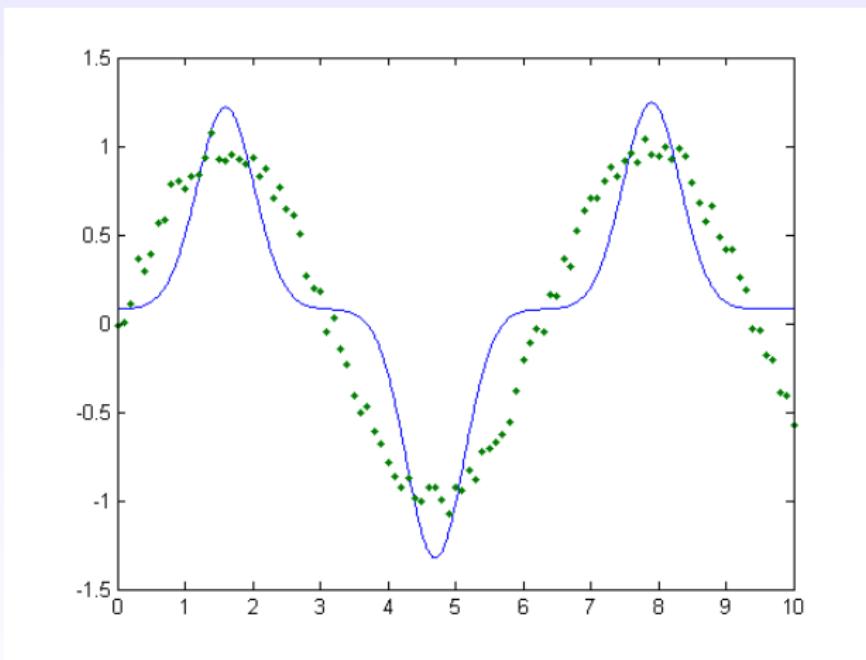
# Metoda newrb – przykład: $\sigma = 0.5$



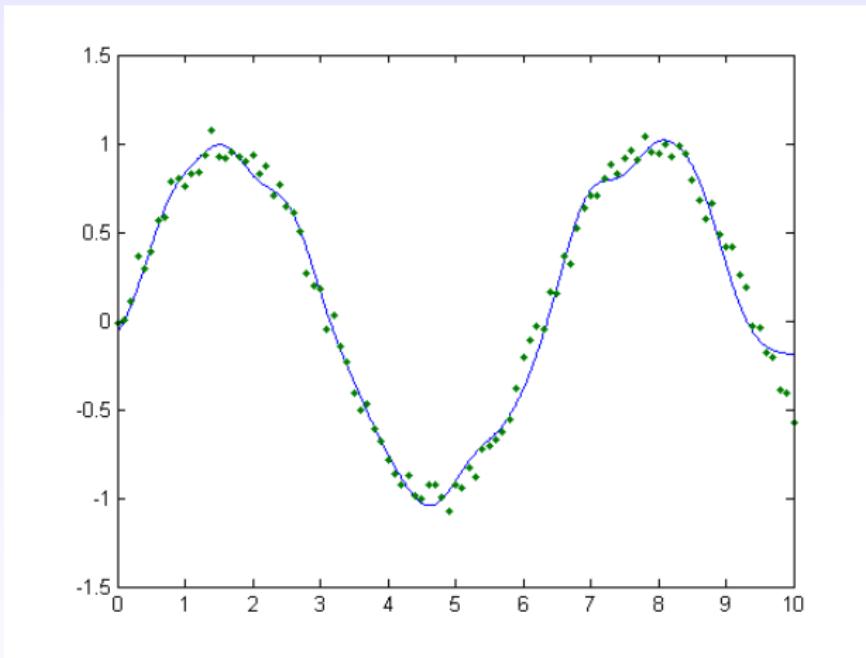
# Metoda newrb – przykład: $\sigma = 0.5$



# Metoda newrb – przykład: $\sigma = 0.5$

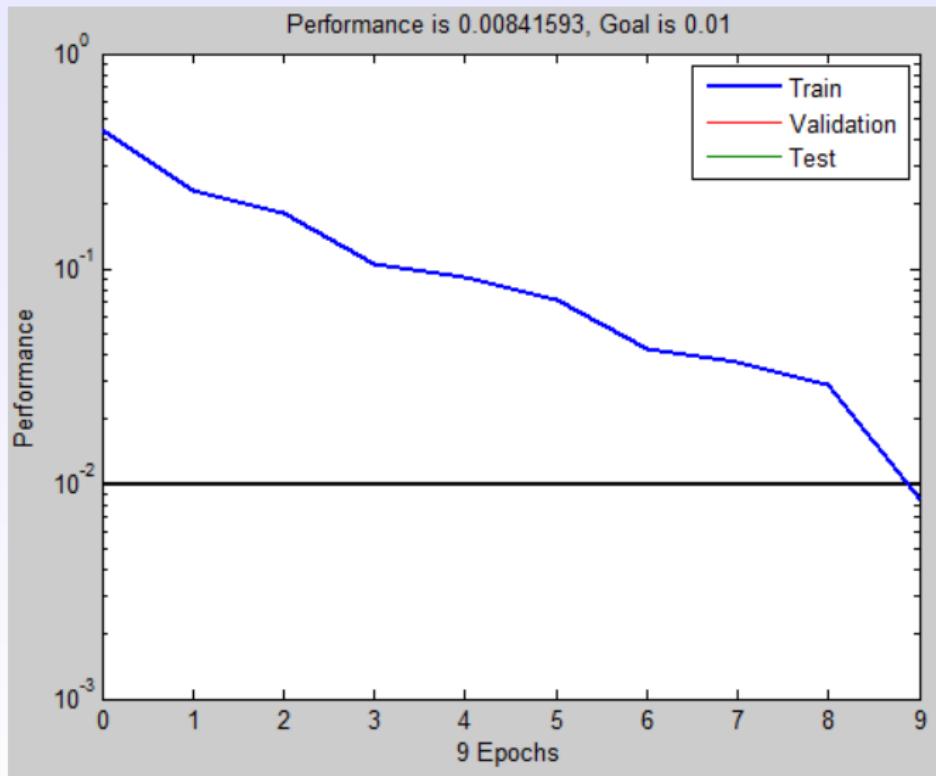


# Metoda newrb – przykład: $\sigma = 0.5$



Charakterystyka dla sieci z 9 neuronami RBF.

# Metoda newrb – przykład: $\sigma = 0.5$



# Metoda newrbe

- 1 Zakładamy szerokość neuronów RBF (jednakową dla wszystkich).
- 2 Przyjmujemy ilość neuronów RBF równą ilości próbek. Centrum każdego neuronu lokujemy w próbkach.
- 3 Wyznaczamy wagi na wyjściu.

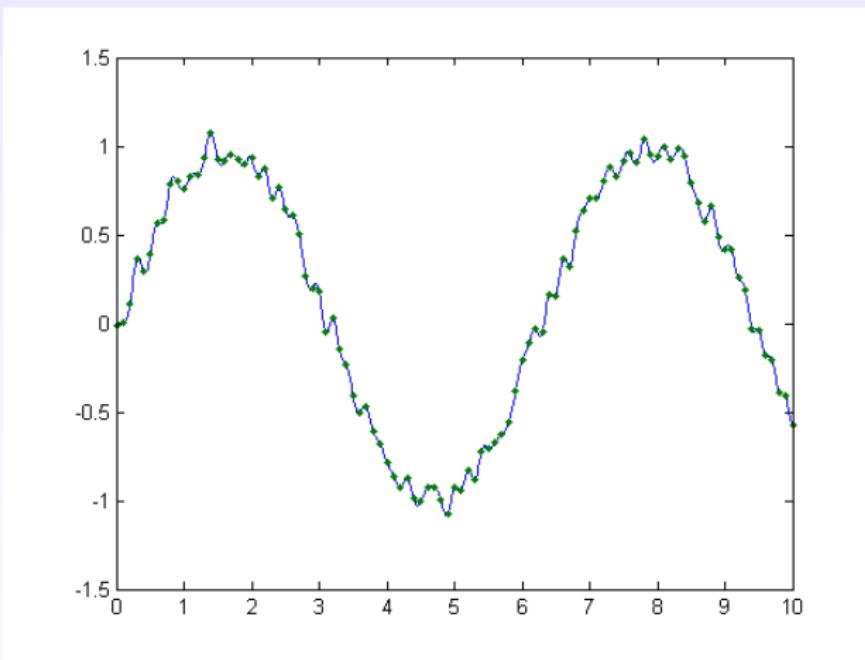
# Metoda newrb

- 1 Zakładamy szerokość neuronów RBF (jednakową dla wszystkich).
- 2 Przyjmujemy ilość neuronów RBF równą ilości próbek. Centrum każdego neuronu lokujemy w próbkach.
- 3 Wyznaczamy wagi na wyjściu.

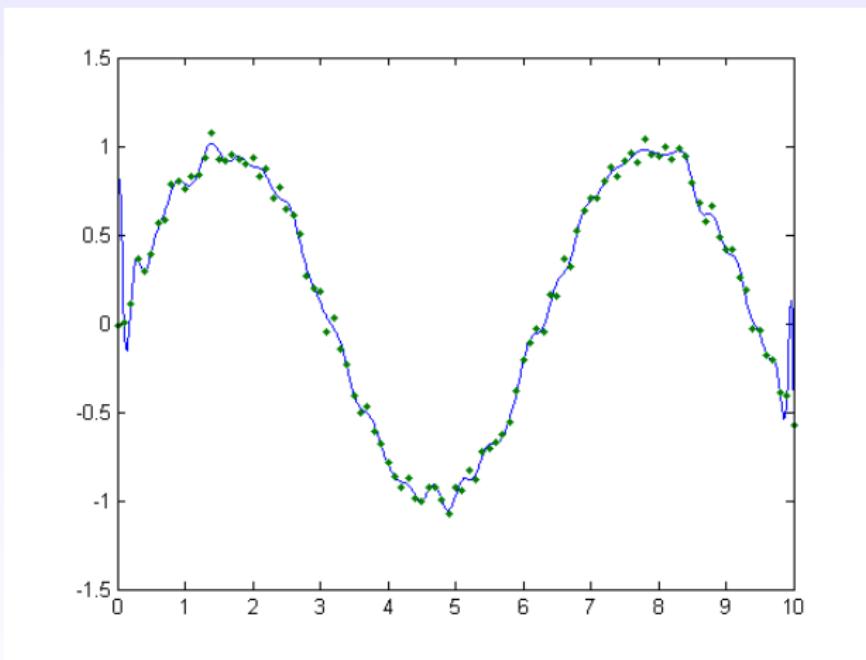
- W sieci dobieramy tylko **jeden (!)** parametr. Reszta jest zdefiniowana przez dane (centra) lub obliczana (wagi).
- Sieć bardzo łatwo przeuczyć. Dla bardzo małych wartości  $\sigma$  błąd danych uczących spada prawie do zera.
- Wartość  $\sigma$  **koniecznie** dobieramy metodą walidacji. Jeśli danych jest dużo, dzielimy je na część uczącą i walidującą. Jako  $\sigma$  przyjmujemy wartość, która minimalizuje błąd danych walidujących.
- Jeśli danych mamy mało – stosujemy kroswalidację (walidację krzyżową).



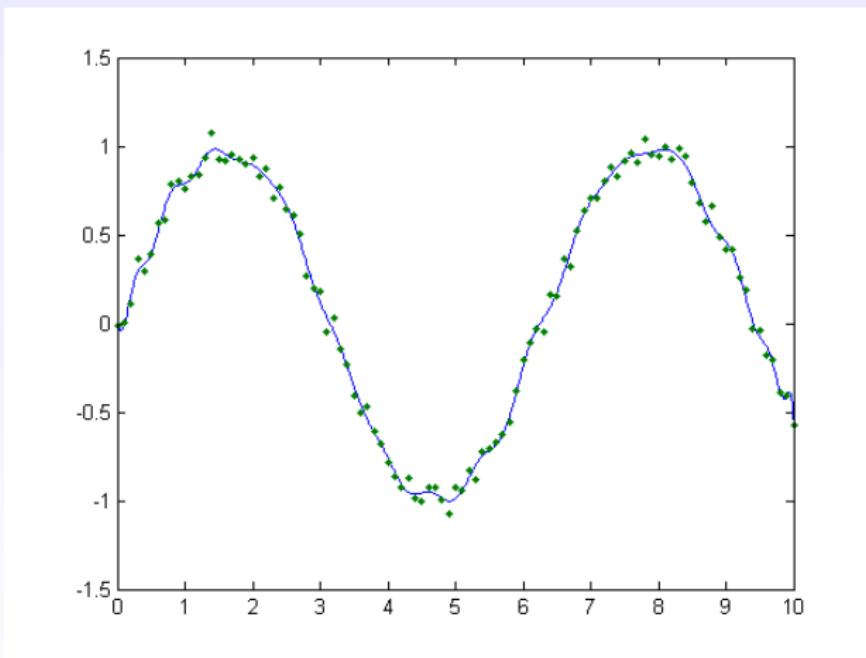
# Metoda newrbe – przykład: $\sigma = 0.1$



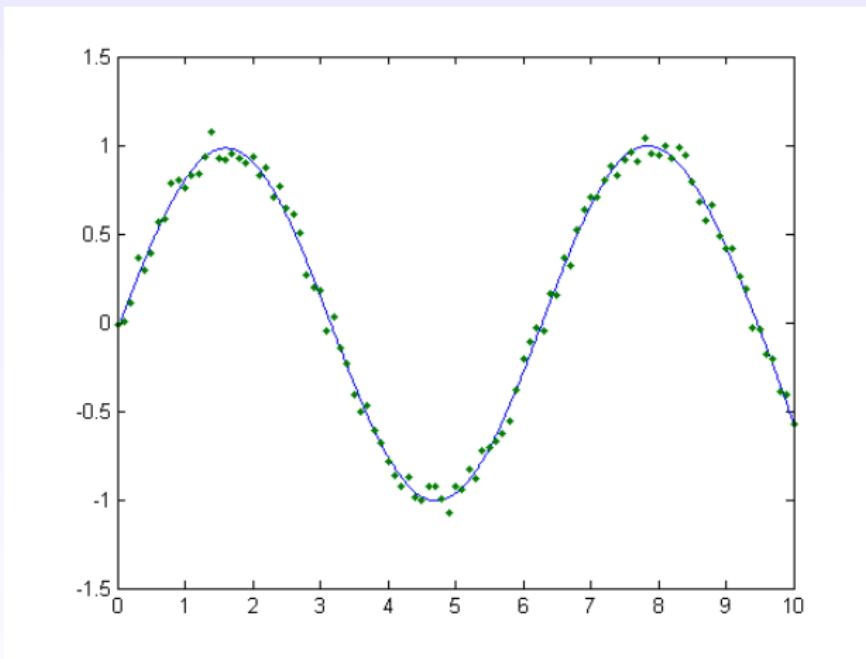
# Metoda newrbe – przykład: $\sigma = 0.5$



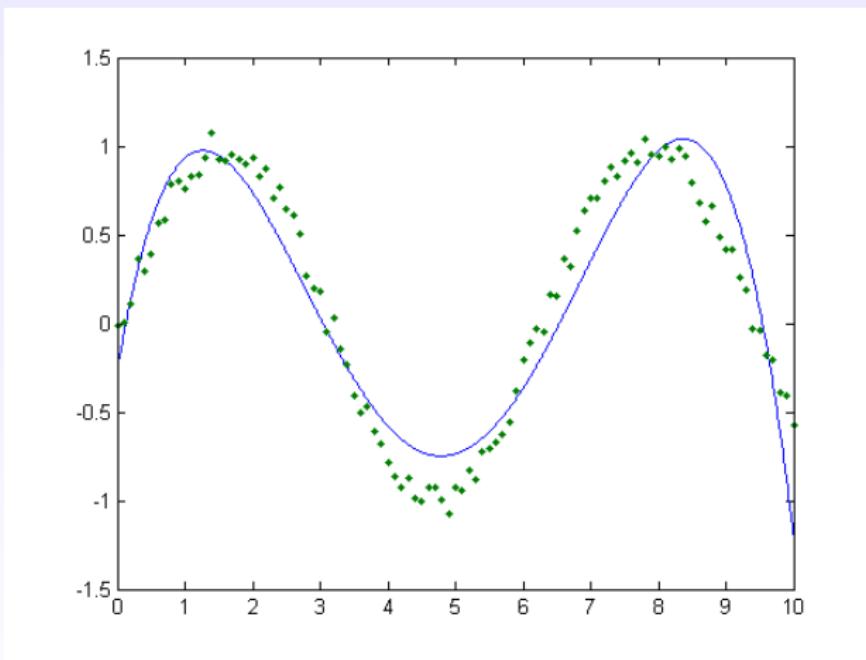
# Metoda newrbe – przykład: $\sigma = 1$



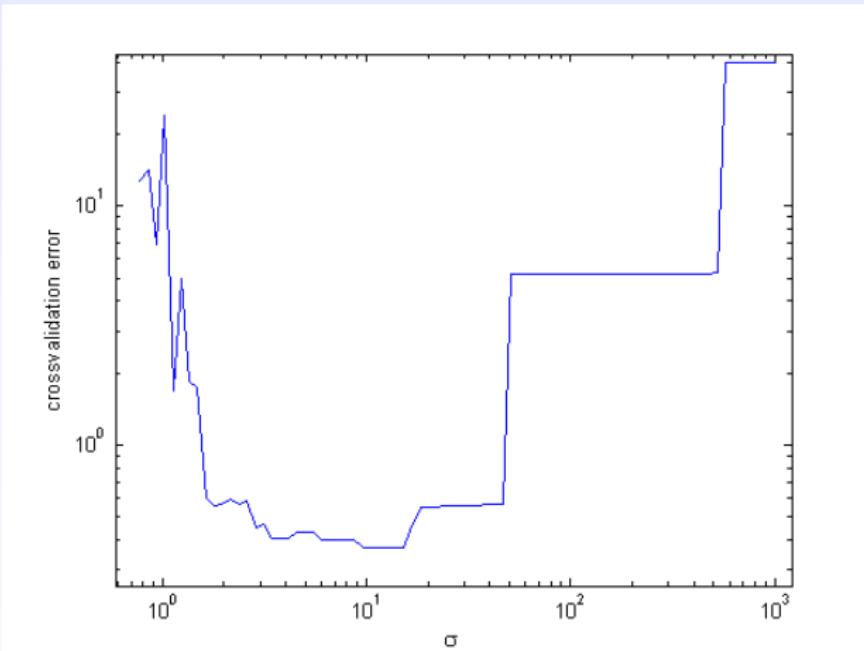
# Metoda newrbe – przykład: $\sigma = 10$



# Metoda newrbe – przykład: $\sigma = 100$



# Metoda newrb – błąd walidacji



Błąd walidacji krzyżowej wyznaczony metodą ‘minus jednego elementu’ dla różnych wartości  $\sigma$ .

# Probabilistyczna sieć neuronowa typu RBF (PRBF)

W sieciach PRBF przyjmujemy następujące założenia:

- 1 ilość neuronów w warstwie ukrytej jest równa ilości próbek uczących  $L$ ,
- 2 środki neuronów RBF pokrywają się z próbami uczącymi,
- 3 szerokości wszystkich neuronów RBF są takie same.

Dzięki powyższym założeniom w sieci mamy do nauczenia tylko **jeden (!)** parametr.

W MATLAB-ie sieci PRBF można tworzyć za pomocą polecenia `newpnn`.

Sieci PRBF wykorzystujemy do klasyfikacji.

# Budowa sieci PRBF

Przyjmijmy, że każda próbka ucząca ma następującą postać:

$$x_{p1}, x_{p2}, \dots, x_{pj}, \dots, x_{pn}, \text{ class}_k$$

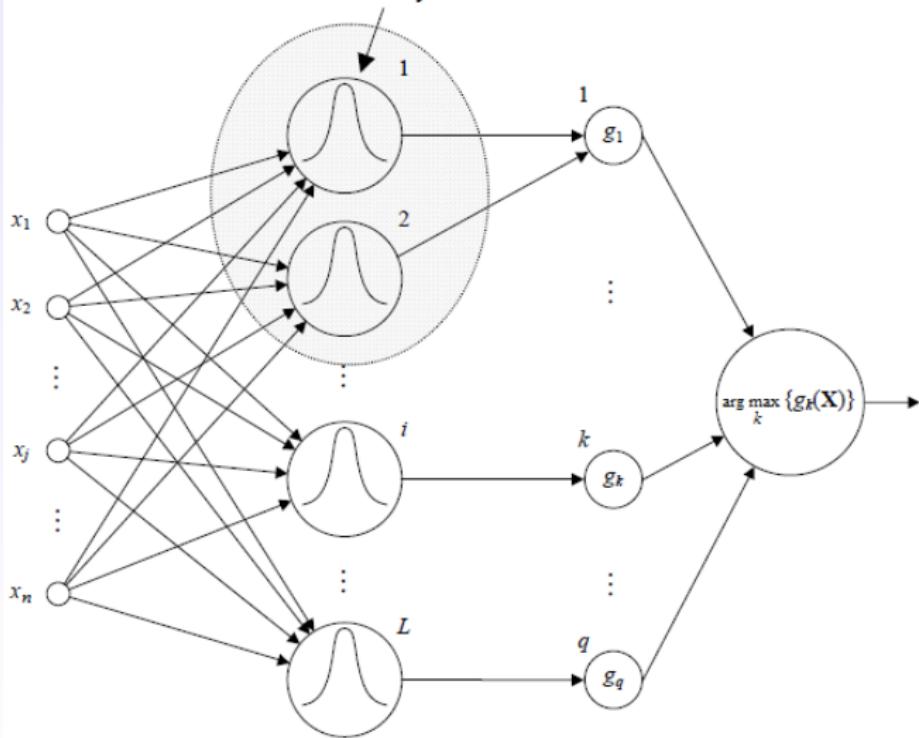
gdzie:

- $p = 1 \dots L$  – to numer próbki ( $L$  – ilość próbek),
- $j = 1 \dots n$  – to numer wejścia ( $n$  – ilość wejść),
- $k = 1 \dots q$  – to numer klasy, do której należy próbka ( $q$  – ilość klas).

Wejścia danych wykorzystywanych przy tworzeniu sieci PRBF powinny być znormalizowane.

# Budowa sieci PRBF

Neurony RBF związane z próbками należącymi do klasy 1



# Działanie sieci PRBF

Podczas klasyfikacji próbki  $\mathbf{x}$  w sieci PRBF obliczane są dla każdej klasy wartości funkcji  $g_k(\mathbf{x})$ , które stanowią estymaty funkcji gęstości prawdopodobieństwa przynależności próbki  $\mathbf{x}$  do klasy  $k$ .  
Estymaty takie można opisać wzorem:

$$\tilde{g}_k(\mathbf{x}) = \frac{1}{m_k \cdot \sigma_k^n \cdot (2\pi)^{n/2}} \cdot \sum_{i=1}^{m_k} \exp \left[ -\frac{\|\mathbf{x} - \mathbf{x}_i^{(k)}\|^2}{2\sigma_k^2} \right]$$

gdzie:

- $m_k$  – ilość próbek ze zbioru danych, należących do klasy  $k$ ,
- $\mathbf{x}_i^{(k)}$  – próbki ze zbioru danych, należące do klasy  $k$ ,
- $\sigma_k$  – szerokość funkcji Gaussa dla klasy  $k$ .

# Działanie sieci PRBF

Założymy jednakową szerokość wszystkich funkcji Gaussa  $\sigma$ . Ponieważ chcemy tylko porównać ze sobą wartości funkcji  $g_k(\mathbf{x})$  można je zapisać w sposób uproszczony jako:

$$g_k(\mathbf{x}) = \frac{1}{m_k} \cdot \sum_{i=1}^{m_k} \exp \left[ -\frac{\|\mathbf{x} - \mathbf{x}_i^{(k)}\|^2}{2\sigma^2} \right]$$

Jako wynik swojego działania, sieć zwraca numer klasy, dla której wartość  $g_k(\mathbf{x})$  (i tym samym prawdopodobieństwo zaklasyfikowania) jest największa.

