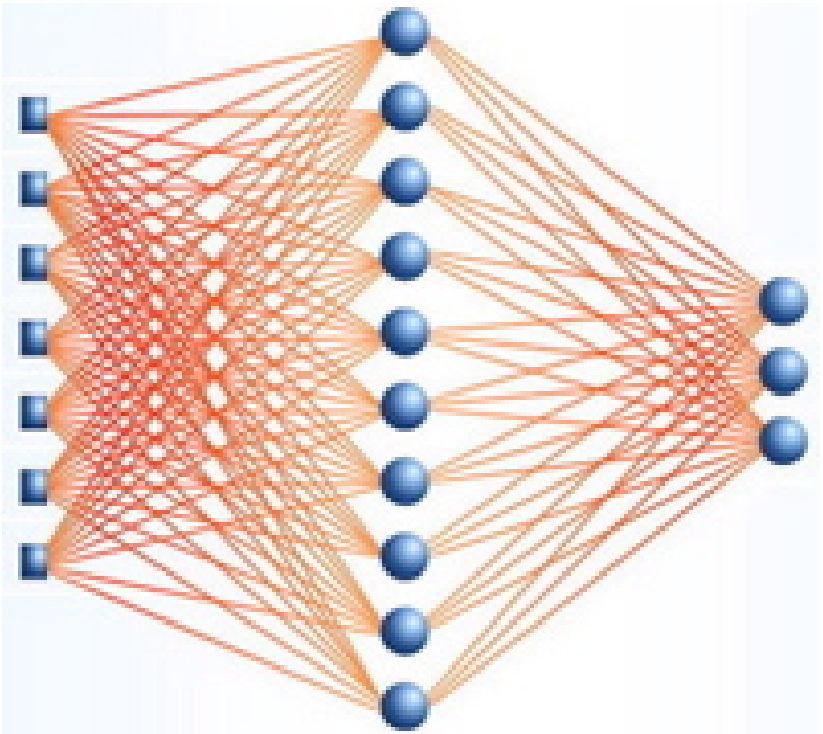
Sieci Rekurencyjne

Nguyen Hung Son

Outline

Na poprzednim wykładzie

- Uczenie perceptronu
- Algorytm propagacji wstecznej w uczeniu sieci wielu-warstwowej



Dodatkowe uwagi

- W przypadku gdy funkcja pobudzenia neuronów jest sigmoidalną funkcją logistyczną (tangensem hiperbolicznym), poprawka wagi jest istotnie prostsza do wyliczenia, gdyż wartość pochodnej jest dana prostą formułą arytmetyczna zależną tylko od wartości funkcji.
- Najprostsza metoda gradientowa propagacji wstecznej (simple backpropagation) na wiele ograniczeń i zwykle nie jest praktyczna w stosowaniu (np. minima lokalne). Dlatego korzysta się z licznych modyfikacji i usprawnień:
  - wykorzystania czynnika momentu,
  - algorytmów heurystycznych (Quickprop) i
  - innych metod wyznaczania kierunku zmiany wagi (metoda gradientów sprzężonych).
- Problemem otwartym pozostaje także dobranie architektury sieci (liczby neuronów, układu warstw i połączeń) oraz parametrów algorytmu uczenia, aby dla danego zbioru treningowego sieć była w stanie aproksymować zadaną zależność.

Możliwości sieci neuronowych

- Okazuje się, że sieci wielowarstwowe o nieliniowych (sigmoidalnych) funkcjach aktywacji neuronów mają, przynajmniej teoretycznie, bardzo duże możliwości. Można za ich pomocą aproksymować w zasadzie wszelkie "rozsądne" funkcje z $\mathbb{R}^s$ w $\mathbb{R}^m$.
- Niestety, twierdzenia okreslające możliwości sieci neuronowych są egzystencjalne. Ich konstruktywność pozostawia wiele do życzenia. Dlatego w praktyce parametry takie jak liczba i układ neuronów (architektura sieci), stałe i strategia uczenia, funkcja aktywacji itp., muszą być wyznaczone heurystycznie na podstawie znajomości problemu. Istnieją pewne wskazówki wspomagajace wybór architektury i parametrów sieci, ale mają one charakter niescisły.
- Można pokazać, że nawet dla bardzo prostej, ustalonej architektury sieci i ustalonej konfiguracji parametrów uczenia, znalezienie optymalnego układu wag jest co najmniej NP-trudne.

Istnienie Sieci o ustalonej strukturze

Niech $I = [0, 1]$, $S = I^s$ będzie s-wymiarową kostką (domkniętą), $0 \leq x_p \leq 1$, $p = 1, .., s$.

