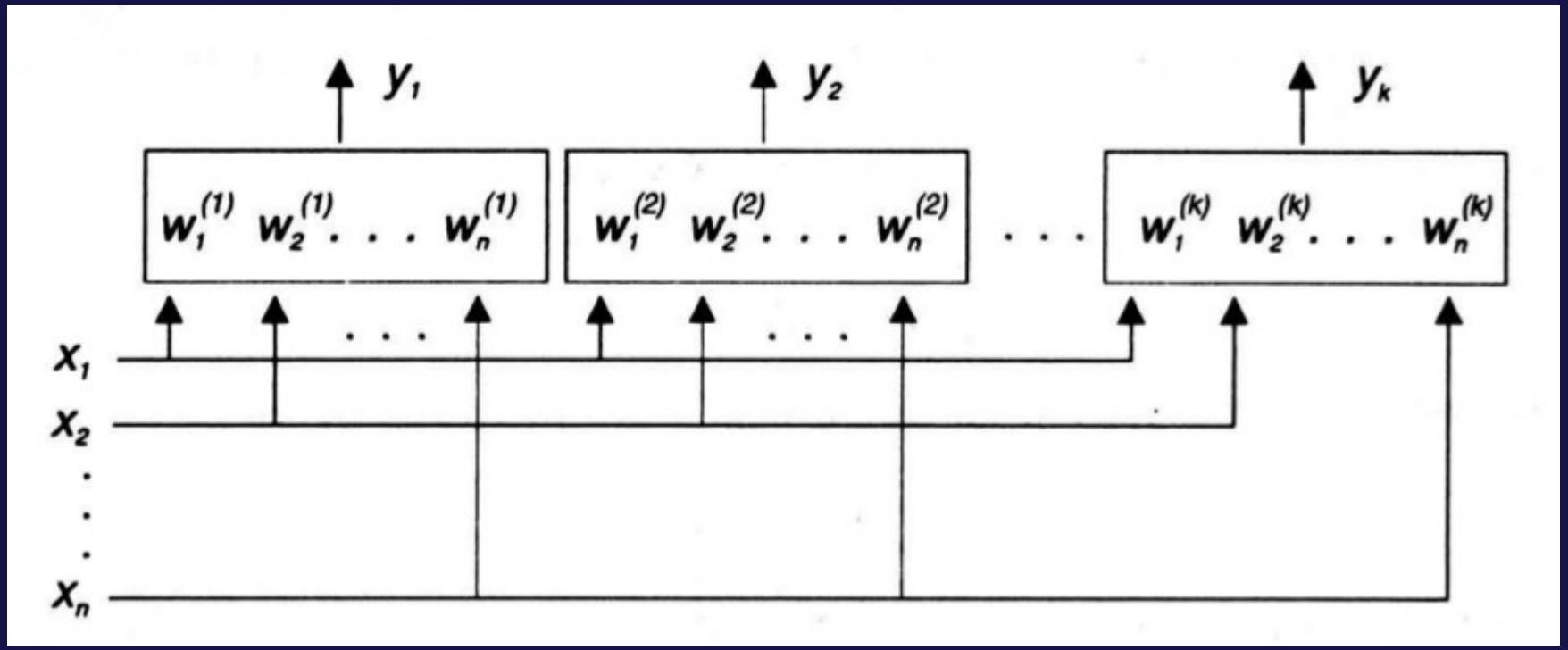
Wstęp do teorii sztucznej inteligencji

Wykład V

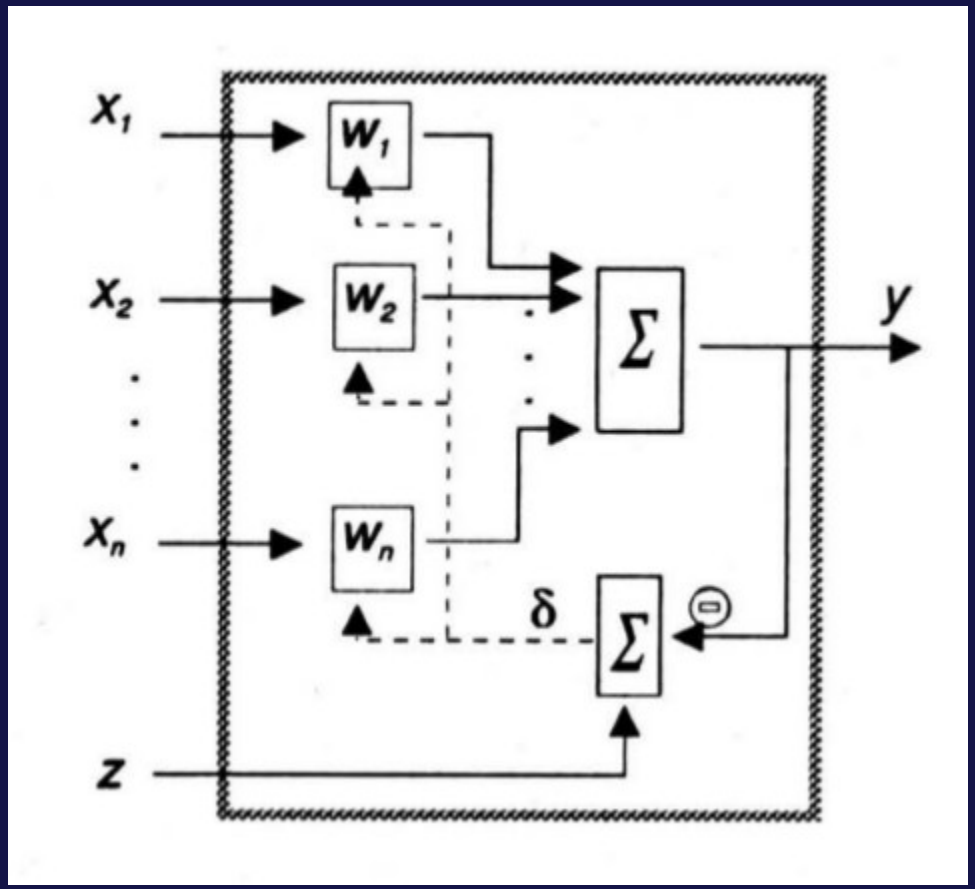
Algorytmy uczenia SSN

Modele sieci neuronowych. SSN = Architektura + Algorytm



Wagi i wejścia dla sieci neuronuowej: reprezentacja macierzowa

$$\mathbf{W_k} = \begin{bmatrix} w_1^{(1)} & w_2^{(1)} & \dots & w_n^{(1)} \\ w_1^{(2)} & w_2^{(2)} & \dots & w_n^{(2)} \\ \vdots & \vdots & & \vdots \\ w_1^{(k)} & w_2^{(k)} & \dots & w_n^{(k)} \end{bmatrix}$$



$$\delta_i = z_i - y_i$$

$$W(k+1)_{ij} = W(k)_{ij} + \eta\, \delta_i\, X_j$$

X – wektor sygnałów wejsciowych
Y – wektor sygnałów wyjsciowych
W – macierz wag
Z – wektor wejsciowych sygnałów porządanych
δ – wektor błedu
η – współczynnik uczenia, zwykle z przedziału [0-1]

Metoda uczenia AdaLiNe (Adaptive Linear Network);
Uczenie z nauczycielem (supervised learning)

Funkcja błędu:

$$E = \sum_{k=1}^{p} (y_i^{(k)} - d_i^{(k)})^2$$

gdzie p – liczba wzorców uczących

Uczenie bez nauczyciela:

• Dobór wag wiąże się z wykorzystaniem bądź to konkurencji neuronów między sobą (strategia Winner Takes All – WTA lub Winner Takes Most – WTM), bądź korelacji sygnałów uczących (metody hebbowskie);

• W uczeniu bez nauczyciela na etapie adaptacji neuronu nie jesteśmy w stanie przewidzieć sygnału wyjściowego neuronu;

Dotychczas zajmowaliśmy się:

• regułą perceptronową (z nauczycielem, sygnał uczący jest różnicą miedzy wartością rzeczywistą a pożądaną)

Nowe idee:

• reguła Hebba (bez nauczyciela, sygnałem uczącym jest sygnał wyjściowy)

• reguła delta (dla neuronów z ciągłymi funkcjami aktywacji I nadzorowaniem). Chodzi o minimalizację kwadratowego kryterium błędu.

• reguła korelacyjna (poprawka każdej składowej wektora wag jest proporcjonalna do iloczynu odpowiedniej składowej obrazu wejściowego i pożądanego przy tym wzorca wyjścia) – metoda więć nieco podobna do uczenia z nauczycielem.

• reguła 'wygrywający bierze wszystko' różni się zdecydowanie od pozostałych tu opisanych. Jest ona przykładem nauki z rywalizacją.