# Uczenie sieci PRBF

- W sieci dobieramy tylko **jeden (!)** parametr. Reszta jest zdefiniowana przez dane (centra neuronów RBF).
- Sieć bardzo łatwo przeuczyć. Dla bardzo małych wartości  $\sigma$  błąd klasyfikacji danych uczących spada prawie do zera.
- Wartość  $\sigma$  **koniecznie** dobieramy metodą walidacji. Jeśli danych jest dużo, dzielimy je na część uczącą i walidującą. Jako  $\sigma$  przyjmujemy wartość, która minimalizuje błąd danych walidujących.
- Jeśli danych mamy mało – stosujemy kroswalidację (walidację krzyżową).

# Generalised Regression Network (GRN)

W sieciach GRN przyjmujemy podobne założenia jak w sieci PRBF:

- ① ilość neuronów w warstwie ukrytej jest równa ilości próbek uczących  $L$ ,
- ② środki neuronów RBF pokrywają się z próbami uczącymi,
- ③ szerokości wszystkich neuronów RBF są takie same.

Dzięki powyższym założeniom w sieci mamy do nauczenia tylko **jeden (!) parametr**.

W MATLAB-ie sieci GRN można tworzyć za pomocą polecenia `newgrnn`.

Sieci GRN wykorzystujemy do aproksymacji (regresji).

# Budowa sieci GRN

Przyjmijmy, że każda próbka ucząca ma następującą postać:

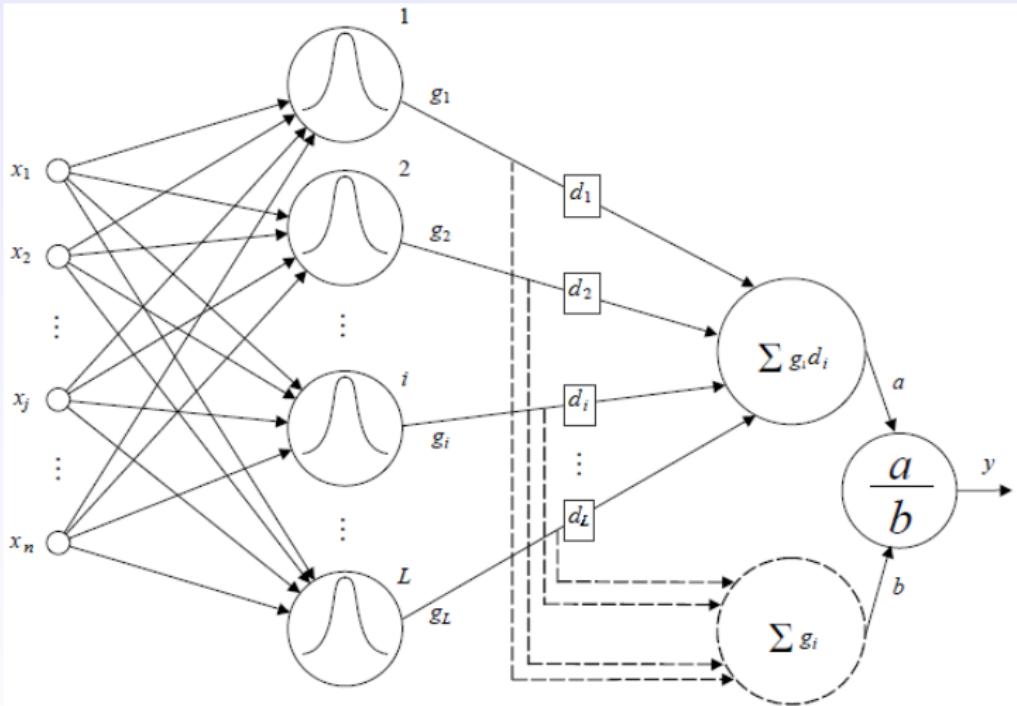
$$x_{p1}, x_{p2}, \dots, x_{pj}, \dots, x_{pn}, d_p$$

gdzie:

- $p = 1 \dots L$  – to numer próbki ( $L$  – ilość próbek),
- $j = 1 \dots n$  – to numer wejścia ( $n$  – ilość wejść),
- $d_p$  – zadane wyjście modelu dla próbki  $p$ .

Wejścia danych wykorzystywanych przy tworzeniu sieci GRN powinny być znormalizowane.

# Budowa sieci GRN



# Działanie sieci GRN

Wagi neuronu wyjściowego równe są wyjściom zadanym!

Sieć oblicza odpowiedź zgodnie z wzorem:

$$y(\mathbf{x}) = \frac{\sum_{p=1}^L d_p \cdot g_p}{\sum_{p=1}^L g_p} = \frac{\sum_{p=1}^L d_p \cdot \exp\left[-\frac{\|\mathbf{x} - \mathbf{x}_p\|^2}{2\sigma^2}\right]}{\sum_{p=1}^L \exp\left[-\frac{\|\mathbf{x} - \mathbf{x}_p\|^2}{2\sigma^2}\right]}$$

gdzie:

- $\mathbf{x}_p$  – wektor wejść dla próbki nr  $p$ ,
- $d_p$  – wyjście zadane dla próbki nr  $p$ .

# Uczenie sieci GRN

- W sieci dobieramy tylko **jeden (!)** parametr. Reszta jest zdefiniowana przez dane (centra neuronów RBF i wagi neuronu wyjściowego).
- Sieć bardzo łatwo przeuczyć. Dla bardzo małych wartości  $\sigma$  błąd modelu obliczony dla danych uczących spada prawie do zera.
- Wartość  $\sigma$  **koniecznie** dobieramy metodą walidacji. Jeśli danych jest dużo, dzielimy je na część uczącą i walidującą. Jako  $\sigma$  przyjmujemy wartość, która minimalizuje błąd danych walidujących.
- Jeśli danych mamy mało – stosujemy kroswalidację (walidację krzyżową).

# Metody uczenia sieci

## Uczenie nadzorowane

Zbiór uczący to zbiór par:

$$(\mathbf{x}_i, d_i), \quad i = 1 \dots L,$$

gdzie  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in} \in \mathbb{R}^n)$  są danymi wejściowymi a  $d_i \in \{0, 1\}$  lub  $d_i \in \mathbb{R}$  są skojarzonymi z nimi zadanymi wyjściami.

## Uczenie nienadzorowane

Zbiór uczący to zbiór:

$$(\mathbf{x}_i), \quad i = 1 \dots L,$$

gdzie  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in} \in \mathbb{R}^n)$  są danymi wejściowymi.

# Uczenie nienadzorowane

- W próbkach nie jest wymagany ‘wzorcowy’ wektor oczekiwanej odpowiedzi.
- Sieć powinna ‘odkryć’ bez zewnętrznej pomocy wzory, cechy, wzajemne zależności, uporządkowanie danych wejściowych, a następnie podać te informacje w odpowiednio zakodowanej postaci na wyjściu.
- Do skutecznego uczenia nienadzorowanego wymagana jest redundancja (nadmiarowość) danych.
- W trakcie uczenia sieć zwykle stara się podzielić zbiór próbek uczących na klasy, według pewnych cech wspólnych. Sieć powinna być zdolna do zidentyfikowania takich cech w dowolnym prezentowanym wektorze wejściowym.

# Uczenie nienadzorowane – zadania

Od struktury sieci oraz od metody jej uczenia zależy rodzaj rozwiązywanego przez nią zadania.

- Badanie podobieństwa – sieć z jednym neuronem wyjściowym, którego wartość informuje na ile podobna jest próbka wejściowa do wzorca zapamiętanego podczas uczenia.
- Klasyfikacja – sieć posiada ilość neuronów wyjściowych równą ilości rozpoznawanych klas. Próbka wejściowa zostaje przyporządkowana do określonej klasy. Zadaniem procesu uczenia jest podział próbek uczących na klasy próbek zbliżonych do siebie i przyporządkowanie każdej klasie jednego neuronu wyjściowego.
- Poszukiwanie pierwotwzoru – sieć spełnia podobną rolę jak przy klasyfikacji, z tą różnicą, że na wyjściu otrzymujemy wzorzec typowy dla danej klasy.

# Uczenie nienadzorowane – zadania

Od struktury sieci oraz od metody jej uczenia zależy rodzaj rozwiązywanego przez nią zadania.

- Kodowanie – wektor wyjściowy sieci jest zakodowaną wersją wektora wejściowego, zachowując zawarte w nim najistotniejsze informacje. Przykładem może tu być kompresja danych.
- Analiza czynników głównych (*PCA – principal component analysis*) – sieć posiada kilka neuronów wyjściowych, z których każdy określa podobieństwo próbki wejściowej względem jednego z badanych czynników głównych.
- Tworzenie map cech – elementy warstwy wyjściowej są geometrycznie uporządkowane (np. w postaci tablicy 2-wymiarowej). Podczas prezentacji próbki wejściowej uaktywnia się tylko 1 wyjście. Idea działania zakłada, aby podobne próbki wejściowe generowały aktywność bliskich geometrycznie neuronów. Warstwa wyjściowa jest więc swego rodzaju mapą topograficzną cech danych wejściowych.

# Reguła Hebb'a

Pierwszą regułę uczenia neuronu podał Donald Hebb. Regułę swą oparł na zjawisku tworzenia się nabytych odruchów warunkowych u ludzi i zwierząt.

Jeżeli neuron A jest cyklicznie pobudzany przez neuron B, to staje się on jeszcze bardziej czuły na pobudzenie tego neuronu.

# Reguła Hebb'a

Pierwszą regułę uczenia neuronu podał Donald Hebb. Regułę swą oparł na zjawisku tworzenia się nabytych odruchów warunkowych u ludzi i zwierząt.

Jeżeli neuron A jest cyklicznie pobudzany przez neuron B, to staje się on jeszcze bardziej czuły na pobudzenie tego neuronu.

Jeśli przez  $\phi_A$  i  $\phi_B$  oznaczymy stany aktywności neuronów A i B, a przez  $w_{AB}$  – wagę ich połączenia synaptycznego, to powyższą regułę można zapisać w postaci zależności:

$$w_{AB}(k+1) = w_{AB}(k) + \eta \cdot \phi_A \cdot \phi_B$$

# Normalizacja danych

Skalowanie każdego wejścia w taki sposób, aby przyjmowało ono wartości z jednego, założonego przedziału  $[l, u]$ . Normalizacja najczęściej skaluje wartości do przedziału  $[0, 1]$  lub  $[-1, 1]$ .

$$x_{norm} = (x - x_{min}) \cdot \frac{u - l}{x_{max} - x_{min}} + l$$

gdzie:

- $x$  – wartość oryginalna,
- $x_{norm}$  – wartość po normalizacji,
- $x_{min}, x_{max}$  – minimalna i maksymalna wartość dla danych.

# Normalizacja danych

```
% normalizacja wektora x
% new_min, new_max - nowy zakres wartości wektora (domyślnie [0,1])
% x_norm - wektor znormalizowany
% min_x, max_x - minimalna i maksymalna wart. w oryginalnym wektorze

function [x_norm, min_x, max_x] = normalizuj(x, new_min, new_max)

if nargin < 2
    new_min = 0;
    new_max = 1;
end
if nargout > 1
    min_x = min(x);
    max_x = max(x);
end

x_norm = (x - min(x))*(new_max-new_min)/(max(x)-min(x)) + new_min;
```

```
>> x = 0:10
x =
    0     1     2     3     4     5     6     7     8     9    10

>> x1 = normalizuj(x)
x1 =
    0     0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.0

>> x2 = normalizuj(x,-10,10)
x2 =
   -10    -8    -6    -4    -2     0     2     4     6     8    10

>> [x2, x_low, x_high] = normalizuj(x,-100,100)
x2 =
   -100   -80   -60   -40   -20     0    20    40    60    80   100
x_low =
      0
x_high =
     10
```

# Normalizacja osi Z

W uczeniu nienadzorowanym częstym wymogiem jest jednakowa długość wektorów uczących.

# Normalizacja osi Z

W uczeniu nienadzorowanym częstym wymogiem jest jednakowa długość wektorów uczących.

- 1 W pierwszym kroku skalujemy wszystkie wejścia do przedziału  $[-1, 1]$ .
- 2 W drugim kroku dodajemy dodatkową zmienną, której wartość jest funkcją rzeczywistych wejść. Wartość ta jest tak dobrana, aby długość wektora po normalizacji była równa 1.

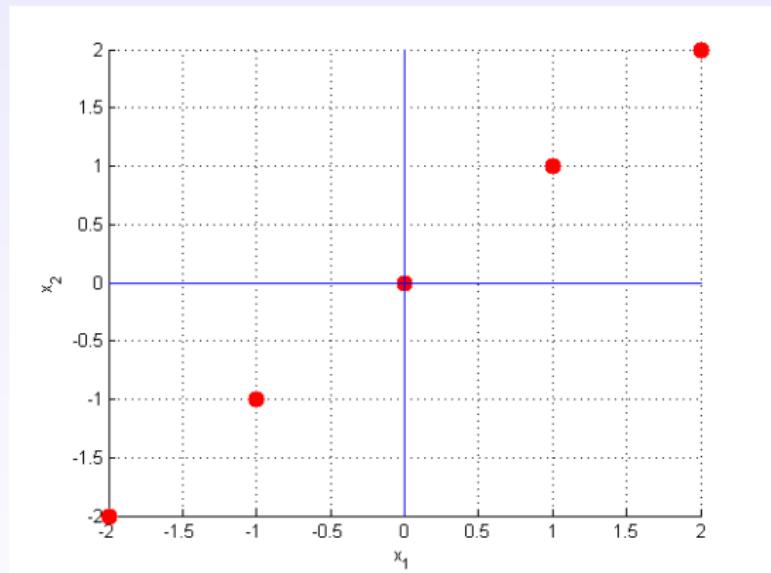
Obliczamy:

$$\hat{x}_i = f \cdot x_i, \quad i = 1 \dots n, \quad f = \frac{1}{\sqrt{n}}$$

$$\hat{x}_{n+1} = f \cdot \sqrt{n - d^2}, \quad d = \sqrt{\sum_{i=1}^n x_i^2}$$

# Normalizacja osi $Z$ – przykład

$x_1$	$x_2$
-2	-2
-1	-1
0	0
1	1
2	2



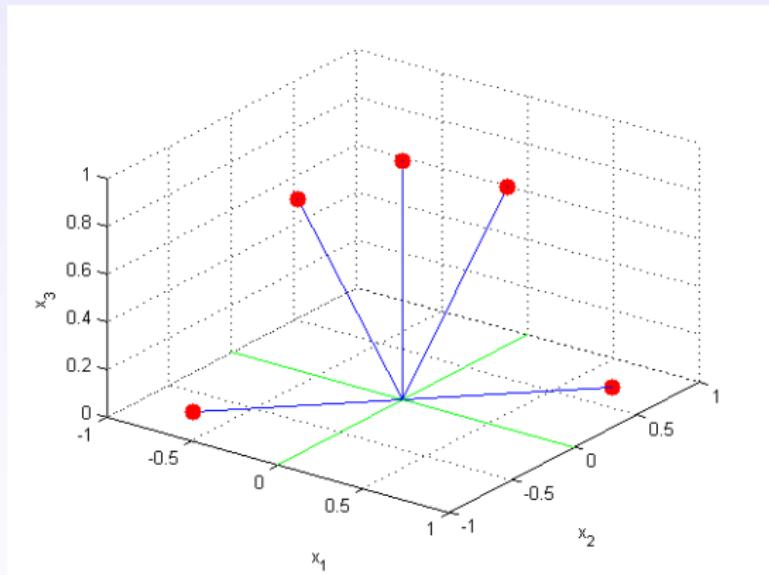
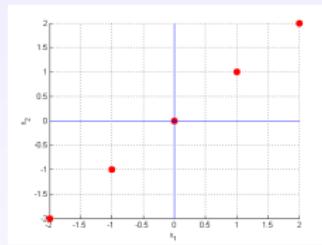
# Normalizacja osi $Z$ – przykład

$$\hat{x}_i = f \cdot x_i, \quad i = 1 \dots n, \quad f = \frac{1}{\sqrt{n}}$$

$$\hat{x}_{n+1} = f \cdot \sqrt{n - d^2}, \quad d = \sqrt{\sum_{i=1}^n x_i^2}$$

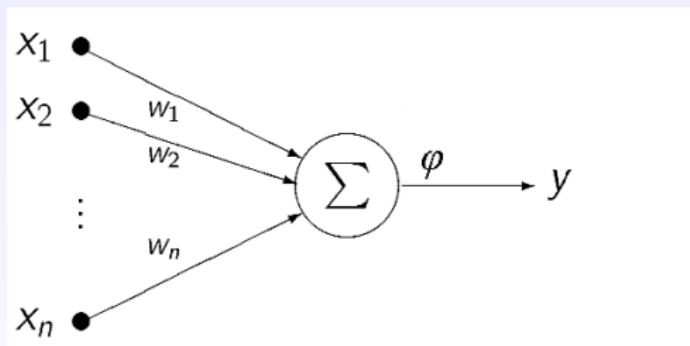
$x_1$	$x_2$	$x_1^{norm}$	$x_2^{norm}$	$d^2$	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$
-2	-2	-1	-1	2	$-\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$	0
-1	-1	-0.5	-0.5	$\frac{1}{2}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{\sqrt{3}}{2}$
0	0	0	0	0	0	0	1
1	1	0.5	0.5	$\frac{1}{2}$	$\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	$\frac{\sqrt{3}}{2}$
2	2	1	1	2	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	0

# Normalizacja osi $Z$ – przykład



# Reguła Grossberga

- Zakładamy, że mamy 1 neuron liniowy.
- Do dyspozycji mamy tylko 1 próbkę uczącą  $\mathbf{x}$ .

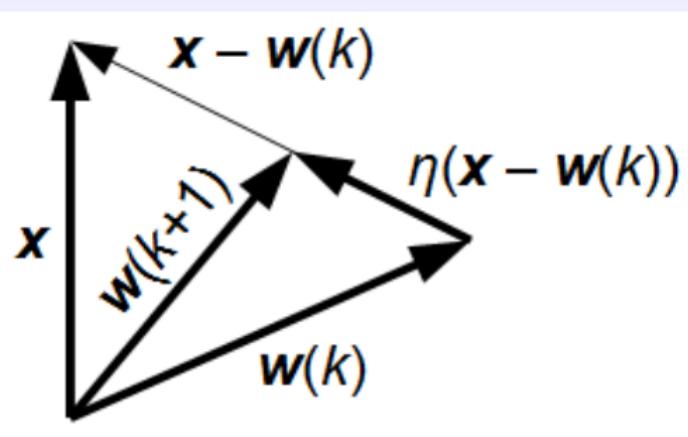


Wagi będziemy modyfikować wg reguły:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$

# Reguła Grossberga

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$



# Reguła Grossberga

Wyjście neuronu to iloczyn skalarny wejścia  $\mathbf{x}$  i wektora wag  $\mathbf{w}$ .

$$y = \mathbf{w}^T \cdot \mathbf{x} = |\mathbf{w}| \cdot |\mathbf{x}| \cdot \cos(\alpha)$$

gdzie:  $\alpha$  to kąt między wektorami  $\mathbf{x}$  i  $\mathbf{w}$ .

Dla znormalizowanych wektorów  $\mathbf{x}$  i  $\mathbf{w}$  wyjście zależy tylko od kąta  $\alpha$ .

# Reguła Grossberga

Wyjście neuronu to iloczyn skalarny wejścia  $\mathbf{x}$  i wektora wag  $\mathbf{w}$ .

$$y = \mathbf{w}^T \cdot \mathbf{x} = |\mathbf{w}| \cdot |\mathbf{x}| \cdot \cos(\alpha)$$

gdzie:  $\alpha$  to kąt między wektorami  $\mathbf{x}$  i  $\mathbf{w}$ .

Dla znormalizowanych wektorów  $\mathbf{x}$  i  $\mathbf{w}$  wyjście zależy tylko od kąta  $\alpha$ .

- Neuron jest najbardziej pobudzony, gdy wektor wejściowy  $\mathbf{x}$  jest podobny do wektora wag  $\mathbf{w}$ .
- W trakcie uczenia wagi upodobniają się do prezentowanej próbki.

# Reguła Grossberga

- Zakładamy, że mamy 1 neuron liniowy.
- Do dyspozycji mamy wiele podobnych próbek uczących  $\mathbf{x}$ .

Wagi będziemy modyfikować wg reguły:

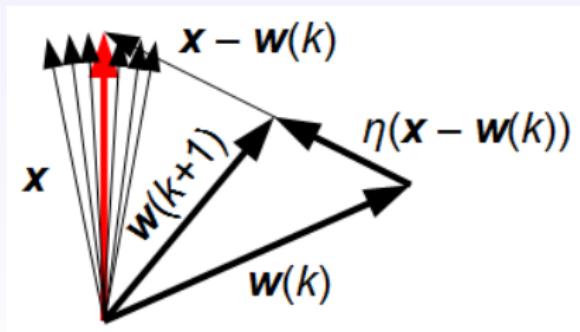
$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$

# Reguła Grossberga

- Zakładamy, że mamy 1 neuron liniowy.
- Do dyspozycji mamy wiele podobnych próbek uczących  $x$ .

Wagi będziemy modyfikować wg reguły:

$$w(k+1) = w(k) + \eta \cdot (x - w(k))$$

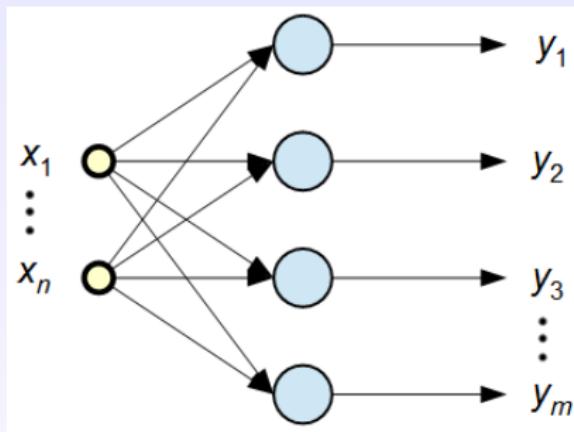


# Reguła Grossberga

- W trakcie uczenia wag upodobniają się do uśrednionej wartości prezentowanych próbek.
- Neuron jest najbardziej pobudzony, gdy wektor wejściowy  $x$  jest podobny do wektora wag  $w$ .

Wytrenowany neuron potrafi określić podobieństwo próbki podanej na wejściu do zapamiętanego wzorca.

# Sieci samoorganizujące



- Sieć zbudowana jest z jednej warstwy neuronów liniowych.
- Wektor wag ma taki sam rozmiar jak wektor wejściowy.
- Każdy neuron reprezentuje jedną klasę (klaster) danych.
- W czasie uczenia neurony konkurują ze sobą o to, który z nich będzie miał modyfikowane wagi.

# Uczenie konkurencyjne – WTA

Wagi najbardziej pobudzonego neuronu będziemy modyfikować wg reguły Grossberga:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k) \cdot (\mathbf{x} - \mathbf{w}(k))$$

W ten sposób każdy neuron uczony będzie tylko tymi próbami, których wartości są podobne do jego wag, czyli należą do klasy którą reprezentuje.

Taką strategię uczenia będziemy nazywali strategią WTA – Winner Takes All.

# Uczenie konkurencyjne – WTA

- W każdym kroku uczenia modyfikacją podlegają wagi tylko jednego neuronu zwycięzcy  $\mathbf{w}^{j^*}$ .
- Neuron zwycięzca jest wyznaczany jako:

$$j^* = \arg \min_{j=1 \dots m} d(\mathbf{x}, \mathbf{w}^j)$$

gdzie:  $d$  – jest miarą odległości pomiędzy wektorami  $\mathbf{x}$  i  $\mathbf{w}^j$ .

# Miary odległości

- iloczyn skalarny:

$$d(\mathbf{x}, \mathbf{w}) = 1 - \mathbf{x} \cdot \mathbf{w} = 1 - \|\mathbf{x}\| \cdot \|\mathbf{w}\| \cdot \cos(\mathbf{x}, \mathbf{w})$$

- Miara euklidesowa:

$$d(\mathbf{x}, \mathbf{w}) = \|\mathbf{x} - \mathbf{w}\| = \sqrt{\sum_{i=1}^n (x_i - w_i)^2}$$

- Miara według normy  $L_1$  (Manhattan):

$$d(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^n |x_i - w_i|$$

- Miara według normy  $L_\infty$ :

$$d(\mathbf{x}, \mathbf{w}) = \max_i |x_i - w_i|$$

# Współczynnik szybkości uczenia $\eta$

Współczynnik szybkości uczenia  $\eta$  powinien się zmniejszać w trakcie uczenia.

Liniowo:

$$\eta(k) = \eta_{min} + (\eta_{max} - \eta_{min}) \frac{k_{max} - k}{k_{max}}$$

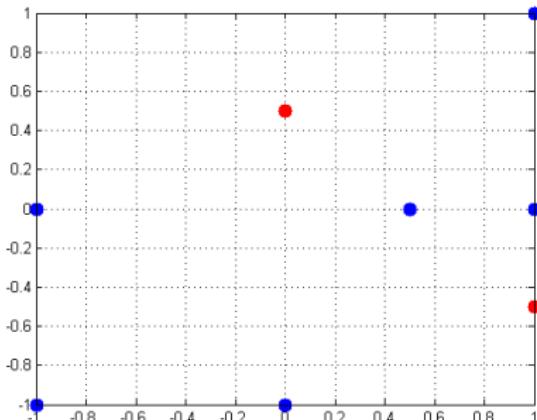
Wykładniczo:

$$\eta(k) = \eta_{max} \left( \frac{\eta_{min}}{\eta_{max}} \right)^{\frac{k}{k_{max}}}$$

# Uczenie konkurencyjne – przykład

Dane uczące:

$x_1$	$x_2$
1	1
3	3
1	2
2.5	2
2	1
3	2

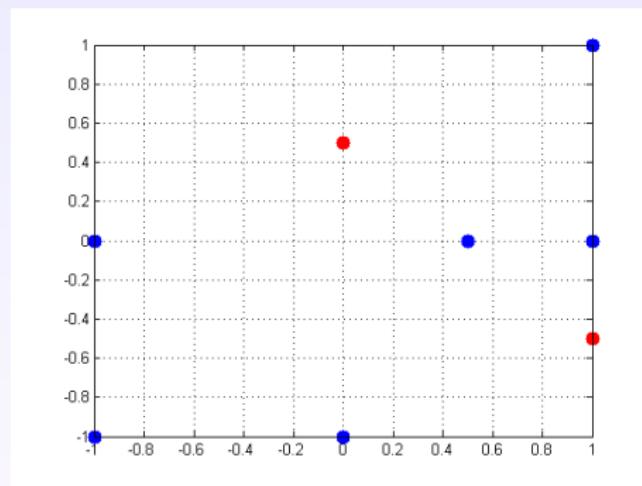


Po normalizacji:

$x_1$	$x_2$
-1	-1
1	1
-1	0
0.5	0
0	-1
1	0

# Uczenie konkurencyjne – przykład

Dane uczące po normalizacji:



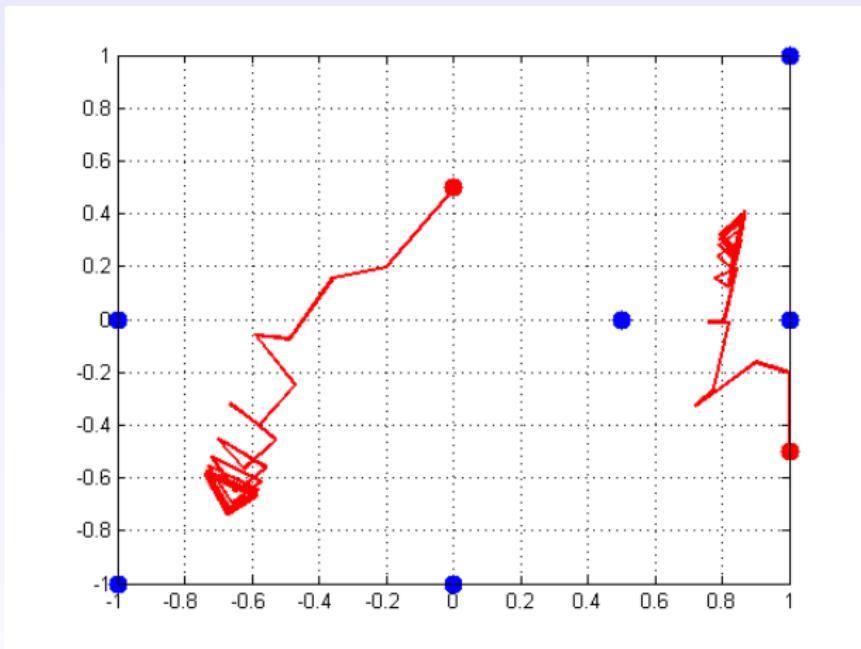
$x_1$	$x_2$
-1	-1
1	1
-1	0
0.5	0
0	-1
1	0

Wagi początkowe:

$$\mathbf{w}^1(0) = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} \quad \mathbf{w}^2(0) = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

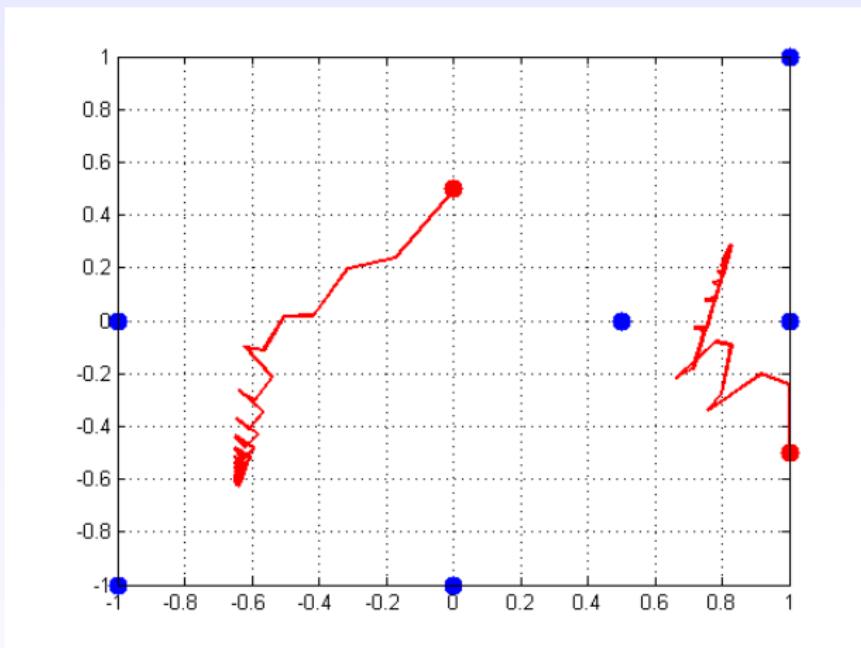
$$\eta = 0.2$$

# Uczenie konkurencyjne – przykład



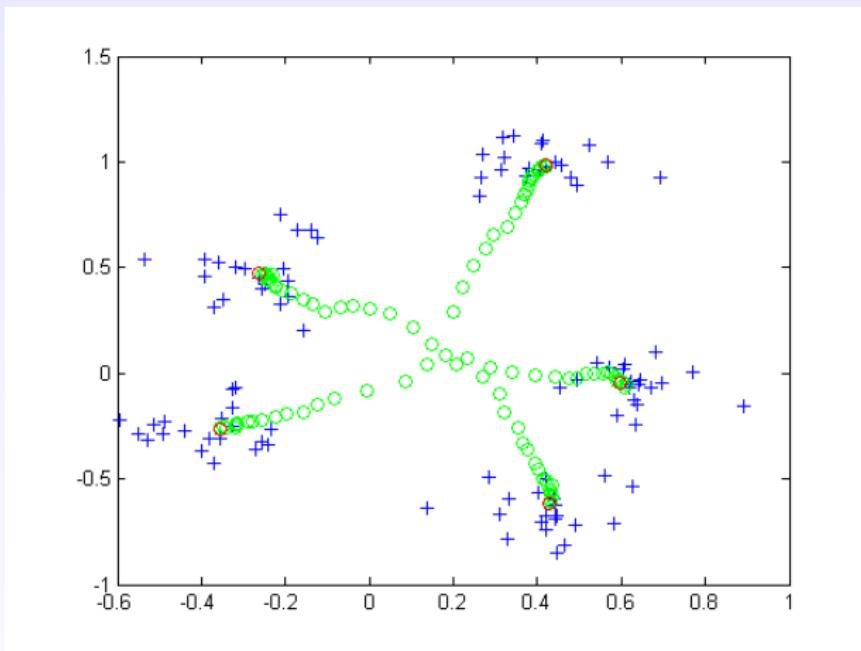
Przebieg uczenia ze stałą wartością  $\eta = 0.2$ .

# Uczenie konkurencyjne – przykład



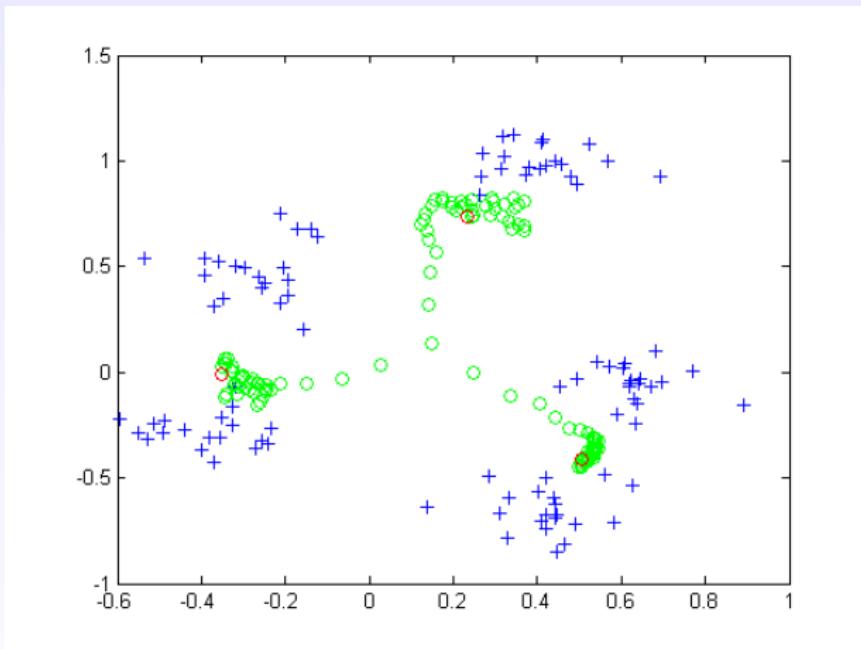
Przebieg uczenia z wartością  $\eta$  zmniejszającą się w trakcie uczenia od 0.2 do 0.

# Uczenie konkurencyjne – przykład



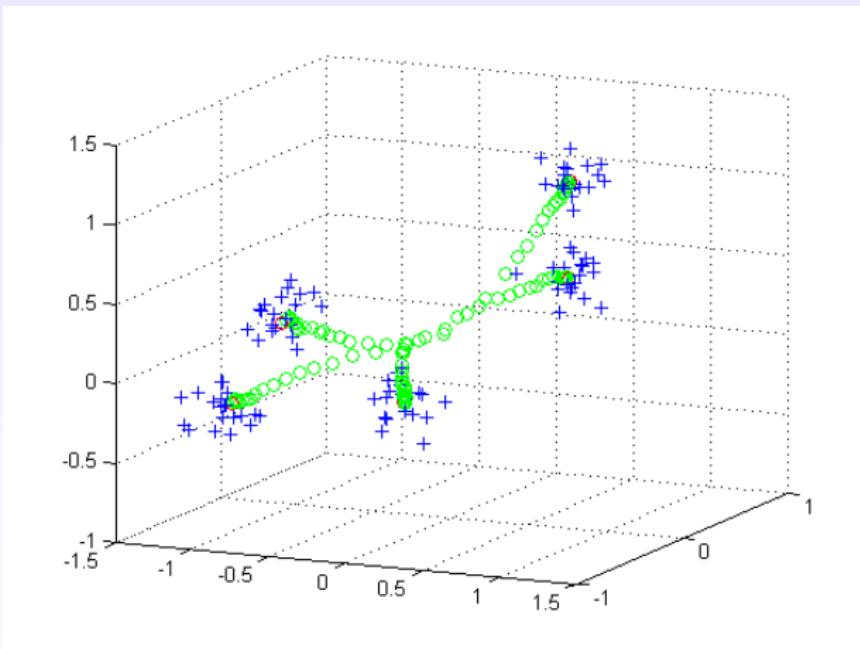
Przebieg uczenia sieci z pięcioma neuronami.

# Uczenie konkurencyjne – przykład



Przebieg uczenia sieci z trzema neuronami.

# Uczenie konkurencyjne – przykład



Przebieg uczenia sieci z pięcioma neuronami.

# Uczenie konkurencyjne – WTM

Za pomocą reguły Kohonena modyfikujemy wagi wszystkich neuronów:

$$\mathbf{w}^j(k+1) = \mathbf{w}^j(k) + \eta(k) \cdot G(\mathbf{w}^j, \mathbf{w}^{j*}) \cdot (\mathbf{x} - \mathbf{w}(k))$$

Taką strategię uczenia będziemy nazywali strategią WTM – Winner Takes Most.

$G(\mathbf{w}^j, \mathbf{w}^{j*})$  – to funkcja sąsiedztwa określająca stopień podobieństwa neuronu nr  $j$  do neuronu zwycięzcy o numerze  $j^*$ .

# Funkcje sąsiedztwa

## WTA

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \begin{cases} 1 & \text{dla } j = j^* \\ 0 & \text{dla pozostałych} \end{cases}$$

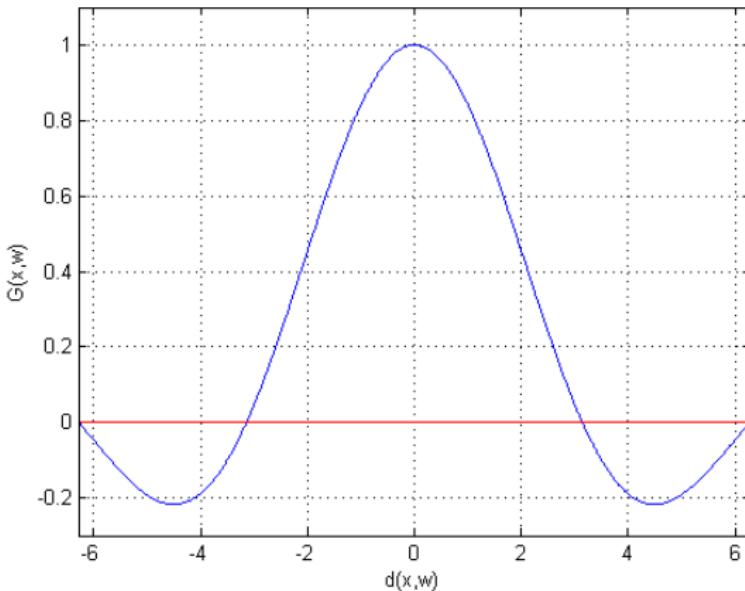
## Sąsiedztwo Gaussowskie

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \exp\left(-\frac{d^2(\mathbf{w}^j, \mathbf{w}^{j^*})}{2\lambda^2}\right)$$

## Funkcja typu 'kapelusz meksykański'

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \begin{cases} 1 & \text{dla } d(\mathbf{w}^j, \mathbf{w}^{j^*}) = 0 \\ \frac{\sin(\lambda \cdot d(\mathbf{w}^j, \mathbf{w}^{j^*}))}{\lambda \cdot d(\mathbf{w}^j, \mathbf{w}^{j^*})} & \text{dla } |d(\mathbf{w}^j, \mathbf{w}^{j^*})| \in (0, 2\pi/\lambda) \\ 0 & \text{dla pozostałych przypadków} \end{cases}$$

# Funkcje sąsiedztwa



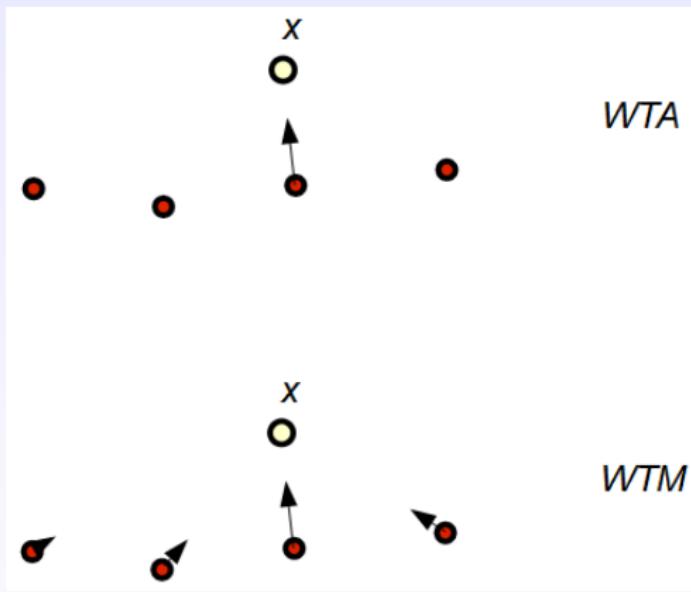
Funkcja typu 'kapelusz meksykański'.

# Funkcje sąsiedztwa

- Szerokość funkcji sąsiedztwa powinna maleć w trakcie uczenia.
- Zmniejszanie wartości  $\lambda$  może się odbywać np. wykładniczo:

$$\lambda(k) = \lambda_{max} \left( \frac{\lambda_{min}}{\lambda_{max}} \right)^{\frac{k}{k_{max}}}$$

# WTA – WTM



Zmiana położenia neuronów w strategii WTA i WTM.

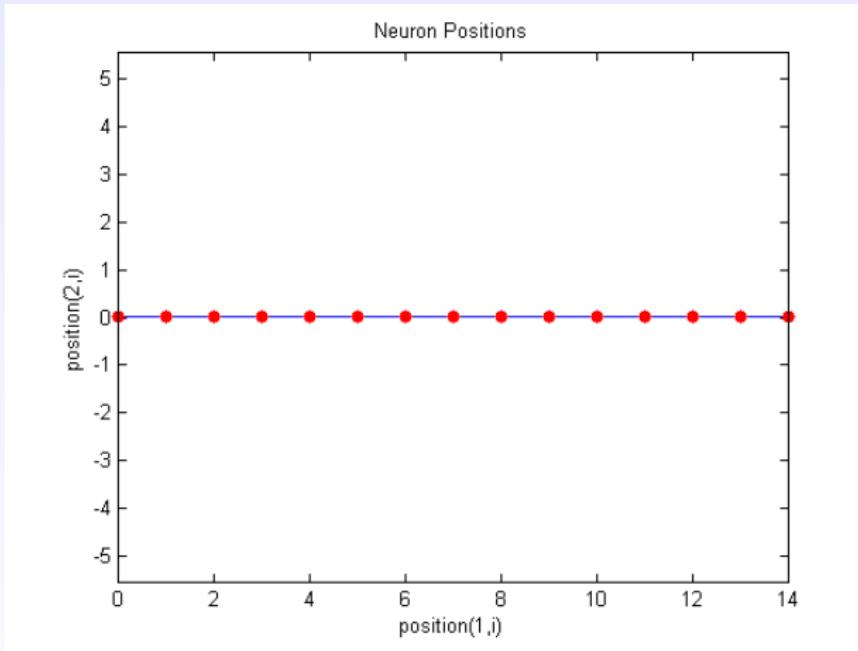
## Topologia

Struktura grafowa definiująca uporządkowanie i sąsiedztwo neuronów.

## Miary podobieństwa

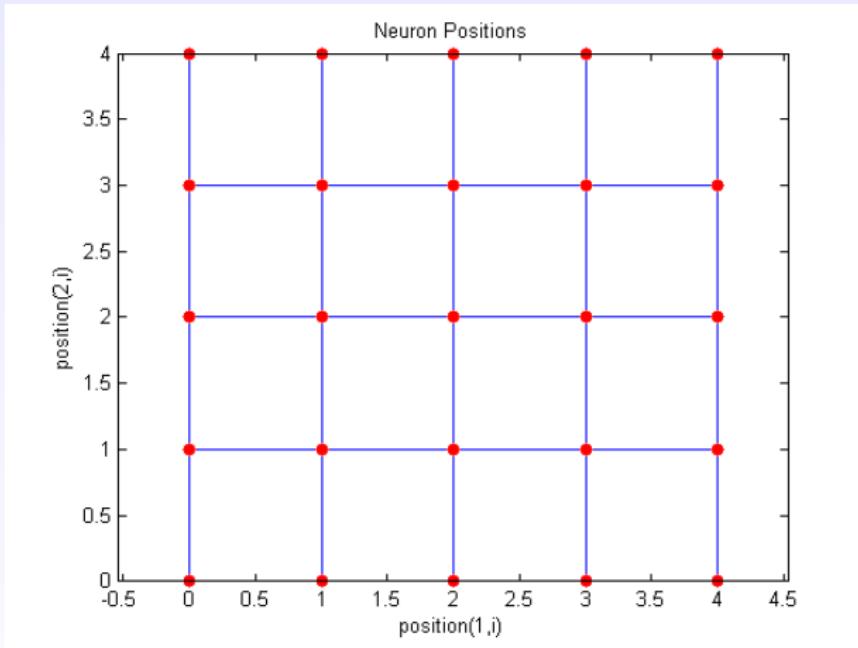
- Metryczne – podobieństwo wektorów określa geometrię przestrzeni wejść.
- Topologiczne – podobieństwo jest zdefiniowane przez narzuczoną na sieć topologię.

# Przykłady topologii



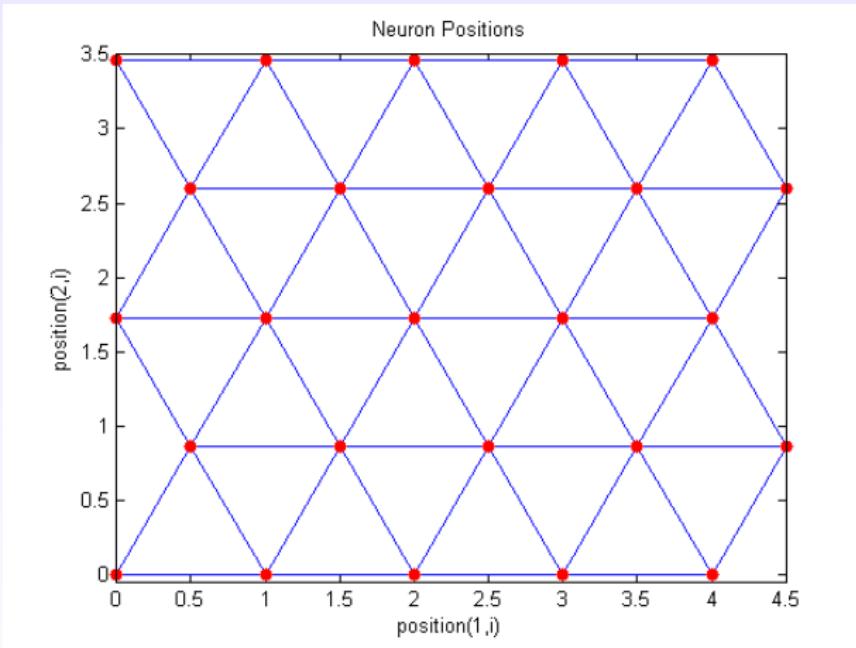
Topologia jednowymiarowa – liniowa.