Model D. Hebba

W modelu Hebba przyrost wagi Δw_{ij} w procesie uczenia jest proporcjonalny do iloczynu sygnałów wejsciowego x_i oraz wyjściowego y_j neuronów połączonych wagą w_{ij}.

$$w_{ij}(k+1) = w_{ij}(k) + \eta \cdot x_i(k) \cdot y_j(k)$$

Model Hebba ma być odwzorowaniem sytuacji w sieci “prawdziwych” neuronów, gdy sygnał ulega wzmocnieniu przy częstym powtarzaniu, gdy następuje wtedy lepsze zapamietywanie.

Powyższa formuła ma jednak ograniczone zastosowanie w praktyce: matematycznie nie może bowiem opisywać procesów, albowiem (udowodniono to ogólnie, ale jest to też widoczne tutaj) będzie prowadzić do eksponencjalnej rozbieżności wyników.

Przykład rozbieżności modelu Hebba w przypadku najprostszym, 1 nueron, 1 wejscie, 1 wyjście:

$$y(0) = w \cdot x$$
$$w(1) = w + \eta \cdot x \cdot y(0)$$
$$y(1) = w(1) \cdot x = (w + \eta \cdot x \cdot y(0)) \cdot x = w \cdot x + \eta \cdot x \cdot y(0) \cdot x =$$
$$= y(0) \cdot (1 + \eta \cdot x \cdot x)$$
$$y(2) = y(1) \cdot (1 + \eta \cdot x \cdot x) = y(0) \cdot (1 + \eta \cdot x \cdot x)^2$$

Ogólnie:

$$y(k) = y(0) \cdot (1 + \eta \cdot x \cdot x)^k$$

A więc mamy do czynienia z rozbieżnością eksponencjalną. Udowodniono ogólnie, że rozbieżnoś eksponencjalna istnieje w przypadku dowolnie złożonej sieci.

Uogólniony Algorytm Hebba
(Generalized Hebbian Algorithm)

… to jedna z odmian modelu Hebba, w której unika się eksponencjalnej rozbieżności, poprzez wprowadzenie negatywnego sprzężenia zwrotnego do zmiany wag:

$$\Delta w_{ij} = \eta \left(y_j x_i - y_j \sum_{k=1}^{j} w_{ik} y_k \right),$$

W przypadku dwukomórkowej sieci Kohonena z dwoma wejściami i dwoma wyjściami, bedziemy mieli następujące równania na zmianę wag:

```
w11= w11 + eta * y1 * (x1 - w11*y1);
w12= w12 + eta * y2 * (x1 - w11*y1 - w12*y2);
w21= w21 + eta * y1 * (x2 - w21*y1);
w22= w22 + eta * y2 * (x2 - w21*y1 - w22*y2);
```

Uogólniony Algorytm Hebba w Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Ten plik zapisujemy pod nazwą z rozszerzeniem .py (np. uczenie2.py)
# i uruchamiamy komenda: python uczenie4.py (albo, lepiej, otwieramy plik w IDLE, w przypadku Windows)
# Siec dwukomorkowa Kohonena (kazde wejscie połączone do każdej komórki):
# 2 wejscia x1, x2 dla komorki 1, 1 wyjscie y1.
# Te same wejscia dla komorki 2. Komorka 2 ma 1 wyjscie y2.
# DEFINIUJEMY wektor wejść:
x1 = 0.5; x2 = 0.9;
# Macierz wag W bedzie więc mieć 4 składowe, w11 i w12 dla komórki pierwszej
# oraz w21 i w22 dla komórki drugiej. Przyjmijmy wagi poczatkowe.
w11= 0.5; w12=0.7;
w21= 0.3; w22=0.5;
# Uogólniony Algorytm HEBBA
# Współczynnik uczenia eta niechaj wynosi 0.5. DEFINIUJEMY:
eta = 0.1;
# Sumowanie S bedzie dane wzorami: y1 = w11*x1 + w12*x2, y2 = w21*x1 + w22*x2
# DEFINIUJEMY funkcje y. Niechaj bedzie ona dana ogólnie w ten sposób:
def y(w1, w2):
        return w1*x1 + w2*x2;
# W modelu HEBBA w PIERWSZYM KROKU OBLICZEN przyjmujemy wartości y1, y2=1 dla obliczeń korekty wag
# y1=y(w11, w12);y2=y(w21, w22);
y1=1;y2=1;
# Wypiszmy pierwszym razem wartości początkowe:
print `w11`+"\t"+`w12`+"\t"+`w21`+"\t"+`w22`+"\t"+`y1`+"\t"+`y2`;

# KROK DRUGI OBLICZEN. Obliczamy nowe wartości macierzy wag w modelu HEBBA:
w11= w11 + eta * y1 * (x1 - w11*y1);
w12= w12 + eta * y2 * (x1 - w11*y1 - w12*y2);
w21= w21 + eta * y1 * (x2 - w21*y1);
w22= w22 + eta * y2 * (x2 - w21*y1 - w22*y2);
# KROK TRZECI OBLICZEN
# Zobaczmy, jakie sa nowe wartosci macierzy w oraz jaka jest nowa wartosc y:
print "w11="+`w11`+"\tw12="+`w12`+"\tw21="+`w21`+"\tw22="+`w22`+"\ty1="+`y1`+"\ty2="+`y2`;

# POWTARZAMY KROKI od 1 do 3 (w pętli), az do czasu gdy y staje sie dostatecznie bliskie z.
for i in range(0,50,1):
        y1=y(w11, w12);y2=y(w21, w22);
        w11= w11 + eta * y1 * (x1 - w11*y1);
        w12= w12 + eta * y2 * (x1 - w11*y1 - w12*y2);
        w21= w21 + eta * y1 * (x2 - w21*y1);
        w22= w22 + eta * y2 * (x2 - w21*y1 - w22*y2);
        print `i`+"\t"+`w11`+"\t"+`w12`+"\t"+`w21`+"\t"+`w22`+"\t"+`y1`+"\t"+`y2`;
```

Algorytm Oja – odmiana Uogólnionego Algorytmu Hebba

$$w_{ij}(k+1) = w_{ij}(k) + \eta \cdot y_j(k) \cdot (x_i(k) - y_i(k) \cdot w_{ij}(k))$$

- jest stabilny (zbieżny)
- ponoć dobrze opisuje “rzeczywiste” sieci neuronowe

W przypadku dwukomórkowej sieci Kohonena z dwoma wejściami i dwoma wyjściami, bedziemy mieli następujące równania na zmianę wag:

```
w11= w11 + eta * y1 * (x1 - w11*y1);
w12= w12 + eta * y2 * (x1 - w12*y2);
w21= w21 + eta * y1 * (x2 - w21*y1);
w22= w22 + eta * y2 * (x2 - w22*y2);
```

A więc niewielka różnica w porównaniu z Uogólnionym Algorytmem Hebba:

```
w11= w11 + eta * y1 * (x1 - w11*y1);
w12= w12 + eta * y2 * (x1 - w11*y1 - w12*y2);
w21= w21 + eta * y1 * (x2 - w21*y1);
w22= w22 + eta * y2 * (x2 - w21*y1 - w22*y2);
```

Wystarczy zmienić te 4 linijki w kodzie powyżej, by otrzymać model Oja.

Algorytm Oja w Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Ten plik zapisujemy pod nazwą z rozszerzeniem .py (np. uczenie2.py)
# i uruchamiamy komenda: python uczenie4.py (albo, lepiej, otwieramy plik w IDLE, w przypadku Windows)
# Siec dwukomorkowa Kohonena (kazde wejscie połączone do każdej komórki):
# 2 wejscia x1, x2 dla komorki 1, 1 wyjscie y1.
# Te same wejscia dla komorki 2. Komorka 2 ma 1 wyjscie y2.
# DEFINIUJEMY wektor wejść:
x1 = 0.5; x2 = 0.9;
# Macierz wag W bedzie więc mieć 4 składowe, w11 i w12 dla komórki pierwszej
# oraz w21 i w22 dla komórki drugiej. Przyjmijmy wagi poczatkowe.
w11= 0.5; w12=0.7;
w21= 0.3; w22=0.5;
# Uogólniony Algorytm HEBBA
# Współczynnik uczenia eta niechaj wynosi 0.5. DEFINIUJEMY:
eta = 0.1;
# Sumowanie S bedzie dane wzorami: y1 = w11*x1 + w12*x2, y2 = w21*x1 + w22*x2
# DEFINIUJEMY funkcje y. Niechaj bedzie ona dana ogólnie w ten sposób:
def y(w1, w2):
        return w1*x1 + w2*x2;
# W modelu HEBBA w PIERWSZYM KROKU OBLICZEN przyjmujemy wartości y1, y2=1 dla obliczeń korekty wag
# y1=y(w11, w12);y2=y(w21, w22);
y1=1;y2=1;
# Wypiszmy pierwszym razem wartości początkowe:
print `w11`+"\t"+`w12`+"\t"+`w21`+"\t"+`w22`+"\t"+`y1`+"\t"+`y2`;

# KROK DRUGI OBLICZEN. Obliczamy nowe wartości macierzy wag w modelu HEBBA:
w11= w11 + eta * y1 * (x1 - w11*y1);
w12= w12 + eta * y2 * (x1 - w12*y2);
w21= w21 + eta * y1 * (x2 - w21*y1);
w22= w22 + eta * y2 * (x2 - w22*y2);
# KROK TRZECI OBLICZEN
# Zobaczmy, jakie sa nowe wartosci macierzy w oraz jaka jest nowa wartosc y:
print "w11="+`w11`+"\tw12="+`w12`+"\tw21="+`w21`+"\tw22="+`w22`+"\ty1="+`y1`+"\ty2="+`y2`;

# POWTARZAMY KROKI od 1 do 3 (w pętli), az do czasu gdy y staje sie dostatecznie bliskie z.
for i in range(0,50,1):
        y1=y(w11, w12);y2=y(w21, w22);
        w11= w11 + eta * y1 * (x1 - w11*y1);
        w12= w12 + eta * y2 * (x1 - w12*y2);
        w21= w21 + eta * y1 * (x2 - w21*y1);
        w22= w22 + eta * y2 * (x2 - w22*y2);
        print `i`+"\t"+`w11`+"\t"+`w12`+"\t"+`w21`+"\t"+`w22`+"\t"+`y1`+"\t"+`y2`;
```