# Przykłady topologii



Topologia dwuwymiarowa – prostokątna.

# Przykłady topologii



Topologia dwuwymiarowa – heksagonalna.

# Sieci Kohonena

W sieciach z narzuconą topologią, przy określaniu sąsiedztwa bierzemy pod uwagę odległość pomiędzy neuronami w grafie określającym topologię.

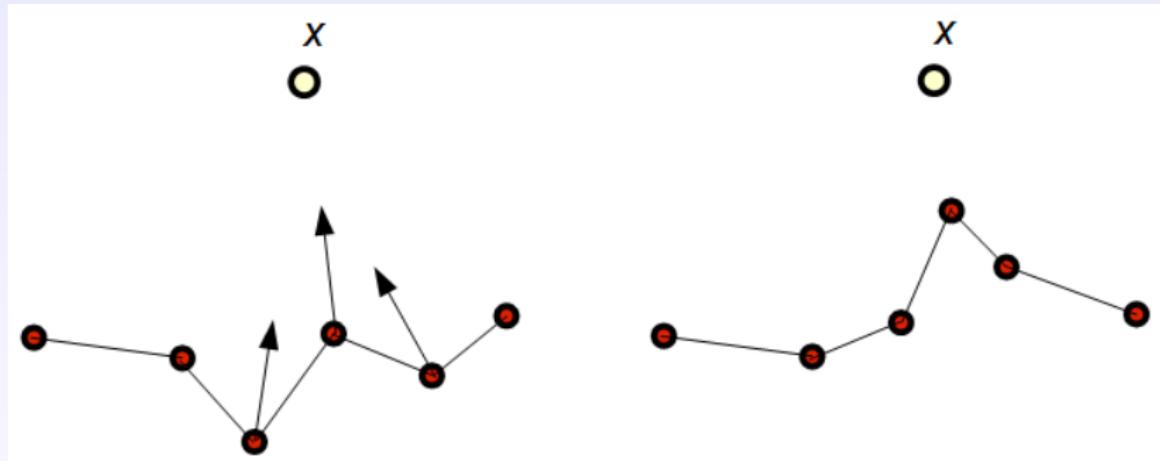
## Sąsiedztwo prostokątne

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \begin{cases} 1 & \text{dla } d(j, j^*) \leq \lambda \\ 0 & \text{dla pozostałych} \end{cases}$$

## Sąsiedztwo Gaussowskie

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \exp\left(-\frac{d^2(j, j^*)}{2\lambda^2}\right)$$

# Sieci Kohonena



Zmiana położenia neuronów przy liniowej topologii i sąsiedztwie prostokątnym o szerokości  $\lambda = 1$ .

# Sieci Kohonena

Sieci z narzuconą topologią tworzą po nauczeniu tzw. mapy cech. Podczas prezentacji próbki na wejściu uaktywnia się tylko 1 wyjście. Idea działania zakłada, aby podobne próbki wejściowe generowały aktywność bliskich (w narzuconym grafie) neuronów. Warstwa wyjściowa jest więc swego rodzaju mapą topograficzną cech danych wejściowych.

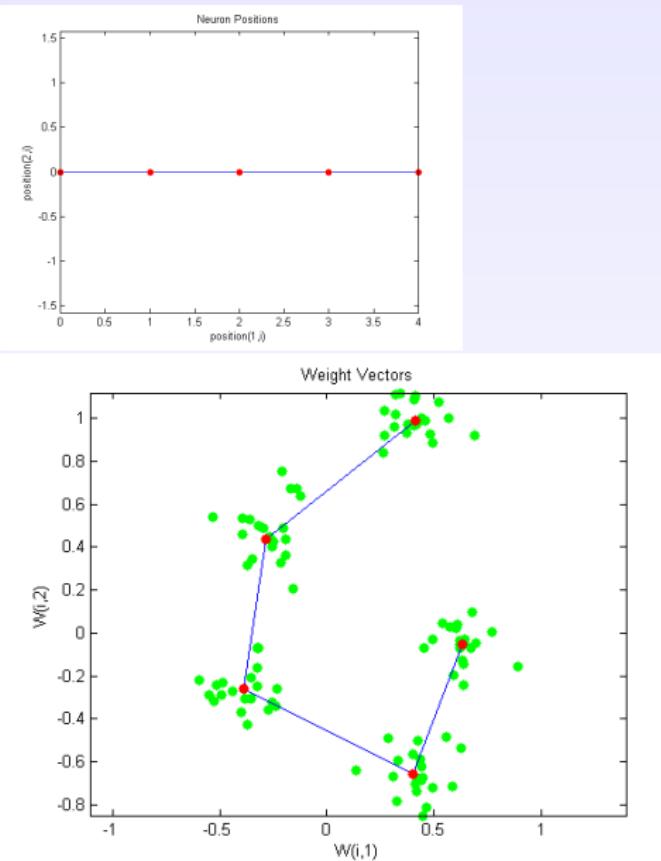
# Jakość sieci samoorganizujących

Jakość sieci samoorganizujących można określić za pomocą zależności:

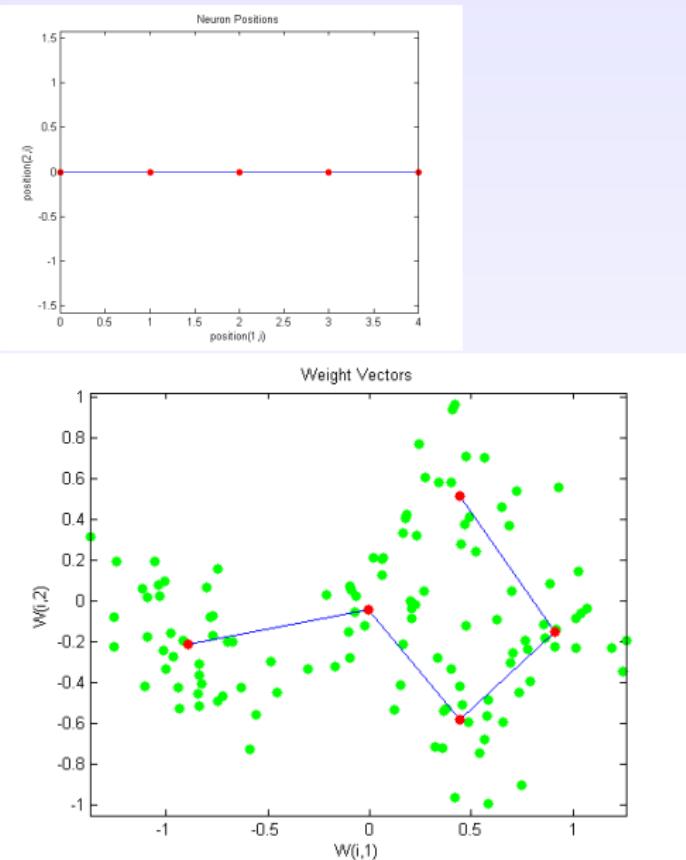
$$q = \sum_{p=1}^L ||\mathbf{x}^p - \mathbf{w}^{j^*}||$$

gdzie:  $\mathbf{w}^{j^*}$  – wagi neuronu zwycięzcy dla próbki  $\mathbf{x}$ .

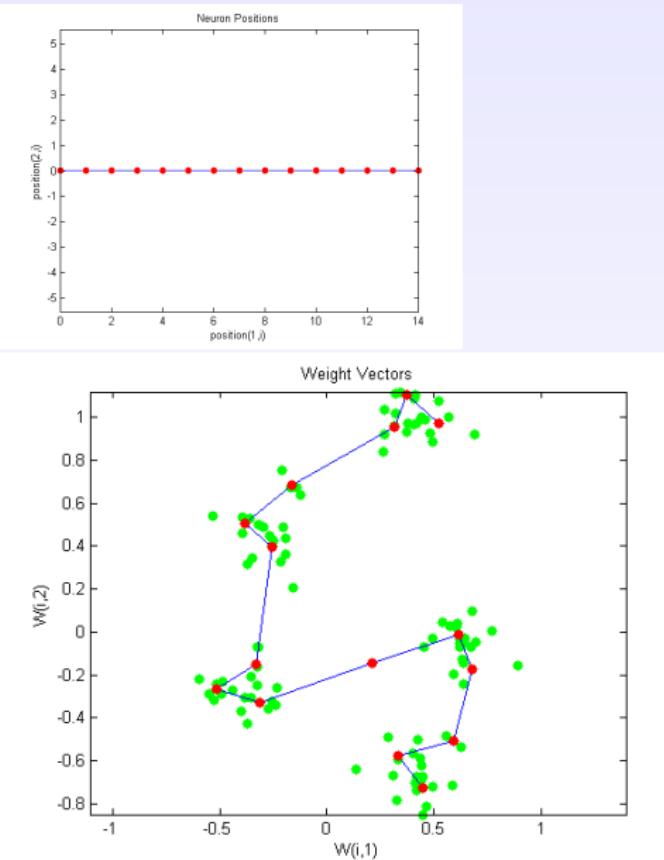
# Sieci Kohonena – przykłady



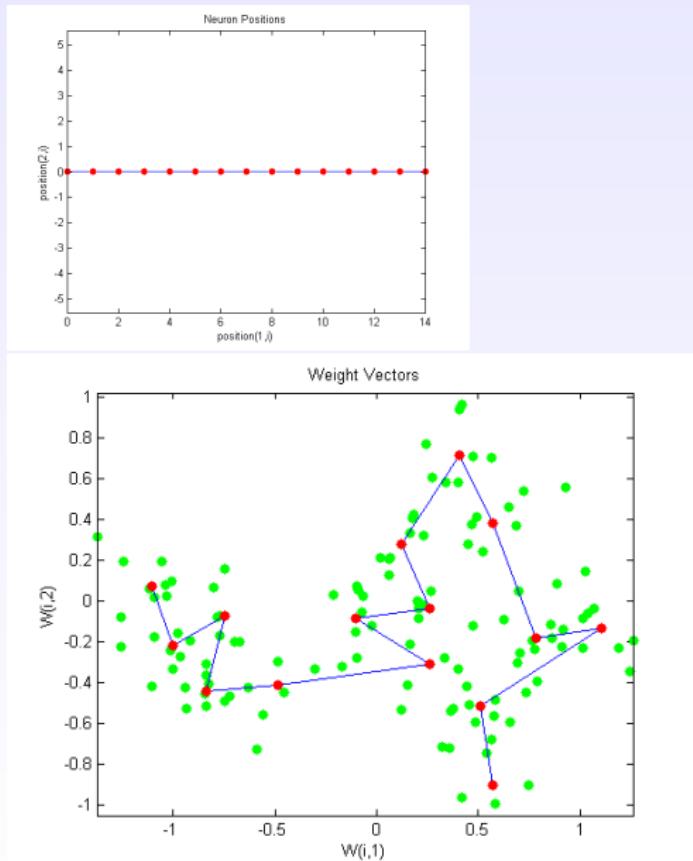
# Sieci Kohonena – przykłady



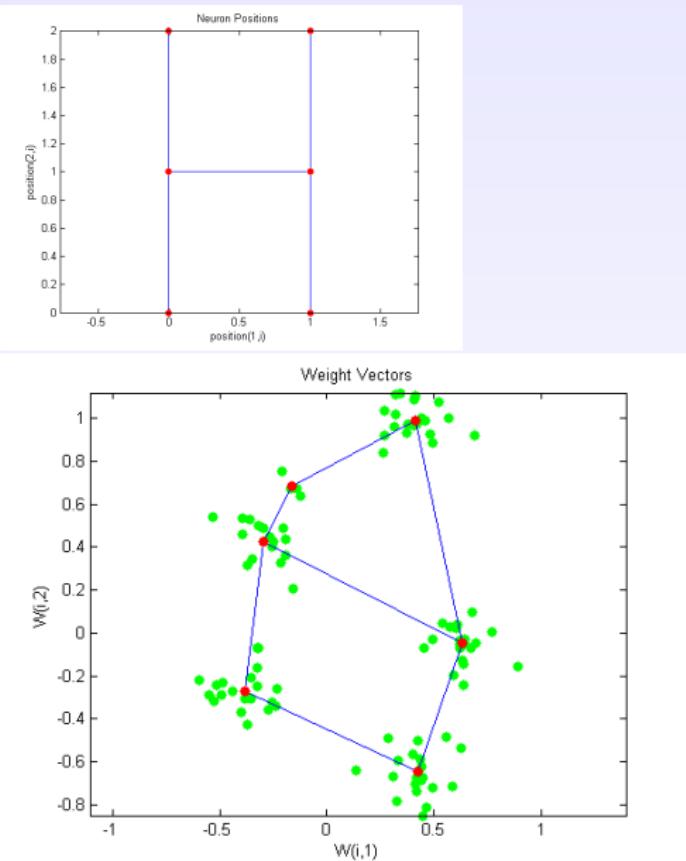
# Sieci Kohonena – przykłady



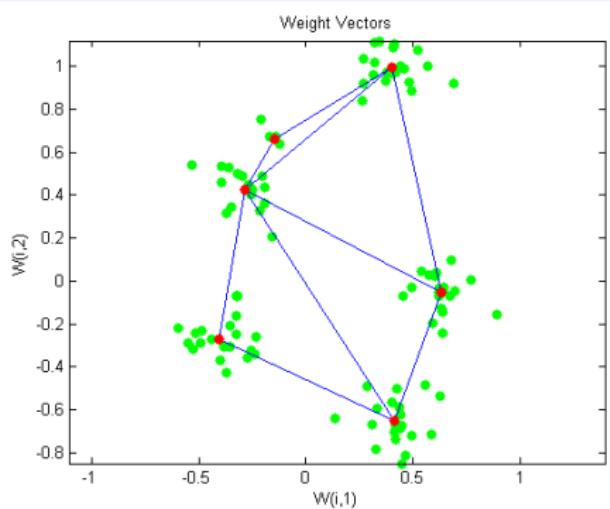
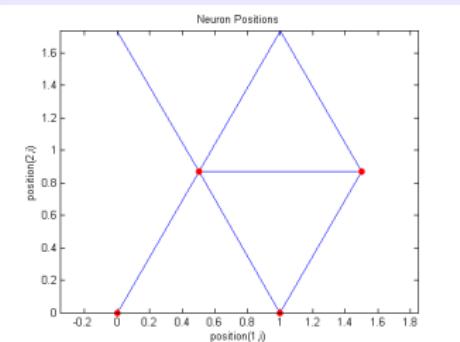
# Sieci Kohonena – przykłady



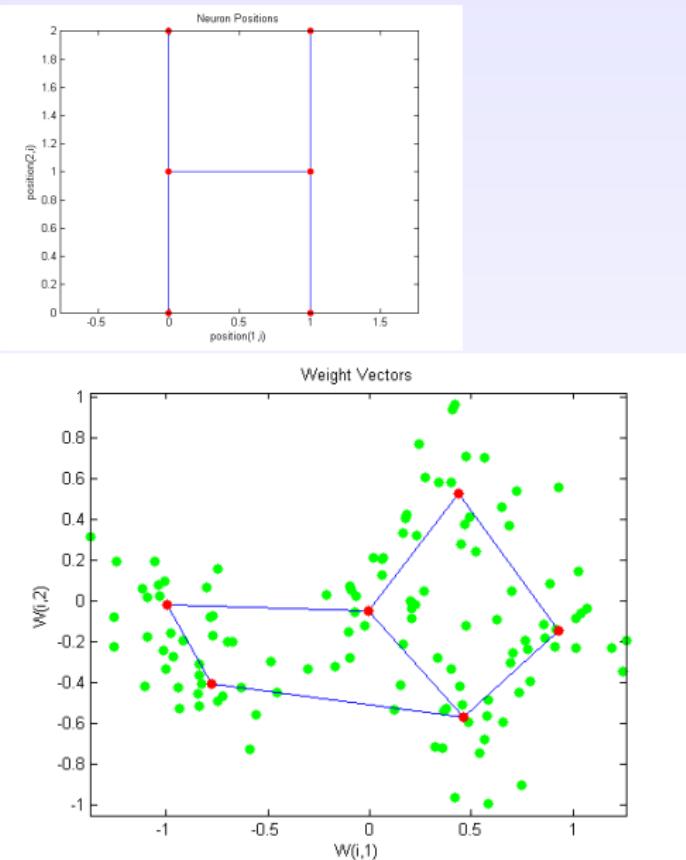
# Sieci Kohonena – przykłady



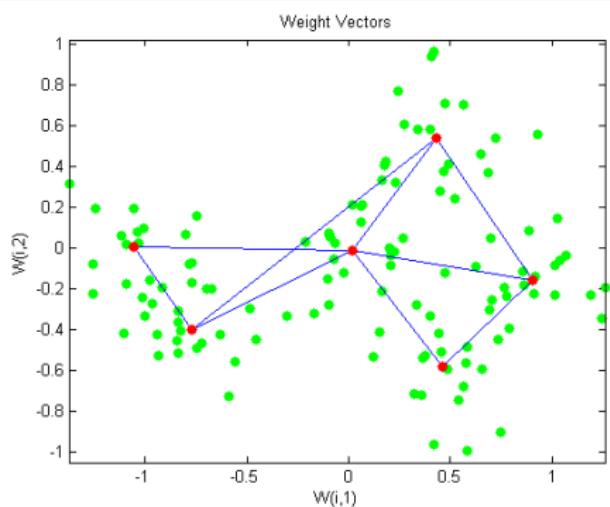
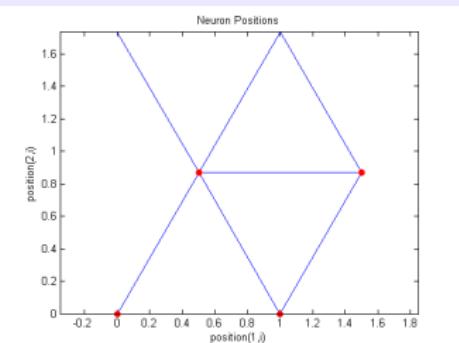
# Sieci Kohonena – przykłady



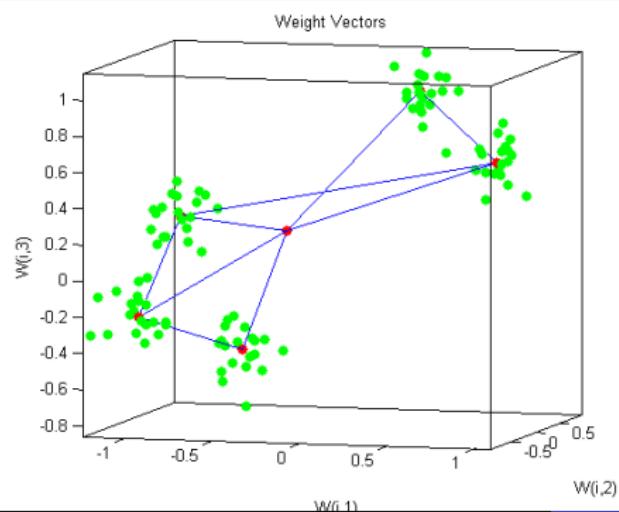
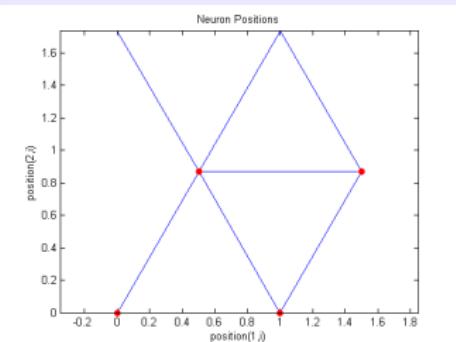
# Sieci Kohonena – przykłady



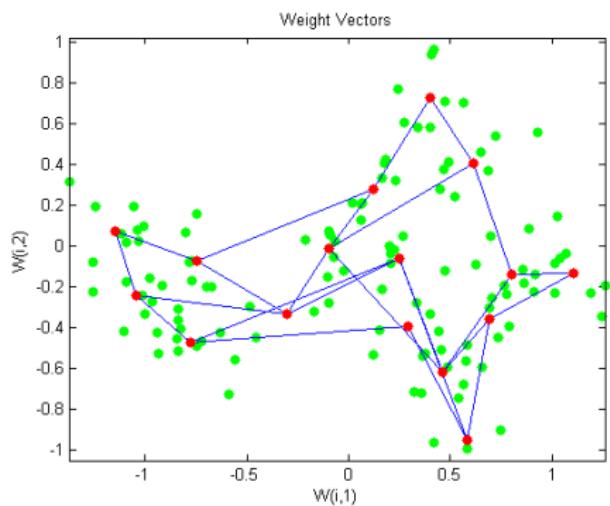
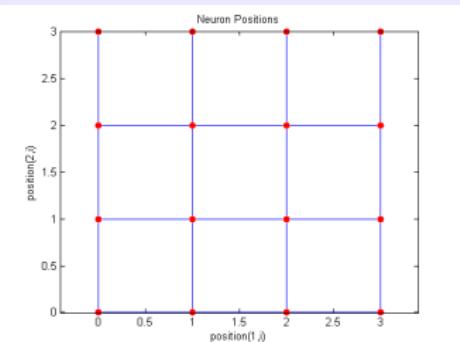
# Sieci Kohonena – przykłady



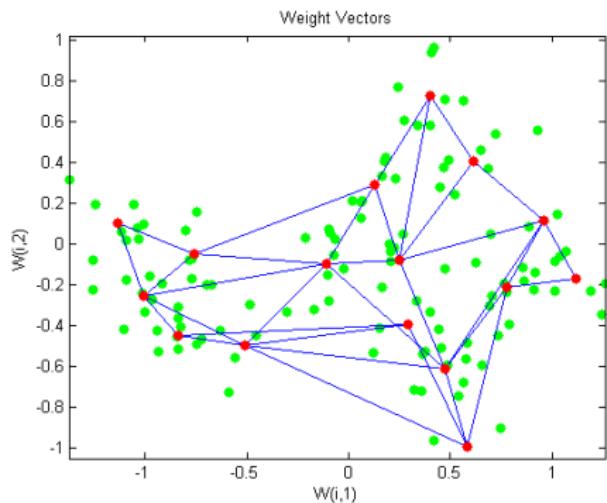
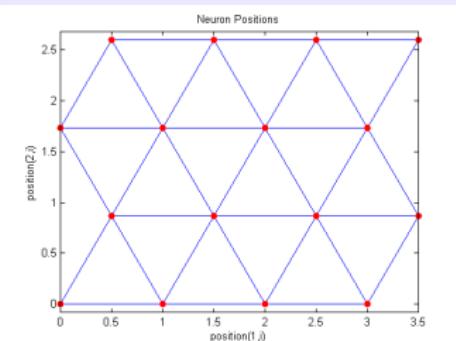
# Sieci Kohonena – przykłady



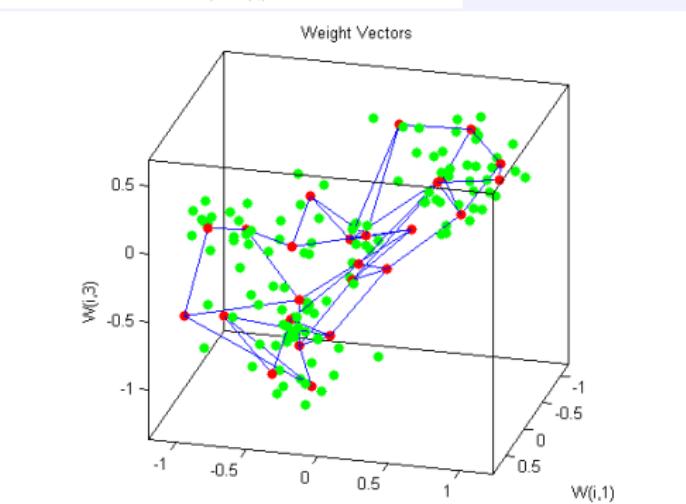
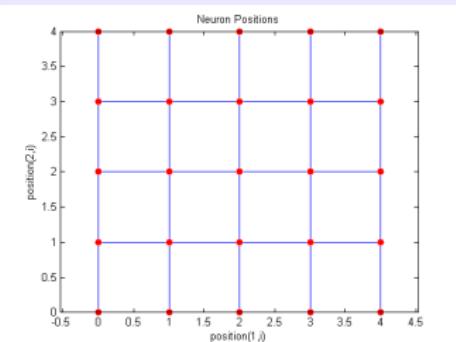
# Sieci Kohonena – przykłady