Jeszcze inna odmiana Uogólnionego Algorytmu Hebba:

Uwzględnia się “zapominanie” przez sieć tego, co już sięc została nauczona”, czyli własnych wag:

$$w_{ij}(k+1) = w_{ij}(k)\cdot(1-\gamma) + \eta \cdot y_j(k) \cdot (x_i(k) - {}_i(k)\cdot w_{ij}(k)),$$

gdzie $\gamma<1$

Analogie

Rozwiązywanie problemu SSN (uczenie SSN) jest jak szukanie drogi, po której rzeka spływa ze źródła górskiego (w wielowymiarowej przestrzeni wag).

W przypadku problemu z nauczycielem, pokazujemy jej drogę.

W przypadku problemu bez nauczyciela, jedynie poznajemy, dokąd rzeka dopłynęła. Czasem, tylko czasem, możemy odgadnąć jej drogę, znając jej ujście do morza.

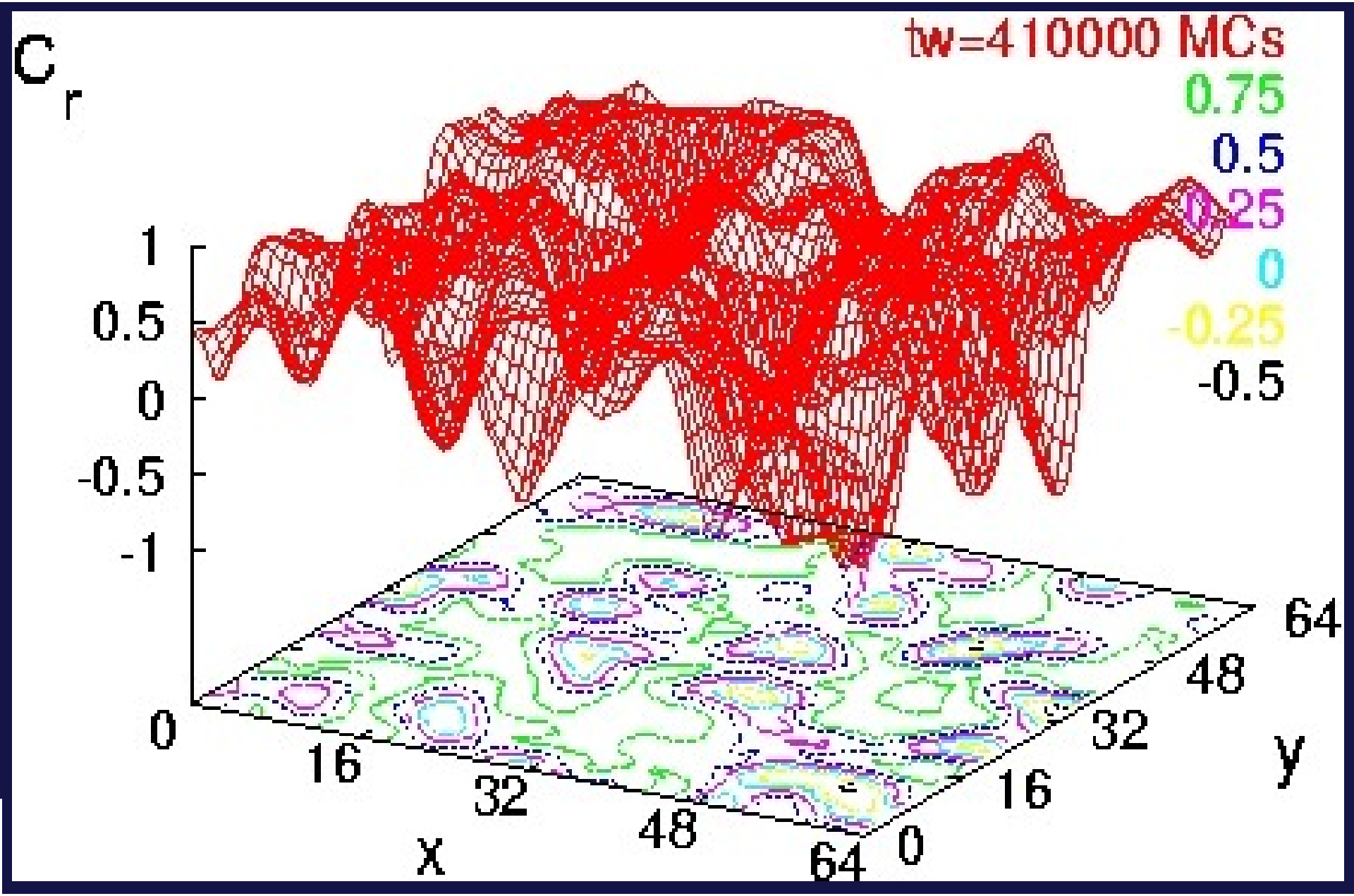

visualparadox.com

Analogie z innych dziedzin:

- Teoria szkła. Albo szkła spinowego.

- Opis niestabilności w wielu układach mezoskopowych.

- Procesy modelowania komputerowego. Rola parametru eta.


C r
tw=410000 MCs
0.75
0.5
0.25
0
-0.25
-0.5
1
0.5
0
-0.5
-1
0
16
32
48
64
x
0
16
32
48
64
y

Późniejsze modele sieci neuronowych:

To m.in. F. Rosenblatt – teoria dynamicznych systemów neuronowych modelujących mózg, oparta na modelu perceptronowym komórki nerwowej.

Pojawiają sie pytania i uogólnienia

Który z tych modeli najlepiej opisuje “rzeczywistą” sieć neuronową?

Co my naprawdę wiemy o funkcjonowaniu pojedynczego neuronu?
Na ile modele matematyczne opisują ich fizyczne funkcjonowanie?