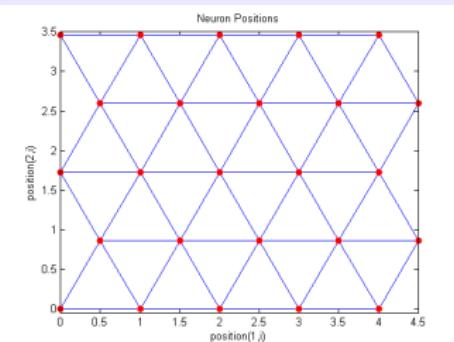
# Sieci Kohonena – przykłady



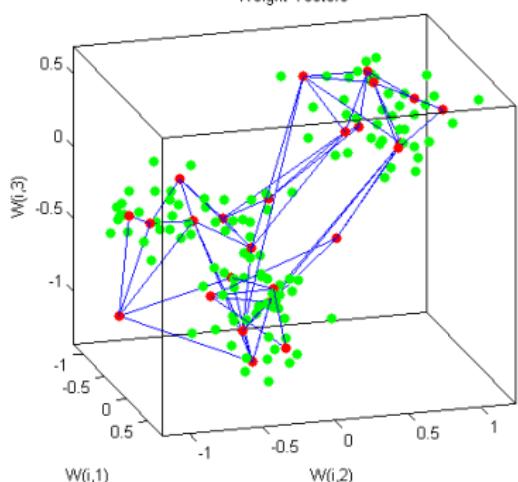
# Sieci Kohonena – przykłady



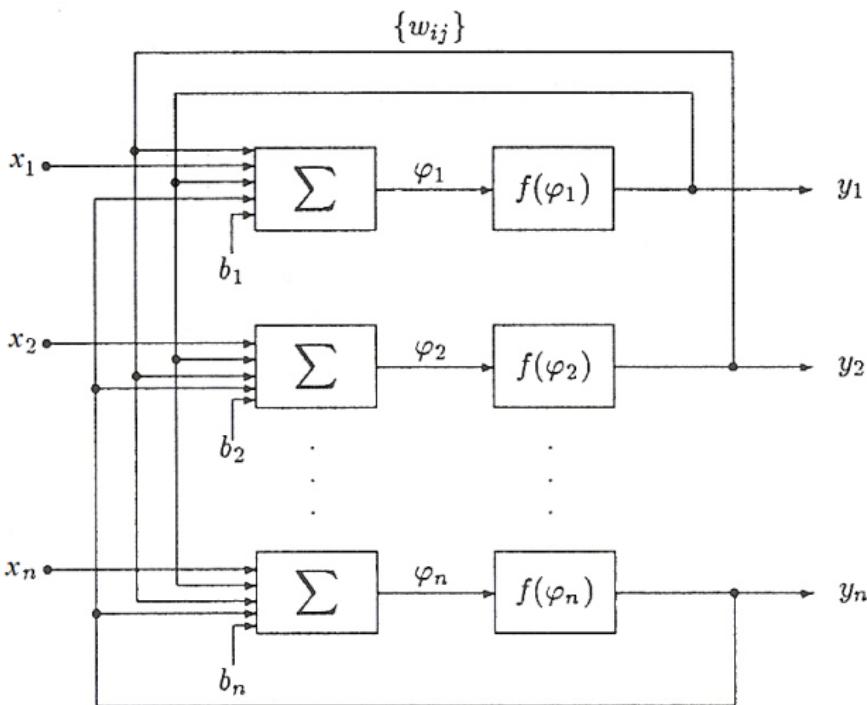
# Sieci Kohonena – przykłady



Weight Vectors



# Sieci rekurencyjne



# Sieci rekurencyjne

## Informacje ogólne

- W sieciach rekurencyjnych występują sprzężenia zwrotne – wyjścia neuronów są połączone z wejściami.
- Sygnał w sieci ‘oscyluje’ pomiędzy wyjściem i wejściem aż do osiągnięcia pewnego kryterium zbieżności – potem zostaje podany na wyjście.
- Z punktu widzenia teorii systemów, sieci rekurencyjne są nieliniowymi układami dynamicznymi.

# Sieci rekurencyjne

## Informacje ogólne

- W sieciach rekurencyjnych występują sprzężenia zwrotne – wyjścia neuronów są połączone z wejściami.
- Sygnał w sieci ‘oscyluje’ pomiędzy wyjściem i wejściem aż do osiągnięcia pewnego kryterium zbieżności – potem zostaje podany na wyjście.
- Z punktu widzenia teorii systemów, sieci rekurencyjne są nieliniowymi układami dynamicznymi.

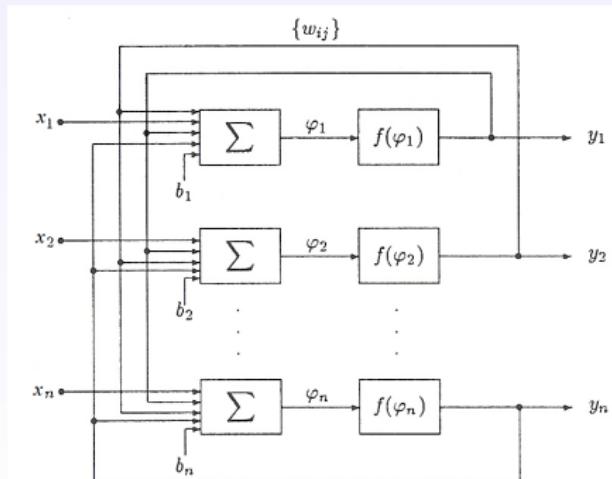
## Zadania

- Pamięć autoasocjacyjna (skojarzeniowa).
- Optymalizacja.

# Dyskretna sieć Hopfielda

## Charakterystyka

- Pojedyncza warstwa (umowna) neuronów objęta sprzężeniem zwrotnym.
- Ilość neuronów = ilości wejść = ilości wyjść =  $n$ .
- Kwadratowa macierz wag o rozmiarze  $n \times n$ .



# Dyskretna sieć Hopfielda

Każdy neuron wyznacza wyjście w oparciu o zależność:

$$\phi_i(k) = \sum_{j=1}^n w_{ij} \cdot y_j + b_i$$

$$y_i(k+1) = \begin{cases} 1 & \text{dla } \phi_i(k) > 0 \\ y_i(k) & \text{dla } \phi_i(k) = 0 \\ 0 & \text{dla } \phi_i(k) < 0 \end{cases}$$

gdzie:

- $y_i(k)$  – wyjście neuronu nr  $i$  w chwili  $k$ ,
- $w_{ij}$  – waga pomiędzy wyjściem neuronu nr  $j$  i wejściem neuronu nr  $i$ ,
- $b_i$  – wejście progowe neuronu nr  $i$  (często nie uwzględniane!).

# Dyskretna sieć Hopfielda

- W sieci występują sprzężenia zwrotne, które polegają na tym, że wyjście każdego elementu jest połączone z wejściami innych elementów. Nie ma połączenia pomiędzy wyjściem i wejściem tego samego neuronu, tzn.:

$$w_{ii} = 0.$$

- Zakłada się, że wagi połączeń są symetryczne, tzn.:

$$w_{ij} = w_{ji}.$$

- Ponieważ każdy neuron jest połączony z pozostałymi, nie ma w niej warstw.

# Dyskretna sieć Hopfielda – działanie

- W chwili początkowej  $k = 0$  doprowadzamy do neuronu sygnały wejściowe  $x_i \in \{0, 1\}$ , które określają stan początkowy sieci.

$$y_i(0) = x_i$$

- W tym momencie wejścia są odłączane, a w sieci rozpoczyna się iteracyjny proces aktualizacji stanu, zgodnie z wzorami:

$$\phi_i(k) = \sum_{j=1}^n w_{ij} \cdot y_j + b_i$$

$$y_i(k+1) = \begin{cases} 1 & \text{dla } \phi_i(k) > 0 \\ y_i(k) & \text{dla } \phi_i(k) = 0 \\ 0 & \text{dla } \phi_i(k) < 0 \end{cases}$$

# Dyskretna sieć Hopfielda – działanie

- Zmiany stanu występują w dyskretnych chwilach czasu.
- Sieć pracuje asynchronicznie – w danej chwili czasu aktualizowane jest tylko 1 wyjście (zwykle wybierane losowo).
- Po **skończonej** liczbie iteracji sieć osiąga stan stabilny:

$$y_i(k+1) = y_i(k), \quad \forall i$$

- W tym momencie kończy się tzw. proces odtwarzania, a stan sieci jest przekazywany na jej wyjście.

# Dyskretna sieć Hopfielda – funkcja energii

- Dla sieci Hopfielda zdefiniowana jest tzw. funkcja energii.
- Każdy stan sieci (zdefiniowany przez wektor wyjścia  $y$ ) określa pewną wartość tej funkcji.
- Funkcja energii jest ograniczona od dołu i nierosnąca w trakcie zmian stanu – oznacza to, że w trakcie procesu odtwarzania wartość funkcji energii maleje lub pozostaje niezmieniona.
- Stan stabilny, osiągany po zakończeniu procesu odtwarzania, odpowiada lokalnemu minimum funkcji energii.

# Dyskretna sieć Hopfielda – funkcja energii

- Dla sieci Hopfielda zdefiniowana jest tzw. funkcja energii.
- Każdy stan sieci (zdefiniowany przez wektor wyjścia  $\mathbf{y}$ ) określa pewną wartość tej funkcji.
- Funkcja energii jest ograniczona od dołu i nierosnąca w trakcie zmian stanu – oznacza to, że w trakcie procesu odtwarzania wartość funkcji energii maleje lub pozostaje niezmieniona.
- Stan stabilny, osiągany po zakończeniu procesu odtwarzania, odpowiada lokalnemu minimum funkcji energii.

Funkcja energii może mieć postać:

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T \mathbf{W}\mathbf{y} + \mathbf{b}^T \mathbf{y} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} y_i y_j + \sum_{i=1}^n b_i y_i$$

# Dyskretna sieć Hopfielda – funkcja energii

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T \mathbf{W}\mathbf{y} + \mathbf{b}^T \mathbf{y} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} y_i y_j + \sum_{i=1}^n b_i y_i$$

Założymy, że stan neuronu  $i$  zmienia się w kroku  $k$  zgodnie z równaniem:

$$y_i(k+1) = y_i(k) + \Delta y_i(k)$$

Ponieważ sieć pracuje asynchronicznie, stan pozostałych neuronów nie zmienia się, tzn.:

$$y_j(k+1) = y_j(k), \quad \text{dla } j \neq i$$

Zmiana energii może być obliczona jako:

$$\Delta E(k) = E(\mathbf{y}(k+1)) - E(\mathbf{y}(k)) = -\Delta y_i(k) \cdot \left( \sum_{j=1}^n w_{ij} y_j + b_i \right) = -\Delta y_i(k) \cdot \phi_i(k)$$

# Dyskretna sieć Hopfielda – funkcja energii

$$\Delta E(k) = E(\mathbf{y}(k+1)) - E(\mathbf{y}(k)) = -\Delta y_i(k) \cdot \left( \sum_{j=1}^n w_{ij} y_j + b_i \right) = -\Delta y_i(k) \cdot \phi_i(k)$$

Jeśli  $\phi_i(k) = 0$  wówczas zmiana energii wynosi 0.

Inne możliwe sytuacje rozpatrzone w tabeli.

$y_i(k+1)$	$y_i(k)$	$\Delta y_i(k)$	$\phi_i(k)$	$\Delta E(k)$
0	0	0	–	0
0	1	-1	–	–
1	0	1	+	–
1	1	0	+	0

Zawsze zatem zachodzi:

$$\Delta E(k) \leq 0$$

$$E(\mathbf{y}(k+1)) \leq E(\mathbf{y}(k))$$

# Dyskretna sieć Hopfielda – funkcja energii

$$E(\mathbf{y}) = -\frac{1}{2} \mathbf{y}^T \mathbf{W} \mathbf{y} + \mathbf{b}^T \mathbf{y} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} y_i y_j + \sum_{i=1}^n b_i y_i$$

Można zauważyć, że funkcja energii jest ograniczona z góry i z dołu bowiem:

$$|E(\mathbf{y})| \leq \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n |w_{ij}| + \sum_{i=1}^n |b_i|$$

# Dyskretna sieć Hopfielda – funkcja energii

$$E(\mathbf{y}) = -\frac{1}{2} \mathbf{y}^T \mathbf{W} \mathbf{y} + \mathbf{b}^T \mathbf{y} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} y_i y_j + \sum_{i=1}^n b_i y_i$$

Można zauważyć, że funkcja energii jest ograniczona z góry i z dołu bowiem:

$$|E(\mathbf{y})| \leq \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n |w_{ij}| + \sum_{i=1}^n |b_i|$$

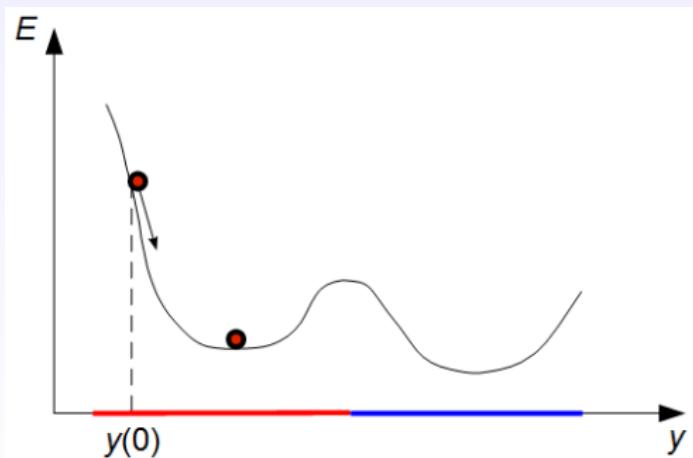
Ponieważ ciąg monotoniczny i ograniczony musi być zbieżny, więc wartość  $E(\mathbf{y})$  będzie dążyła do pewnej skończonej wartości  $E_{min}$ .

# Dyskretna sieć Hopfielda – funkcja energii

- Zbiór wartości funkcji energii jest skończony – wynika to z faktu, że jej dziedzina jest skończona bowiem  $y_i \in \{0, 1\}$ .
- Oznacza to, że zbiór możliwych zmian  $\Delta E$  jest też skończony.
- Możliwe zmiany energii nie mogą być nieskończonymi małe (co mogłoby powodować nieskończonie długie ustalanie się stanu).
- Ostatecznie, energia osiąga stan ustalony  $E_{min}$  w skończonej liczbie  $k_{max}$  kroków.
- Stan  $\mathbf{y}$ , w którym  $E(\mathbf{y}) = E_{min}$  (minimum lokalne) jest stanem stabilnym.

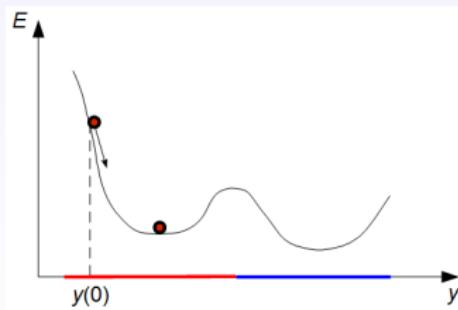
# Dyskretna sieć Hopfielda – funkcja energii

- Początkowy stan sieci jest określony przez wybór  $y(0)$ .
- O ile nie jest to stan stabilny, w trakcie następujących później iteracji (faza odtworzeniowa)  $y$  zmienia się w taki sposób, że wartość funkcji energii maleje, aż do osiągnięcia lokalnego minimum.

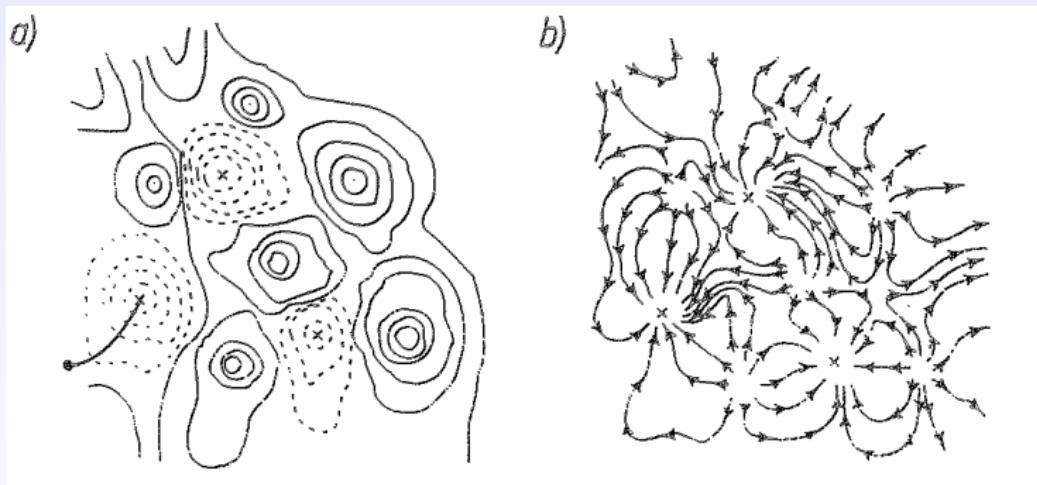


# Dyskretna sieć Hopfielda – funkcja energii

- Stany stabilne nazywamy **atraktorami**.
- Każdemu z nich można przyporządkować zbiór stanów początkowych  $y(0)$ , które inicjują ewolucję stanu sieci kończącą się w tym atraktorze. Zbiór ten nazywamy **niecką przyciągania** danego atraktora.
- Na ilość atraktorów, wzajemne oddalenie oraz odpowiadające im wartości funkcji energii (głębokość niecek) decydujący wpływ ma dobór wag połączeń pomiędzy neuronami.



# Dyskretna sieć Hopfielda – funkcja energii



Ukształtowanie obszarów atrakcji sieci rekurencyjnej: a) wykres poziomnicowy, b) odwzorowanie kierunków zmian wartości funkcji energii w trakcie odtwarzania.

## Pamięć autoasocjacyjna (autoskojarzeniowa)

- Koncepcja takiej pamięci wiąże się z jedną z podstawowych funkcji mózgu. Np. wytężając uwagę potrafimy rozpoznać niewyraźną mowę, przeczytać nieczytelne pismo odręczne, odgadnąć całe hasło w krzyżówce widząc tylko część liter, itp.
- Polega to na odtwarzaniu na zasadzie skojarzeń, całości informacji na podstawie dostępnego jej fragmentu lub informacji właściwej na podstawie informacji zniekształconej.

## Pamięć autoasocjacyjna (autoskójarzeniowa)

Założymy, że mamy  $M$  różnych wzorców:  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M\} \subset R^n$ .  
Pamięcią autoasocjacyjną związaną z tym zbiorem nazwiemy układ realizujący odwzorowanie:

$$F : R^n \rightarrow R^n,$$

takie, że:

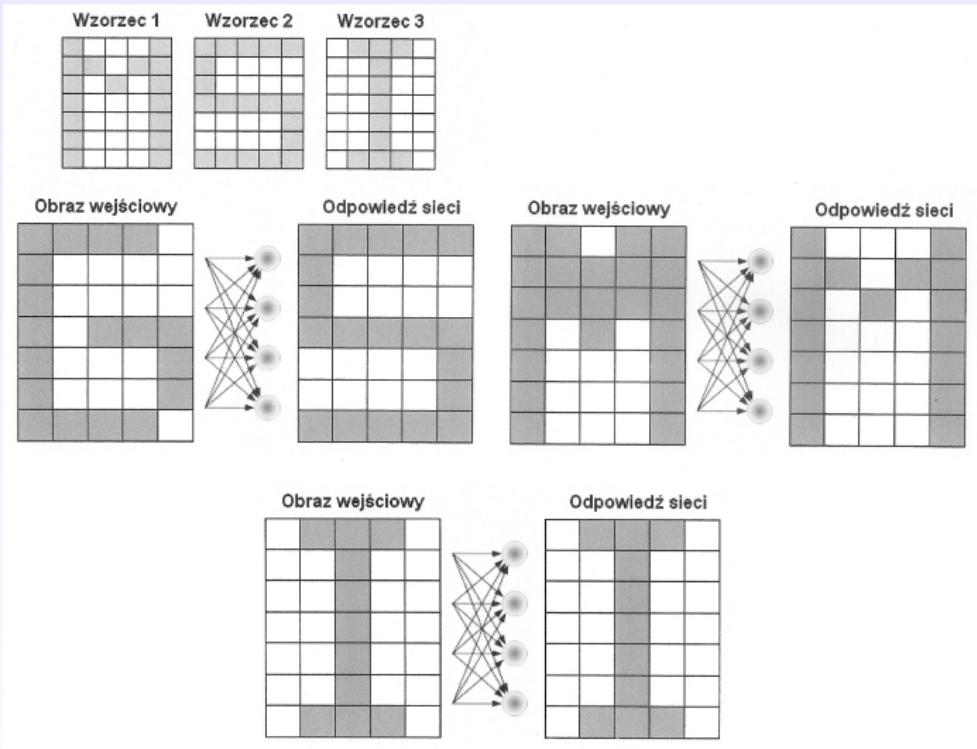
$$F(\mathbf{x}_i) = \mathbf{x}_i, \quad \text{dla } i = 1 \dots M$$

oraz:

$$F(\mathbf{x}) = \mathbf{x}_S,$$

gdzie  $\mathbf{x}_S$  jest najbardziej ‘podobny’ do  $\mathbf{x}$  spośród wszystkich  $M$  wzorców.

# Pamięć autoasocjacyjna (autoskójarzeniowa)



# Stopień podobieństwa

Stopień podobieństwa pomiędzy dwoma wektorami  $\mathbf{x}$  i  $\mathbf{y}$  można określić posługując się np. miarą euklidesową:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad \text{dla } \mathbf{x}, \mathbf{y} \in R^n$$

# Stopień podobieństwa

Stopień podobieństwa pomiędzy dwoma wektorami  $\mathbf{x}$  i  $\mathbf{y}$  można określić posługując się np. miarą euklidesową:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad \text{dla } \mathbf{x}, \mathbf{y} \in R^n$$

Ponieważ analizowane przez nas wektory należą do przestrzeni Hamminga:

$$H^n = \{\mathbf{x} \in R^n : x_i \in \{0, 1\}\},$$

tzn. zbioru wektorów  $n$ -wymiarowych o składowych będących 0 i 1, możemy zastosować tzw. odległość Hamminga  $d_H$ , która dla wektorów  $\mathbf{x}$  i  $\mathbf{y}$  jest równa liczbie różniących się elementów.

# Stopień podobieństwa

Stopień podobieństwa pomiędzy dwoma wektorami  $\mathbf{x}$  i  $\mathbf{y}$  można określić posługując się np. miarą euklidesową:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad \text{dla } \mathbf{x}, \mathbf{y} \in R^n$$

Ponieważ analizowane przez nas wektory należą do przestrzeni Hamminga:

$$H^n = \{\mathbf{x} \in R^n : x_i \in \{0, 1\}\},$$

tzn. zbioru wektorów  $n$ -wymiarowych o składowych będących 0 i 1, możemy zastosować tzw. odległość Hamminga  $d_H$ , która dla wektorów  $\mathbf{x}$  i  $\mathbf{y}$  jest równa liczbie różniących się elementów.

Dla:

$$\mathbf{x} = [1, 0, 0, 1, 0] \quad \text{i} \quad \mathbf{y} = [1, 0, 1, 1, 1]$$

mamy:

$$d_H(\mathbf{x}, \mathbf{y}) = 2$$

- Opisana własność sieci Hopfielda polegająca na istnieniu jednoznacznych atraktorów, w kierunku których ewoluje stan sieci z zadanego stanu początkowego, umożliwia jej wykorzystanie jako pamięci autoasocjacyjnej.
- Kształt funkcji energii (w tym położenie atraktorów) zależy od wag sieci.
- Wystarczy więc tak dobrać wagi, aby każdy wzorzec stał się jednym z atraktorów, a odpowiadająca mu niecka przyciągania była jak najszersza i najgłębsza aby zapewnić poprawność skojarzeń pomiędzy warunkami początkowymi a stanami końcowymi.

# Uczenie sieci Hopfielda

Dla oryginalnie opisanej sieci, Hopfield zaproponował następujący sposób obliczania wag:

$$w_{ij} = \begin{cases} \sum_{m=1}^M (2x_i^{(m)} - 1)(2x_j^{(m)} - 1) & \text{dla } i \neq j \\ 0 & \text{dla } i = j \end{cases}$$

gdzie:

- $M$  – ilość wzorców,
- $x_i^{(m)}$  – wejście nr  $i$  dla wzorca nr  $m$ .

# Uczenie sieci Hopfielda

Dla sieci dyskretnej bipolarnej (stan opisany wartościami  $-1$  i  $1$ ) wagi można obliczyć za pomocą innych zależności.

Jeśli mamy tylko jeden wzorzec:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \{-1, 1\}$$

wagi można obliczyć jako (tzw. reguła Hebb'a):

$$w_{ij} = \begin{cases} \frac{1}{n} \cdot x_i \cdot x_j & \text{dla } i \neq j \\ 0 & \text{dla } i = j \end{cases}$$

lub macierzowo:

$$\mathbf{W} = \frac{1}{n} (\mathbf{x} \cdot \mathbf{x}^T - \mathbf{1})$$

# Uczenie sieci Hopfielda

Jeśli mamy  $M$  wzorców:

$$\mathbf{x} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(M)} \\ \vdots & \dots & & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(M)} \end{bmatrix}, \quad x_i \in \{-1, 1\}$$

wagi można obliczyć jako (tzw. reguła Hebb'a):

$$w_{ij} = \begin{cases} \frac{1}{n} \cdot \sum_{m=1}^M x_i^{(m)} \cdot x_j^{(m)} & \text{dla } i \neq j \\ 0 & \text{dla } i = j \end{cases}$$

lub macierzowo:

$$\mathbf{W} = \frac{1}{n} (\mathbf{X} \cdot \mathbf{X}^T - \mathbf{1}M)$$

# Uczenie sieci Hopfielda

Jeśli mamy  $M$  wzorców:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(M)} \\ \vdots & \dots & & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(M)} \end{bmatrix}, \quad x_i \in \{-1, 1\}$$

wagi można również obliczyć z wykorzystaniem pseudoinwersji:

$$\mathbf{W} = \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$$

# Pojemność sieci Hopfielda

- Pojemność sieci uczonej regułą Hebb'a wynosi  $0.138 \cdot n$  (gdzie  $n$  – liczba neuronów).
- Pojemność sieci uczonej metodą pseudoinwersji wynosi  $n - 1$ .

# Pojemność sieci Hopfielda

## Uwaga!

- Sieć pamiętająca wzorce  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M\}$  pamięta i jest w stanie odtworzyć również ich negacje  $\{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \dots, \bar{\mathbf{x}}_M\}$ .
- Jeśli zbiór wzorców składa się z jednego wzorca, to sieć posiada 2 atraktory związane z wzorcem i jego dopełnieniem.
- Dla większej ilości wzorców atraktorami stają się wzorce, ich dopełnienia oraz koniunkcje wzorców i ich dopełnień ( $\mathbf{x}_1 \text{ AND } \mathbf{x}_2$ ,  $\mathbf{x}_1 \text{ AND } \bar{\mathbf{x}}_2$ , itd.).
- Jeśli wzorców jest dużo może istnieć wiele fałszywych atraktorów, które nie odpowiadają żadnemu wzorcowi. Wadą sieci jest też to, że nie można zagwarantować, że wszystkie wzorce staną się atraktorami.

# Sieć Hopfielda – przykład

Założymy, że mamy tylko 1 wzorzec ( $M = 1$ ):

$$\mathbf{x} = [1 \ 0 \ 1 \ 0]^T$$

Wagi dobieramy za pomocą wzoru:

$$w_{ij} = \begin{cases} \sum_{m=1}^M (2x_i^{(m)} - 1)(2x_j^{(m)-1}) & \text{dla } i \neq j \\ 0 & \text{dla } i = j \end{cases}$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

# Sieć Hopfielda – przykład

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Sprawdzamy zachowanie sieci dla wejścia:

$$\mathbf{x}_t = [1 \ 0 \ 1 \ 0]^T = \mathbf{y}(0)$$

Obliczamy aktywność neuronów:

$$\phi_i(k) = \sum_{j=1}^n w_{ij} \cdot y_j(k)$$

Otrzymujemy w wyniku:

$$\begin{array}{ll} \phi_1(0) = 1 & y_1(1) = 1 \\ \phi_2(0) = -2 & \Rightarrow y_2(1) = 0 \\ \phi_3(0) = 1 & y_3(1) = 1 \\ \phi_4(0) = -2 & y_4(1) = 0 \end{array}$$

# Sieć Hopfielda – przykład

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Sprawdzamy zachowanie sieci dla wejścia:

$$\mathbf{x}_t = [1 \ 0 \ 0 \ 0]^T = \mathbf{y}(0)$$

Obliczamy aktywność neuronów:

$$\phi_i(k) = \sum_{j=1}^n w_{ij} \cdot y_j(k)$$

Otrzymujemy w wyniku:

$$\begin{array}{ll} \phi_1(0) = 0 & y_1(1) = 1 \\ \phi_2(0) = -1 & \Rightarrow y_2(1) = 0 \\ \phi_3(0) = 1 & y_3(1) = 1 \\ \phi_4(0) = -1 & y_4(1) = 0 \end{array}$$

# Sieć Hopfielda – przykład

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Sprawdzamy zachowanie sieci dla wejścia:

$$\mathbf{x}_t = [0 \ 1 \ 0 \ 1]^T = \mathbf{y}(0)$$

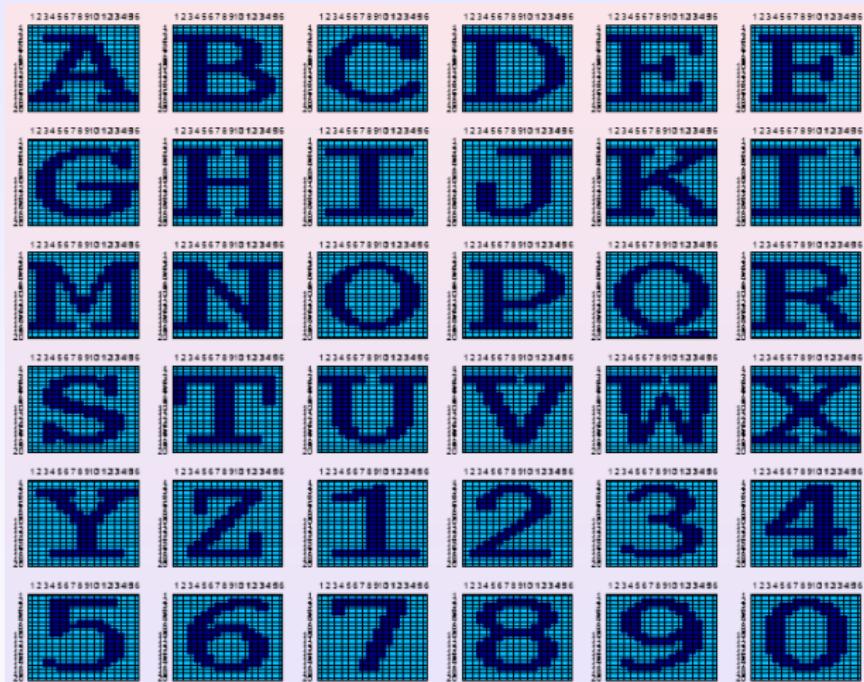
Obliczamy aktywność neuronów:

$$\phi_i(k) = \sum_{j=1}^n w_{ij} \cdot y_j(k)$$

Otrzymujemy w wyniku:

$$\begin{array}{ll} \phi_1(0) = -2 & y_1(1) = 0 \\ \phi_2(0) = 1 & y_2(1) = 1 \\ \phi_3(0) = -2 & \Rightarrow y_3(1) = 0 \\ \phi_4(0) = 1 & y_4(1) = 1 \end{array}$$

# Sieć Hopfielda – rozpoznawanie znaków



36 wzorców uczących – matryca  $20 \times 16$ .

# Sieć Hopfielda – rozpoznawanie znaków

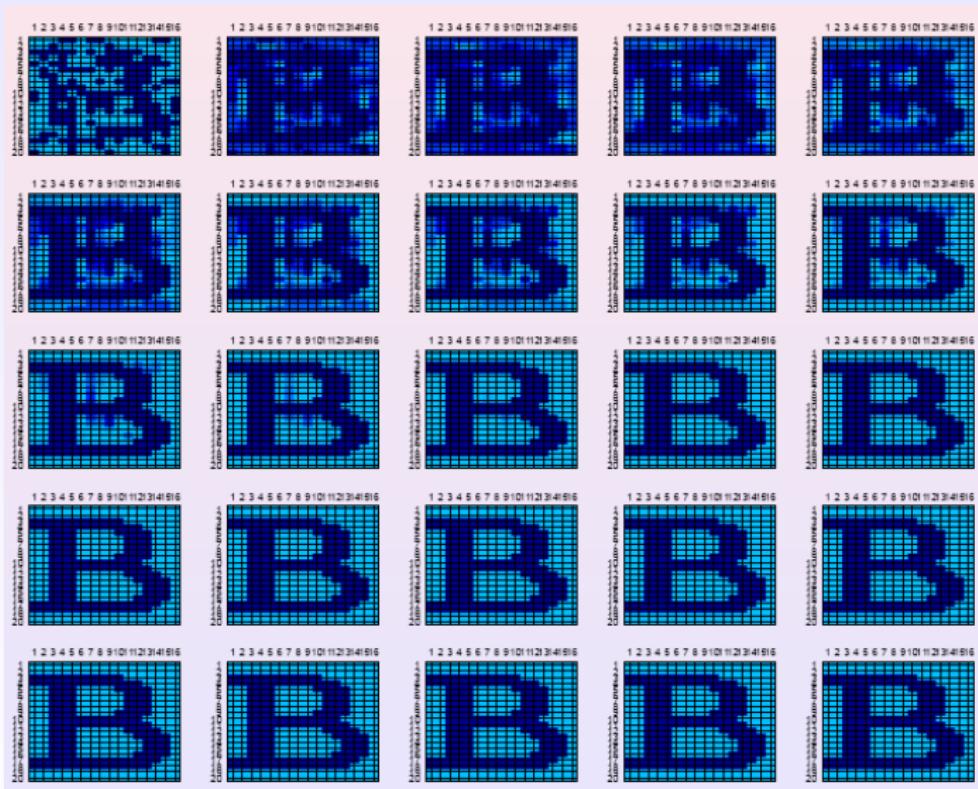
- Każdy znak reprezentuje macierz o rozmiarze  $20 \times 16$  wypełniona liczbami  $-1$  i  $1$ .
- Każdą matrycę ze znakiem zamieniamy na wektor (w MATLAB-ie funkcja `reshape(M,new_row,new_col)`).

$$\begin{bmatrix} \bullet & \dots & \bullet \\ \vdots & & \vdots \\ \bullet & \dots & \bullet \end{bmatrix}_{20 \times 16} \rightarrow \begin{bmatrix} \bullet \\ \bullet \\ \vdots \\ \bullet \end{bmatrix}_{320 \times 1}$$

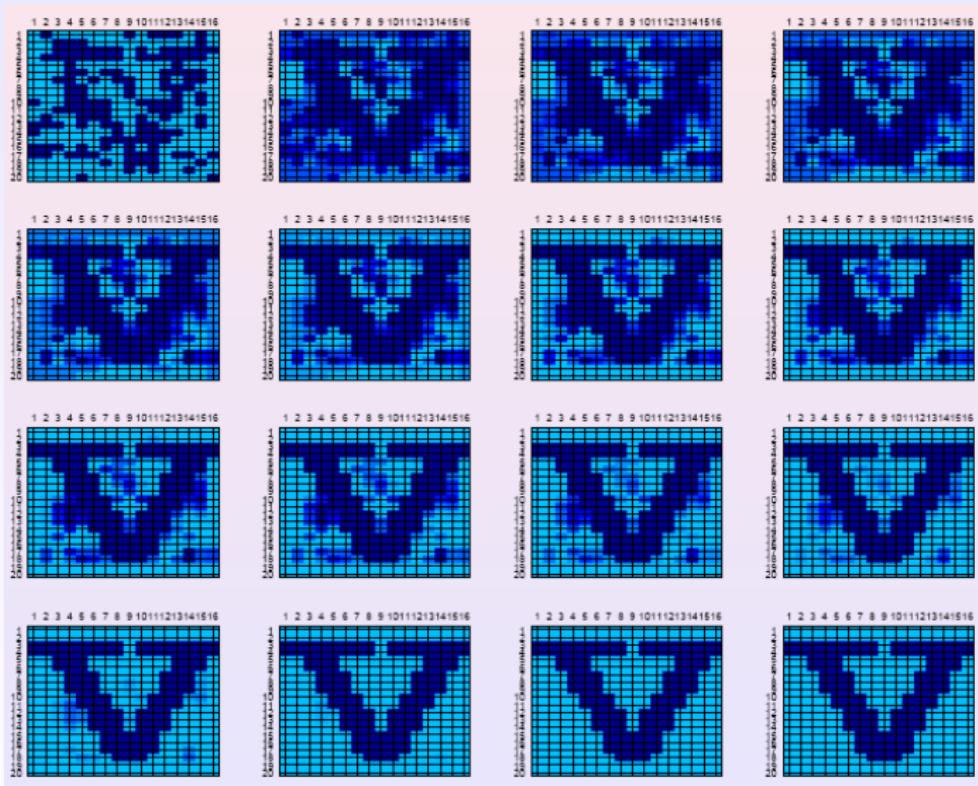
- Dostajemy macierz wzorców:

$$\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(M)}]$$

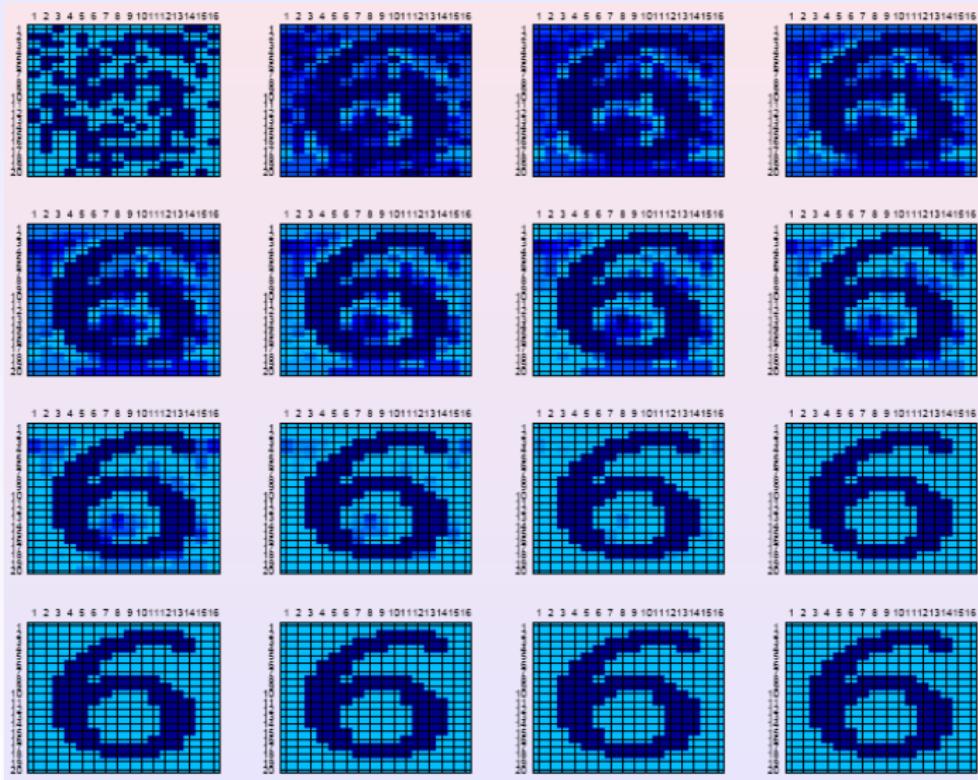
# Sieć Hopfielda – rozpoznawanie znaków



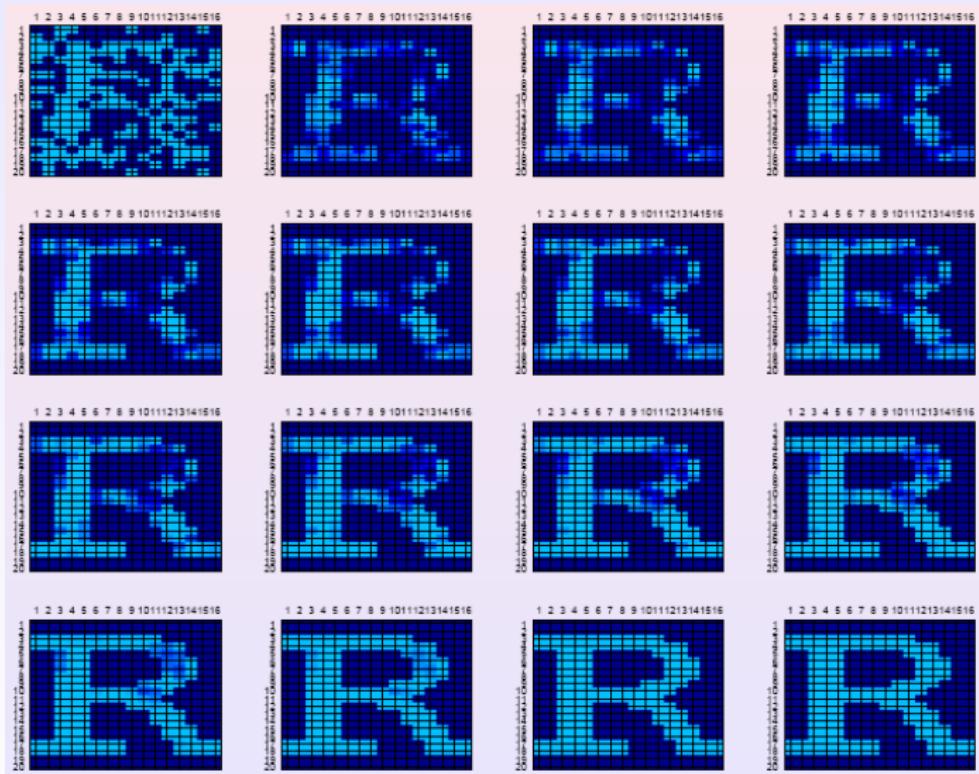
# Sieć Hopfielda – rozpoznawanie znaków



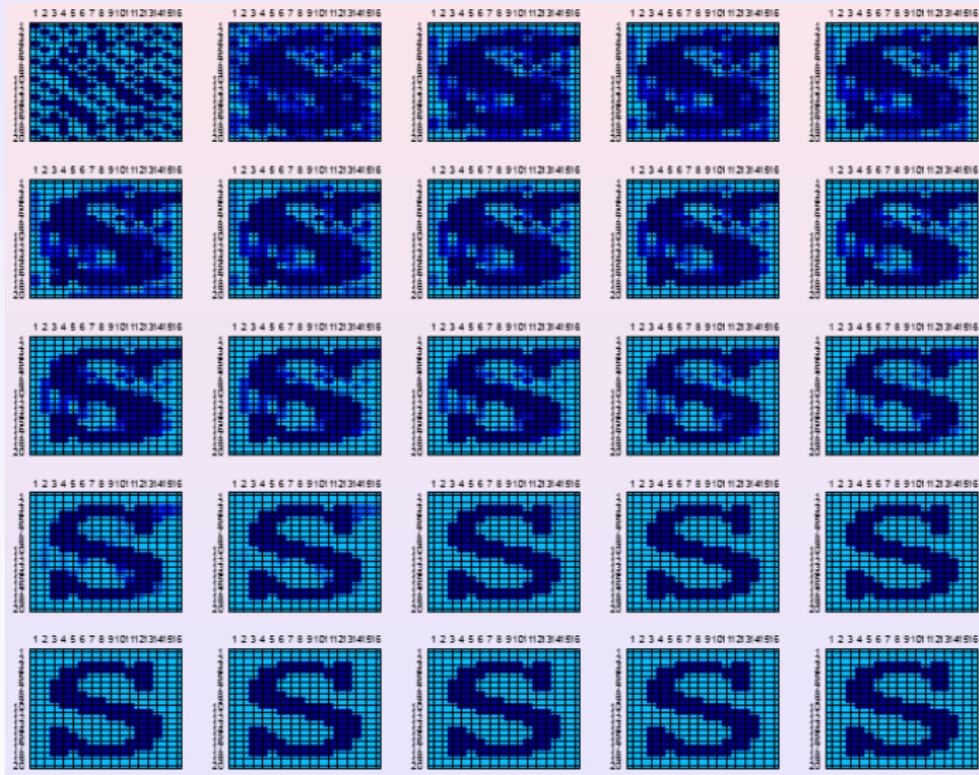
# Sieć Hopfielda – rozpoznawanie znaków



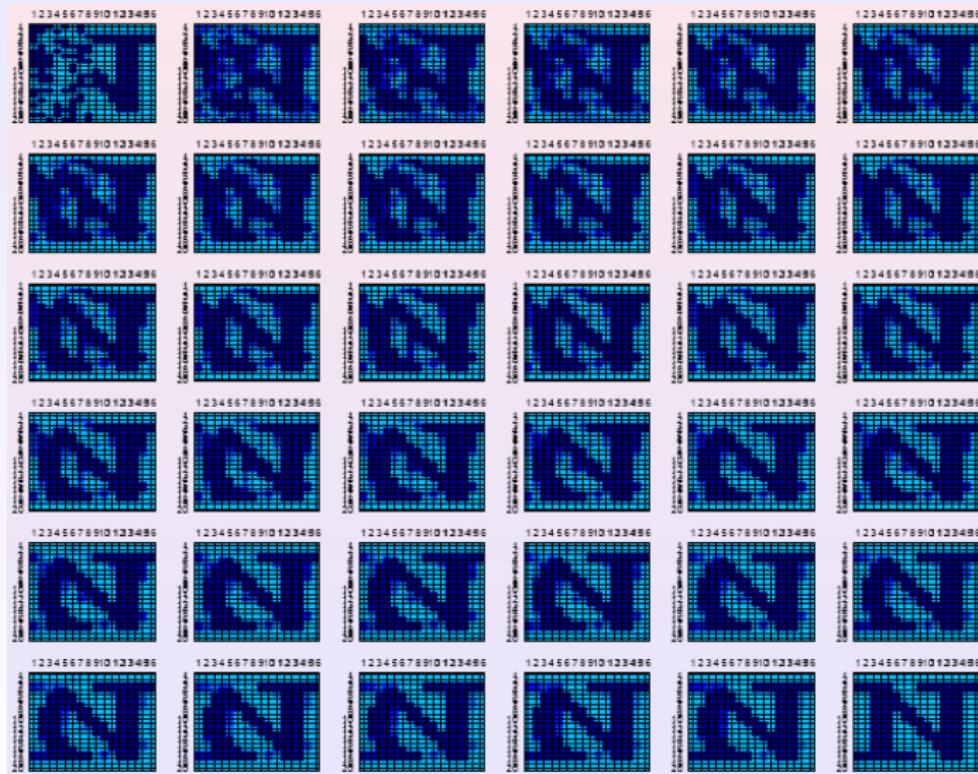
# Sieć Hopfielda – rozpoznawanie znaków



# Sieć Hopfielda – rozpoznawanie znaków



# Sieć Hopfielda – rozpoznawanie znaków



# Interpolacja przebiegów czasowych

# Interpolacja przebiegów czasowych

Zadania:

- ① predykcja,
- ② filtracja sygnałów.