Teoria SSN tworzona jest nie tylko z myślą o modelowaniu naturalnych sieci neuronowych. A nawet bardziej jest modelem do opisu złożonych systemów (np. społecznych). Do opisu i przewidywania złożonych sytuacji, gdy określonym stanom wejsciowym odpowiadają określone stany wyjściowe, ale nie znamy mechanizmów tego, co dzieje się w “środku” procesu.

Zgaduję, że istnieje w tej dziedzinie bardzo wiele jeszcze do zrobienia...

Studenckie prezentacje na wybrane tematy (5-15 minut)

1. Co to jest sieć Hamminga? Albo Hopfielda? (stosowane w uczeniu) – 13/XI

2. Pompa sodowo-potasowa – 27/XI
3. EEG, fale mózgowe. Jak są badane? Na ile model SSN jest przydatny do ich analizy? - 27/XI

4 . Model Hodgkin-Huxley'a propagacji sygnału – 11/XII
5 . Niespodzianka (oby na temat ;) – 11/XII

6. Historia badań sztucznych sieci neuronowych i sztucznej inteligencji. Rola polskiej nauki? - 8/I
7. Sztuczna Inteligencja – 8/I

8. Chemo-fizjologia neuronu. Jak duże potencjały elektryczne występują? Jak można je mierzyć? - 22/I

6. Dostępne oprogramowanie do modelowania SSN, czy to na poziomie dydaktycznym (jako pomoc dla studentów i naczycieli) czy też zaawansowane systemy komputerowe.
7. Elektroniczne (sprzętowe) realizacje układów logicznych (bramki AND, OR. NOT, XOR, etc)
8. Czy i na ile wykorzystywane modele matematyczne SSN odpowiadają “prawdziwym” SN ?

	27/XI	11/XII	8/I	22/I