# Interpolacja przebiegów czasowych

- 1 Zakładamy, że sygnał można opisać deterministycznym modelem.
- 2 Przyjmujemy ponadto 'naiwne' założenie, że wartość sygnału w danej, dyskretnej chwili czasu  $k$ , zależy tylko od wartości sygnału w chwilach wcześniejszych:  $k - 1, k - 2$ , itd.
- 3 Podstawą takiego założenia jest przyjęcie modelu sygnału w postaci równania różnicowego.

# Model sygnału – przykład

Założymy, że mamy jednorodne, liniowe równanie różniczkowe w postaci:

$$\ddot{y} + 2\dot{y} + 2y = 0$$

# Model sygnału – przykład

Założymy, że mamy jednorodne, liniowe równanie różniczkowe w postaci:

$$\ddot{y} + 2\dot{y} + 2y = 0$$

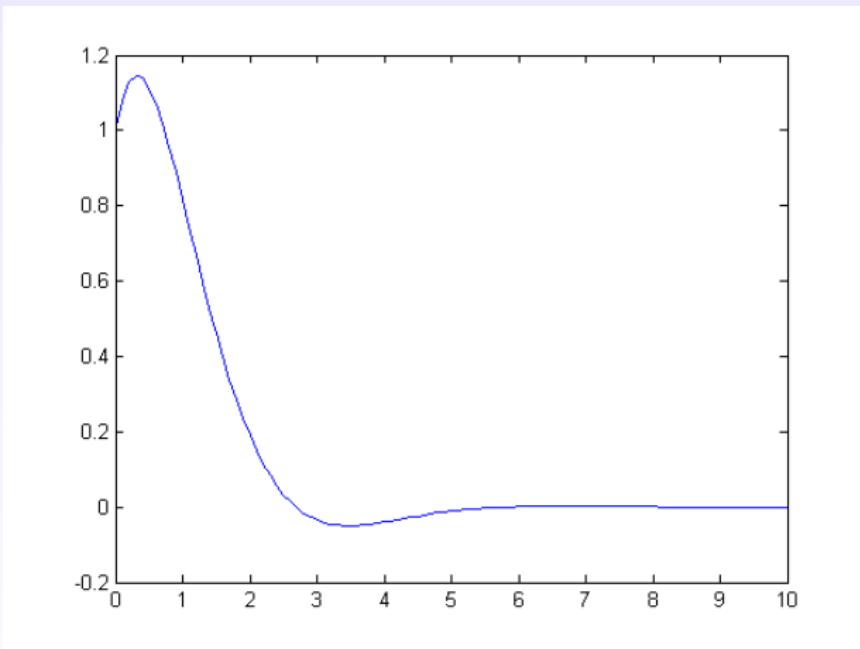
Równanie takie posiada rozwiązanie ogólne:

$$y = e^{-t} \cdot (C_1 \cos(t) + C_2 \sin(t))$$

Przykładowo, dla warunków początkowych:  $y(0) = 1$  i  $\dot{y}(0) = 1$ , można wyznaczyć  $C_1 = 1$  i  $C_2 = 2$ , stąd mamy:

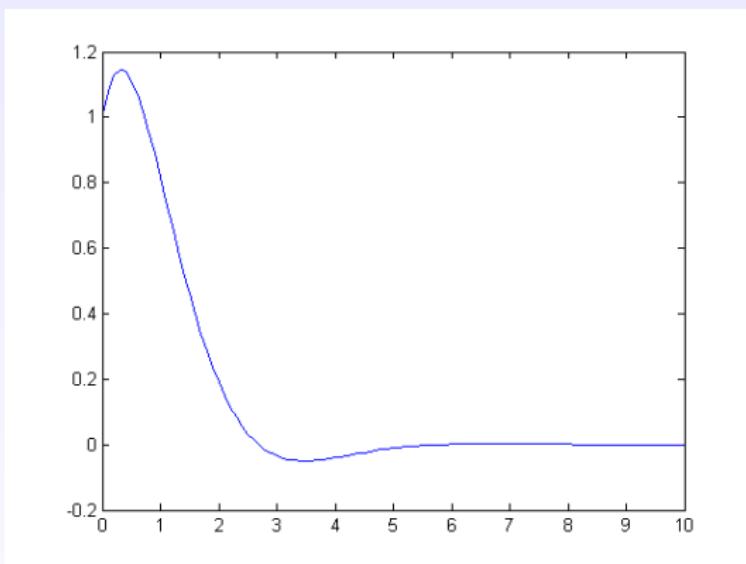
$$y = e^{-t} \cdot (\cos(t) + 2 \sin(t))$$

# Model sygnału – przykład



$$y = e^{-t} \cdot (\cos(t) + 2 \sin(t))$$

# Model sygnału – przykład



Równanie różniczkowe  $\ddot{y} + 2\dot{y} + 2y = 0$  można potraktować zatem jako ciągły model przebiegu, który widzimy na wykresie.

# Model sygnału – przykład

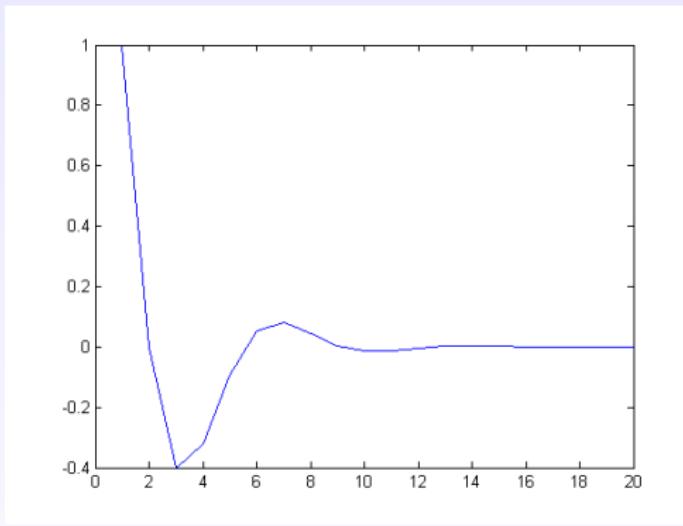
W modelowaniu komputerowym równanie różniczkowe  $\ddot{y} + 2\dot{y} + 2y = 0$  możemy zdyskretyzować.

Otrzymamy wtedy równanie różnicowe:

$$y(k) = y(k-1) \cdot \frac{2+2T}{1+2T+2T^2} - y(k-2) \cdot \frac{1}{1+2T+2T^2}$$

Forma i parametry równania zależy od przyjętej metody dyskretyzacji.  
Parametry zależą także od przyjętego kroku próbkowania  $T$ .

# Model sygnału – przykład

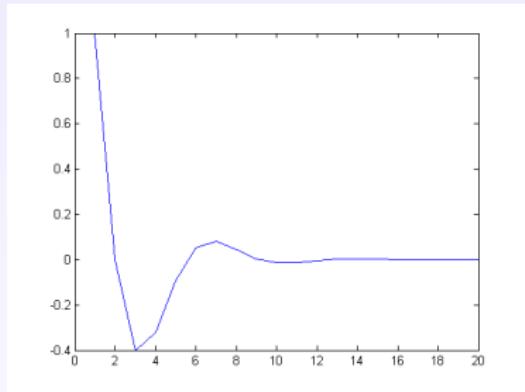
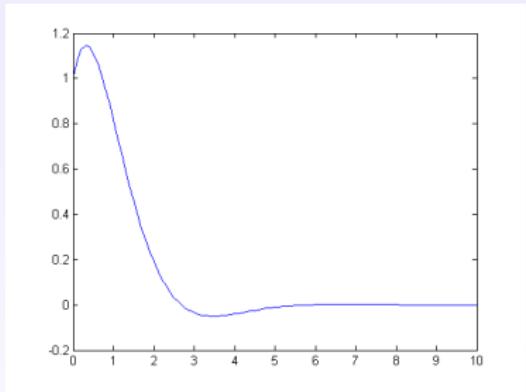


Symulacja modelu opisanego równaniem różnicowym:

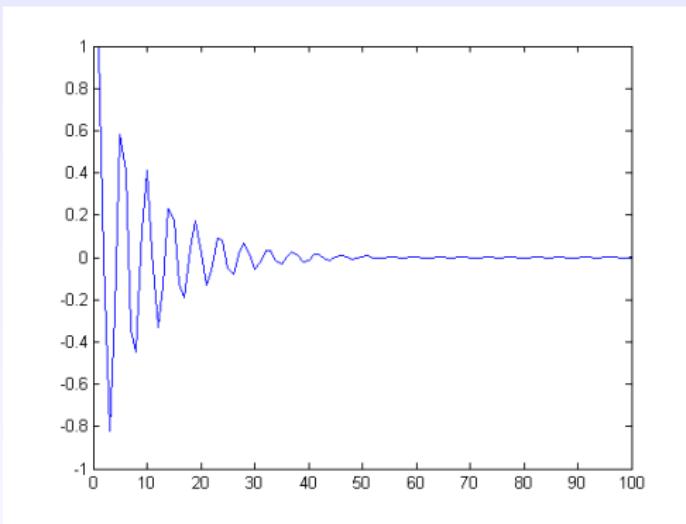
$$y(k) = y(k-1) \cdot \frac{2 + 2T}{1 + 2T + 2T^2} - y(k-2) \cdot \frac{1}{1 + 2T + 2T^2}$$

dla kroku próbkowania  $T = 0.5$  i warunków początkowych  $y(0) = 1, y(1) = 0$ .

# Model sygnału – przykład



# Model sygnału – przykład



Dyskretyzacja może znacząco zmienić zachowanie modelu. Wykres przedstawia symulację modelu opisanego równaniem różnicowym:

$$y(k) = y(k-1) \cdot \frac{2 + 2T}{1 + 2T + 2T^2} - y(k-2) \cdot \frac{1}{1 + 2T + 2T^2}$$

dla kroku próbkowania  $T = 0.1$  i warunków początkowych  $y(0) = 1, y(1) = 0$ .

# Model sygnału – przykład

Równanie różnicowe:

$$y(k) = y(k-1) \cdot \frac{2+2T}{1+2T+2T^2} - y(k-2) \cdot \frac{1}{1+2T+2T^2}$$

można zapisać bardziej ogólnie w postaci:

$$y(k) = w_1 \cdot y(k-1) + w_2 \cdot y(k-2)$$

gdzie:  $w_1 = \frac{2+2T}{1+2T+2T^2}$  i  $w_2 = \frac{1}{1+2T+2T^2}$

Równanie takie może być łatwo zamodelowane przez jeden neuron liniowy!

# Model sygnału

Z pomocą bardziej złożonych sieci, można zamodelować dowolne (liniowe lub nieliniowe) równanie różnicowe o ogólnej postaci:

$$y(k) = F(y(k-1), y(k-2), \dots, y(k-n))$$

$n$  – określa rozmiar okna czasowego, tzn. będziemy korzystać z  $n$  wartości wcześniejszych sygnału, aby przewidzieć wartość aktualną.

Przy modelowaniu za pomocą sieci neuronowej, będziemy mieli  $n$  wejścia (wcześniejsze wartości sygnału) i jedno wyjście (prognozowana wartość aktualna).

# Przygotowanie danych

Załóżmy, że znamy ceny akcji pewnej firmy:

Dzień 1	56
Dzień 2	58
Dzień 3	55
Dzień 4	53
Dzień 5	52
Dzień 6	50

# Przygotowanie danych

Dzień 1	56
Dzień 2	58
Dzień 3	55
Dzień 4	53
Dzień 5	52
Dzień 6	50

Jeśli przyjmiemy okno czasowe o rozmiarze 3, próbki uczące będą miały postać:

Próbka 1:	56 58 55	53
Próbka 2:	58 55 53	52
Próbka 3:	55 53 52	50

# Przygotowanie danych

Dzień 1	56
Dzień 2	58
Dzień 3	55
Dzień 4	53
Dzień 5	52
Dzień 6	50

Jeśli przyjmiemy okno czasowe o rozmiarze 3, próbki uczące będą miały postać:

Próbka 1:	56	58	55	53
Próbka 2:	58	55	53	52
Próbka 3:	55	53	52	50

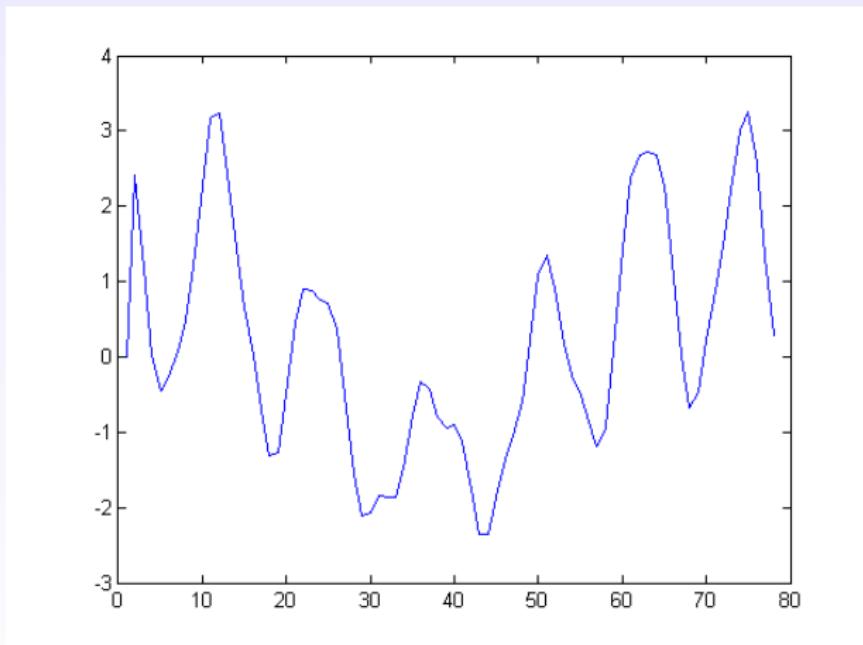
Sieć nauczona z ich pomocą, będzie modelować równanie różnicowe:

$$y(k) = F(y(k-1), y(k-2), y(k-3))$$

# Przykłady

Dany jest sygnał:

$$y = \sin(t) + \cos(t) + \sin(3t) \cdot \cos(3t) + \sin(5t) + \cos(5t) + 0.3 \sin(50t)$$



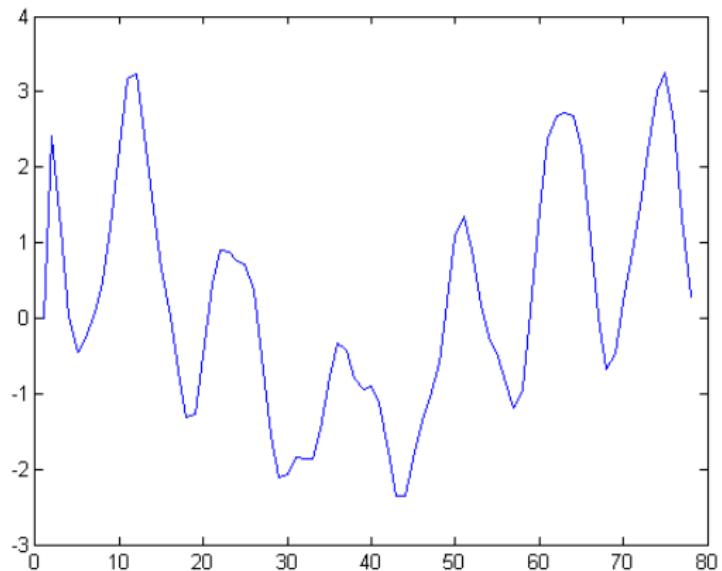
# Przykłady

- Przyjmujemy rozmiar okna czasowego  $n = 4$ .
- Sieć będzie modelować równanie różnicowe:

$$y(k) = F(y(k - 1), y(k - 2), y(k - 3), y(k - 4))$$

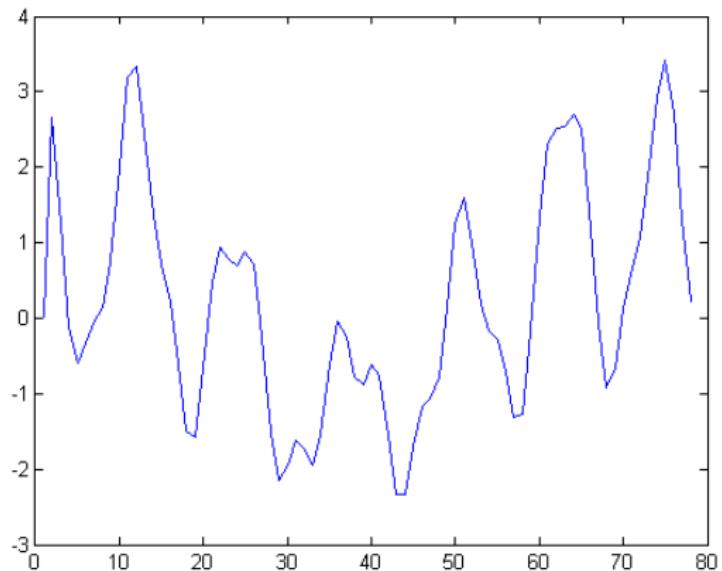
- Przyjmujemy 6 neuronów na warstwie ukrytej.
- Warstwa ukryta ma funkcję aktywacji typu tangens hiperboliczny, warstwa wyjściowa jest liniowa.
- Dane uczące przygotowujemy w sposób pokazany wcześniej.

# Przykłady



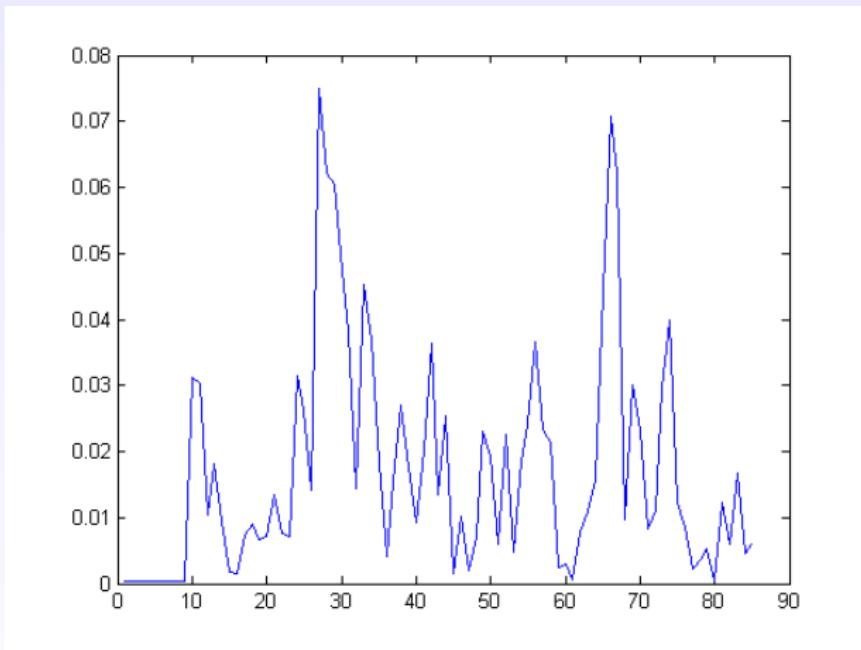
Sygnał uczący.

# Przykłady



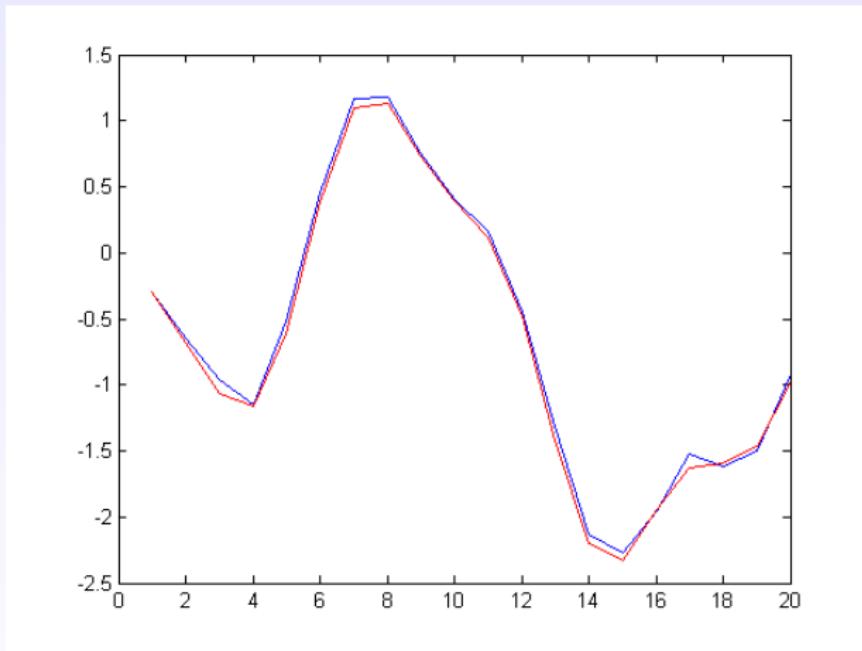
Sygnał zamodelowany przez sieć neuronową.

# Przykłady



Błąd sieci.

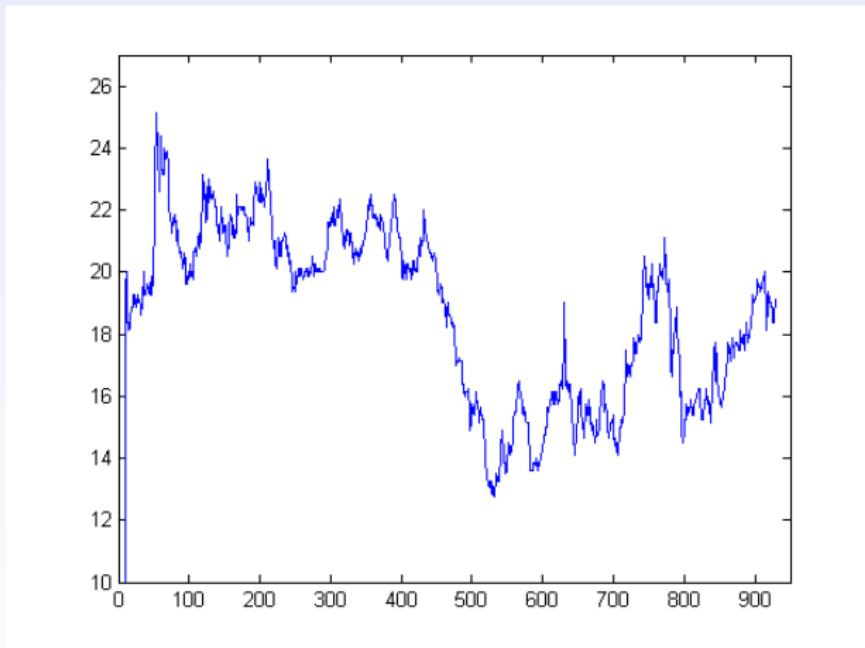
# Przykłady



Prognozowanie danych nie wykorzystanych w uczeniu. Linia niebieska – prognoza, linia czerwona – wartość faktyczna.

# Przykłady

Dane są wartości akcji pewnej firmy:



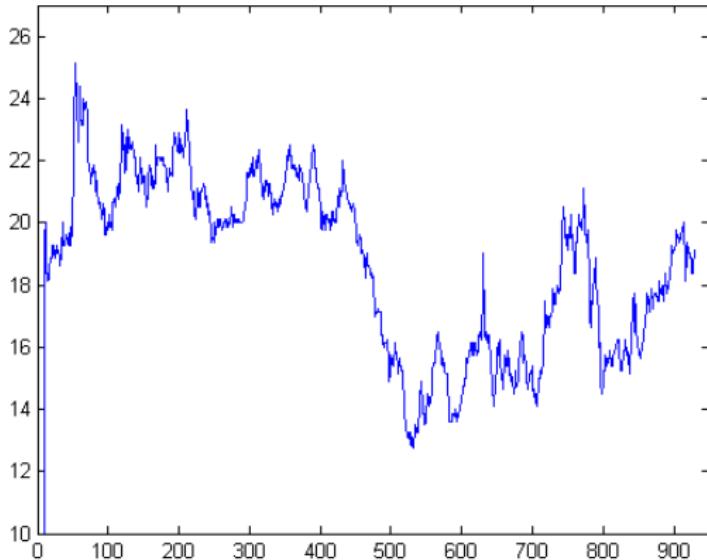
# Przykłady

- Przyjmujemy rozmiar okna czasowego  $n = 5$ .
- Sieć będzie modelować równanie różnicowe:

$$y(k) = F(y(k-1), y(k-2), y(k-3), y(k-4), y(k-5))$$

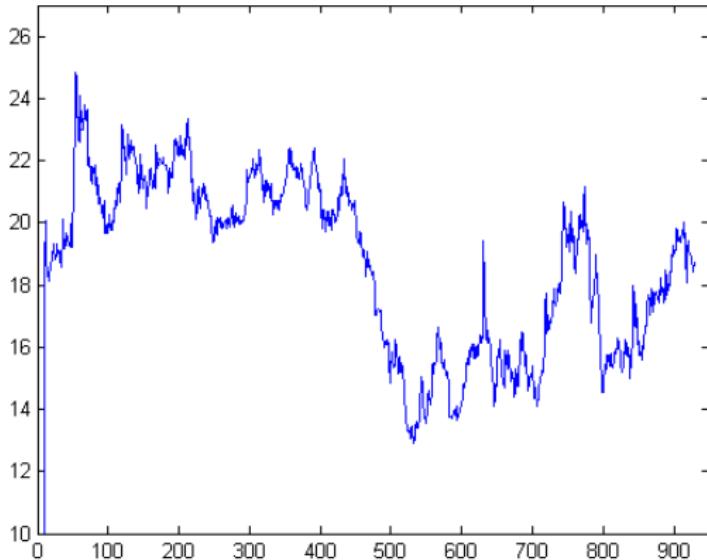
- Przyjmujemy 6 neuronów na warstwie ukrytej.
- Warstwa ukryta ma funkcję aktywacji typu tangens hiperboliczny, warstwa wyjściowa jest liniowa.
- Dane uczące przygotowujemy w sposób pokazany wcześniej.

# Przykłady



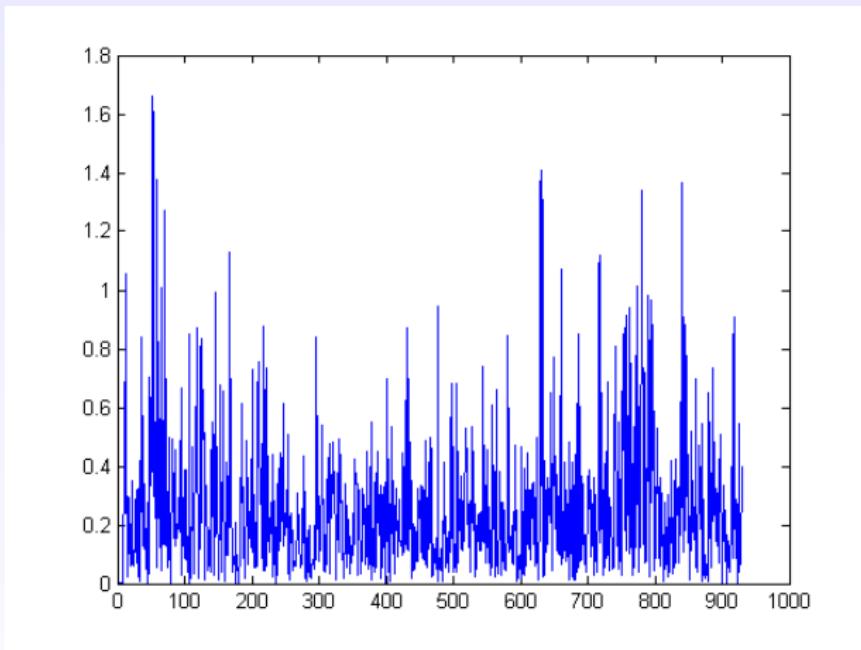
Sygnal uczący.

# Przykłady



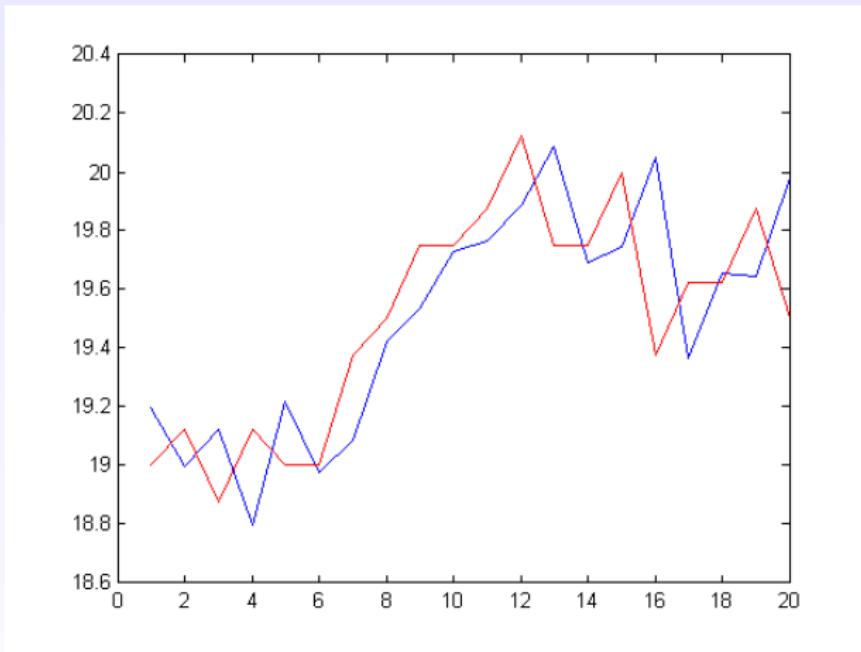
Sygnał zamodelowany przez sieć neuronową.

# Przykłady



Błąd sieci.

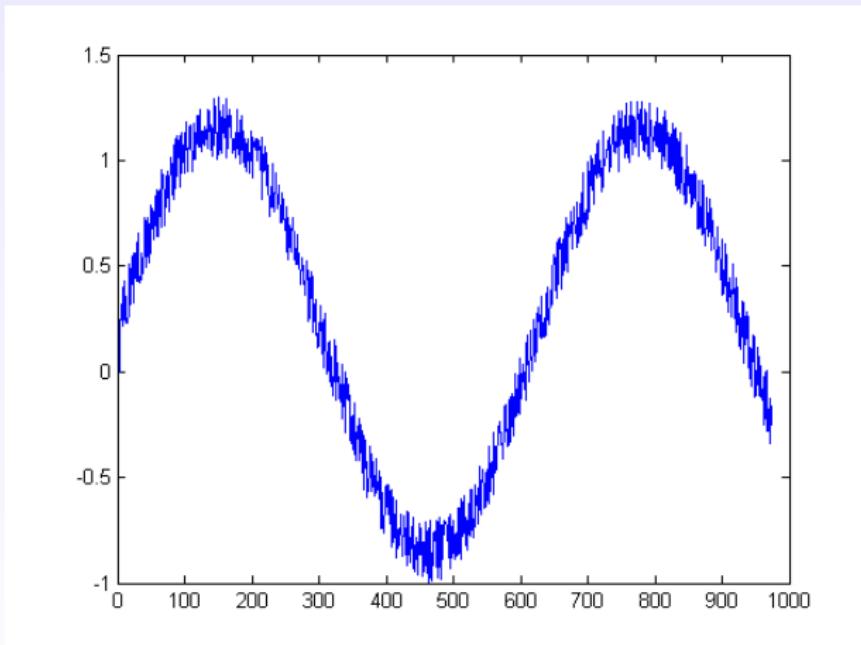
# Przykłady



Prognozowanie danych nie wykorzystanych w uczeniu. Linia niebieska – prognoza, linia czerwona – wartość faktyczna.

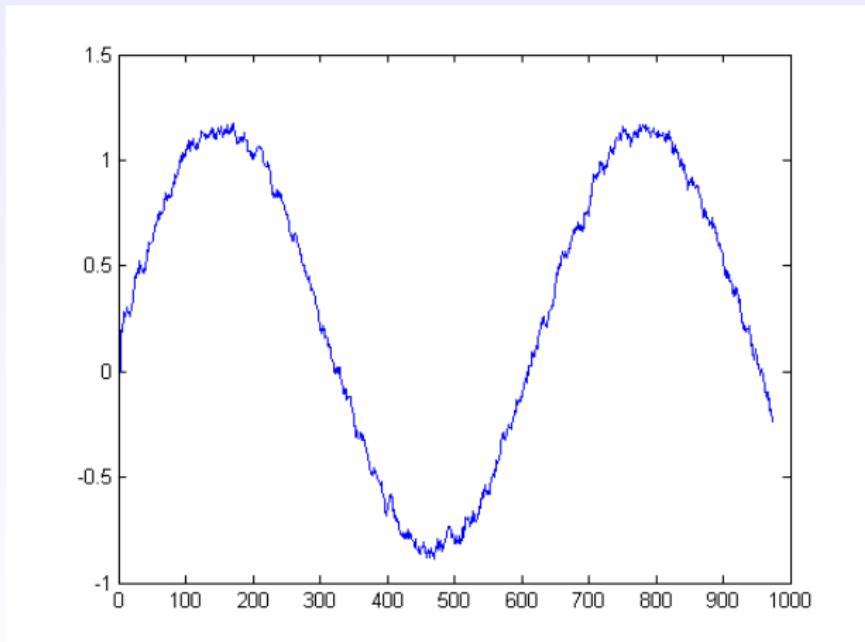
# Przykłady

Sygnał uczący – zaszumiona sinusoida.



# Przykłady

Sygnał zamodelowany przez sieć. Efekt filtracji szumu uzyskujemy stosując prostą sieć złożoną z jednego neuronu liniowego.



# Modelowanie obiektów dynamicznych

Zadania:

- ① identyfikacja,
- ② regulacja.

# Modelowanie obiektów dynamicznych

Ogólny model obiektu dynamicznego ma postać równania różniczkowego:

$$F(y^{(n)}(t), y^{(n-1)}(t), \dots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \dots, \dot{x}(t), x(t)) = 0$$

Dla obiektów rzeczywistych zawsze zachodzi:  $n \geq m$ .

# Modelowanie obiektów dynamicznych

Ogólny model obiektu dynamicznego ma postać równania różniczkowego:

$$F(y^{(n)}(t), y^{(n-1)}(t), \dots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \dots, \dot{x}(t), x(t)) = 0$$

Dla obiektów rzeczywistych zawsze zachodzi:  $n \geq m$ .

Jeśli obiekt jest liniowy, można go opisać równaniem różniczkowym liniowym:

$$\begin{aligned} a_n y^{(n)}(t) + a_{n-1} y^{(n-1)}(t) + \dots + a_1 \dot{y}(t) + a_0 y(t) &= \\ &= b_m x^{(m)}(t) + b_{m-1} x^{(m-1)}(t) + \dots + b_1 \dot{x}(t) + b_0 x(t) \end{aligned}$$

# Modelowanie obiektów dynamicznych

Liniowe równanie różniczkowe:

$$\begin{aligned} a_n y^{(n)}(t) + a_{n-1} y^{(n-1)}(t) + \dots + a_1 y(t) + a_0 y(t) = \\ = b_m x^{(m)}(t) + b_{m-1} x^{(m-1)}(t) + \dots + b_1 \dot{x}(t) + b_0 x(t) \end{aligned}$$

po dyskretyzacji ma formę liniowego równania różnicowego:

$$\begin{aligned} y(k) = A_1 y(k-1) + A_2 y(k-2) + \dots + A_n y(k-n) + \\ + B_0 x(k) + B_1 x(k-1) + \dots + B_m x(k-m) \end{aligned}$$

Równanie takie można zamodelować za pomocą jednego neuronu liniowego.

# Modelowanie obiektów dynamicznych

Nieliniowe równanie różniczkowe:

$$F(y^{(n)}(t), y^{(n-1)}(t), \dots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \dots, \dot{x}(t), x(t)) = 0$$

po dyskretyzacji ma formę nieliniowego równania różnicowego:

$$y(k) = F(y(k-1), y(k-2), \dots, y(k-n), x(k), x(k-1), \dots, x(k-m))$$

# Modelowanie obiektów dynamicznych

Nieliniowe równanie różniczkowe:

$$F(y^{(n)}(t), y^{(n-1)}(t), \dots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \dots, \dot{x}(t), x(t)) = 0$$

po dyskretyzacji ma formę nieliniowego równania różnicowego:

$$y(k) = F(y(k-1), y(k-2), \dots, y(k-n), x(k), x(k-1), \dots, x(k-m))$$

- Równanie takie można zamodelować za pomocą nieliniowej sieci wielowarstwowej.
- $n$  – określa rozmiar okna czasowego dla sygnału wyjściowego, a  $m$  – określa rozmiar okna czasowego dla sygnału wejściowego.
- Przy modelowaniu za pomocą sieci neuronowej, będziemy mieli  $n + m + 1$  wejść i jedno wyjście.

# Przygotowanie danych

Założmy, że chcemy utworzyć model dynamiki zmian cen paliwa w zależności od kursu dolara. Co tydzień wyznaczamy średni kurs dolara i średnią cenę paliwa.

	Kurs	Cena
Tydzień 1	3.20	4.70
Tydzień 2	3.25	4.88
Tydzień 3	3.15	4.73
Tydzień 4	3.10	4.65
Tydzień 5	3.12	4.60
Tydzień 6	3.21	4.68

# Przygotowanie danych

	Kurs	Cena
Tydzień 1	3.20	4.70
Tydzień 2	3.25	4.88
Tydzień 3	3.15	4.73
Tydzień 4	3.10	4.65
Tydzień 5	3.12	4.60
Tydzień 6	3.21	4.68

Jeśli przyjmiemy okno czasowe dla wyjścia  $n = 2$  i dla wejścia  $m = 1$ , próbki uczące będą miały postać:

Próbka 1:	4.70	4.88	3.25	3.15	4.73
Próbka 2:	4.88	4.73	3.15	3.10	4.65
Próbka 3:	4.73	4.65	3.10	3.12	4.60
Próbka 3:	4.65	4.60	3.12	3.21	4.68

# Przygotowanie danych

	Kurs	Cena
Tydzien 1	3.20	4.70
Tydzien 2	3.25	4.88
Tydzien 3	3.15	4.73
Tydzien 4	3.10	4.65
Tydzien 5	3.12	4.60
Tydzien 6	3.21	4.68

Jeśli przyjmiemy okno czasowe dla wyjścia  $n = 2$  i dla wejścia  $m = 1$ , próbki uczące będą miały postać:

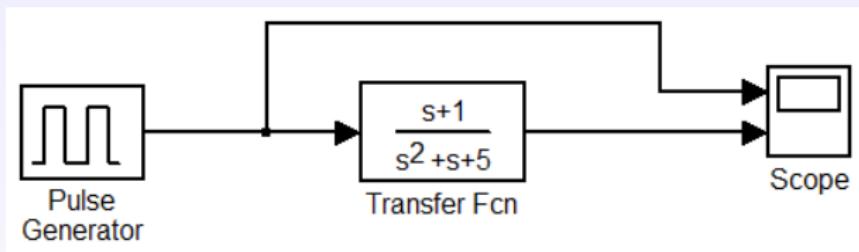
Próbka 1:	4.70	4.88	3.25	3.15	4.73
Próbka 2:	4.88	4.73	3.15	3.10	4.65
Próbka 3:	4.73	4.65	3.10	3.12	4.60
Próbka 3:	4.65	4.60	3.12	3.21	4.68

Sieć nauczona z ich pomocą, będzie modelować równanie różnicowe:

$$y(k) = F(y(k-1), y(k-2), x(k), x(k-1))$$

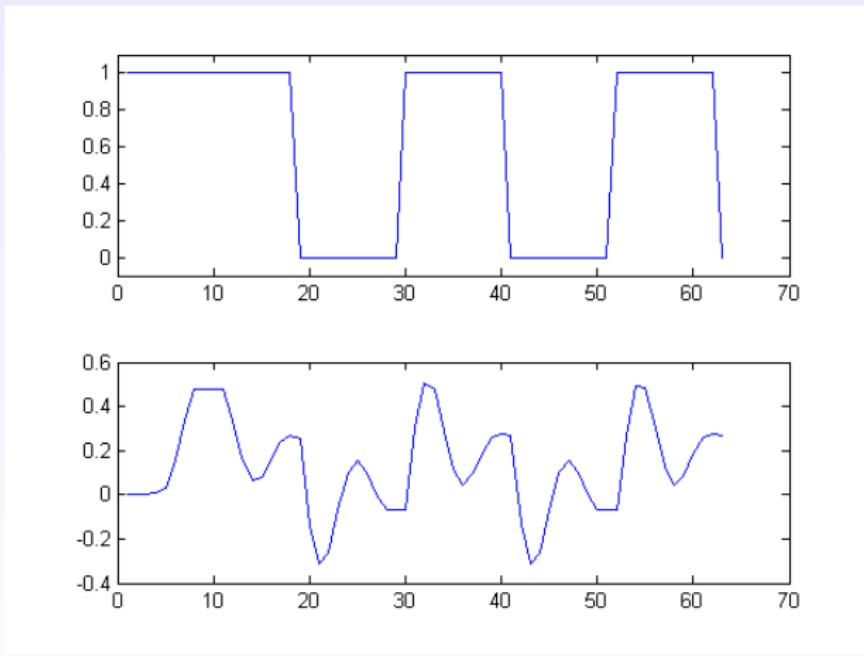
# Przykład

Zamodelowano zachowanie obiektu dynamicznego:



# Przykład

Zarejestrowano sygnał wejściowy i wyjściowy obiektu:



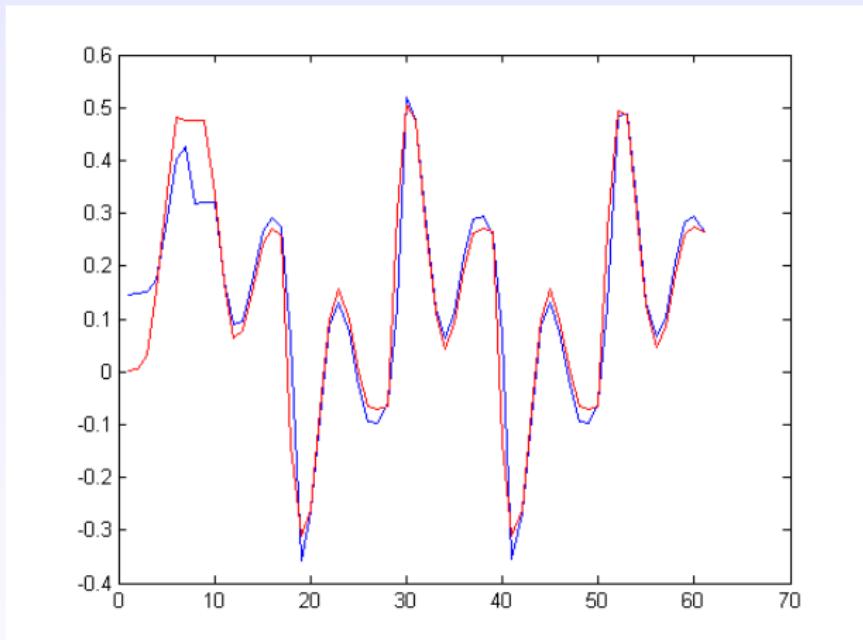
# Przykład

- Przyjmujemy rozmiar okna czasowego wyjścia  $n = 2$  i wejścia  $m = 1$ .
- Sieć będzie modelować równanie różnicowe:

$$y(k) = F(y(k-1), y(k-2), x(k), x(k-1))$$

- Przyjmujemy sieć z jednym neuronem liniowym.
- Dane uczące przygotowujemy w sposób pokazany wcześniej.

# Przykład



Linia czerwona – wyjściowy sygnał uczący, linia niebieska – sygnał wygenerowany przez sieć.

# Sieć neuronowa jako regulator

# Przygotowanie danych uczących

```
@relation labor
@attribute 'duration' real
@attribute 'wage-increase-first-year' real
@attribute 'wage-increase-second-year' real
@attribute 'wage-increase-third-year' real
@attribute 'cost-of-living-adjustment' {'none','tcf','tc'}
@attribute 'working-hours' real
@attribute 'pension' {'none','ret_allw','empl_contr'}
@attribute 'standby-pay' real
@attribute 'shift-differential' real
@attribute 'education-allowance' {'yes','no'}
@attribute 'statutory-holidays' real
@attribute 'vacation' {'below_average','average','generous'}
@attribute 'longterm-disability-assistance' {'yes','no'}
@attribute 'contribution-to-dental-plan' {'none','half','full'}
@attribute 'bereavement-assistance' {'yes','no'}
@attribute 'contribution-to-health-plan' {'none','half','full'}
@attribute 'class' {'bad','good'}
@data
1,5,?,?,40,?,?,2,?,11,'average',?,?,'yes',?,'good'
2,4,5,5.8,?,?,35,'ret_allw',?,?,'yes',11,'below_average',?,'full',?,'full','good'
?,?,?,?,?,38,'empl_contr',?,5,?,11,'generous','yes','half','yes','half','good'
3,3.7,4,5,'tc',?,?,'yes',?,'yes',?,'yes',?,'good'
3,4,5,4,5,5,?,40,?,?,?,12,'average',?,'half','yes','half','good'
2,2,2,5,?,?,35,?,?,6,'yes',12,'average',?,'yes',?,'good'
3,4,5,5,'tc',?,'empl_contr',?,?,'yes',12,'generous','yes','none','yes','half','good'
3,6,9,4,8,2,3,?,40,?,?,3,?,12,'below_average',?,'yes',?,'good'
2,3,7,?,?,38,?,12,25,'yes',11,'below_average','yes','half','yes',?,'good'
1,5,7,?,?,'none',40,'empl_contr',?,4,?,11,'generous','yes','full',?,'good'
3,3,5,4,4,6,'none',36,?,?,3,?,13,'generous',?,'yes','full','good'
2,6,4,6,4,?,?,38,?,?,4,?,15,?,?,'full',?,'good'
```

# Typy danych

- Atrybuty nominalne: nie mają żadnej wartości liczbowej, jedyne relacje to 'równość' i 'nierówność'.
- Atrybuty porządkowe: zdefiniowana jest dla nich relacja porządku.
- Atrybuty interwałowe: zdefiniowana jest miara odległości między wartościami zmiennej. Położenie zera w skali interwałowej jest dowolne.
- Atrybuty rzeczywiste.

# Normalizacja danych

- Normalizacja nie zawsze jest konieczna, ale prawie zawsze zalecana.
- Ujednolicenie ‘ważności’ atrybutów.
- Normalizacja wyjść (w przypadku modeli wielowyjściowych) – ważna z powodu minimalizacji błędu dla każdego wyjścia.
- Łatwiejsza interpretacja wartości wag.
- Niektóre sieci (np. RBF, Kohonena) wymagają normalizacji dla prawidłowej pracy.

# Brakujące atrybuty

- Usuwanie próbek.
- Uzupełnianie atrybutów.
- Podział próbki.
- Dla atrybutów nominalnych – potraktowanie braku jako kolejnej wartości.