Twierdzenie Kołmogorowa-Arnolda-Sprechera

Istnieje s stałych $0 < \lambda_p \leq 1, p = 1, ..., s$, i $2s + 1$ funkcji $\phi_q(x), q = 0, ..., 2s$ określonych na I, ściśle rosnących i należących do klasy Lip_α, dla $\alpha > 0$ takich, że dla każdej funkcji ciągłej f określonej na S istnieje funkcja ciągła $g(u)$, dla której:

$$f(x_1, ..., x_s) = \sum_{q=0}^{2s} g(\lambda_1 \phi_q(x_1) + ... + \lambda_s \phi_q(x_s)) = \sum_{q=0}^{2s} g\left(\sum_{p=1}^{s} \lambda_p \phi_q(x_p)\right)$$

Wniosek – Hecht-Nielsen

Dla dowolnej funkcji ciągłej $h : [a, b]^s \longrightarrow \mathbb{R}^m$ istnieje czterowarstwowa sieć neuronowa zupełna o s wejściach, odpowiednio $m(2s + 1)$ i $s(2s + 1)$ neuronach w warstwach ukrytych i m wyjściach, która oblicza h.

Outline

Wybrane typy zastosowań

- **Predykacja:** Sieci neuronowe są często wykorzystywane, aby na podstawie pewnych danych wejściowych przewidywała dane wyjściowe (np. w układach sterowania). Ważną zaletą jest to, że sieć może nauczyć się przewidywania sygnałów wyjściowych bez jawnego zdefiniowania związku między danymi wejściowymi a wyjściowymi.
- **Klasyfikacja i rozpoznawanie:** Zadanie polega na przewidywaniu identyfikatora klasy, do której dany obiekt należy na podstawie wcześniej zaobserwowanych (nauczonych) przykładów..
- **Kojarzenie danych:** Sieci neuronowe, dzięki zdolności uczenia się i uogólniania doświadczeń, pozwalają zautomatyzować procesy wnioskowania i pomagają wykrywać istotne powiązania pomiędzy danymi. Sieci neuronowe, dzięki zdolności uczenia się i uogólniania doświadczeń, pozwalają zautomatyzować procesy wnioskowania i pomagają wykrywać istotne powiązania pomiędzy danymi.

Wybrane typy zastosowań (c.d.)

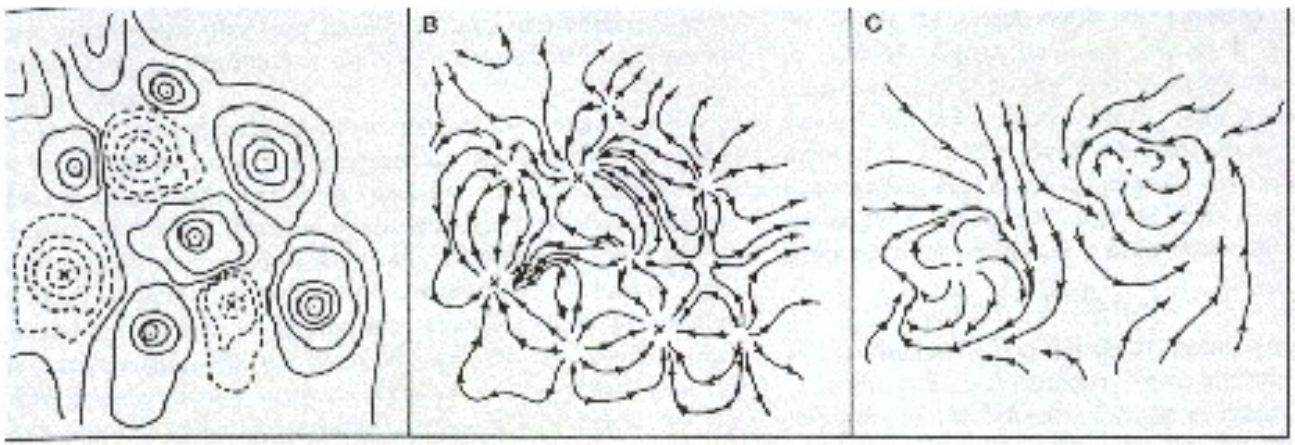
- **Analiza danych:** Zadanie polega na znalezieniu związków pomiędzy danymi. Realizacja tego zadania przez sieci neuronowe daje nowe możliwości w zakresie prowadzenia analiz ekonomicznych.
- **Filtracja sygnałów** Dane gospodarcze pochodzące z różnych źródeł są zakłócone. Klasyczne metody eliminacji "szumów" pozwalają usunąć zakłócenia o charakterze losowym, lecz nie dają podstaw do eliminacji przekłamań systematycznych.
- **Optymalizacja** Sieci neuronowe - zwłaszcza sieci Hopfielda - dobrze nadają się do optymalizacji decyzji gospodarczych. Doświadczalnie potwierdzono możliwości sieci do rozwiązywania zadań optymalizacji statycznej i dynamicznej. Szczególnie ciekawe jest zastosowanie sieci do optymalizacji kombinatorycznej i zagadnień (NP)trudnych obliczeniowo, np. TSP.

Różne modele sieci neuronowych

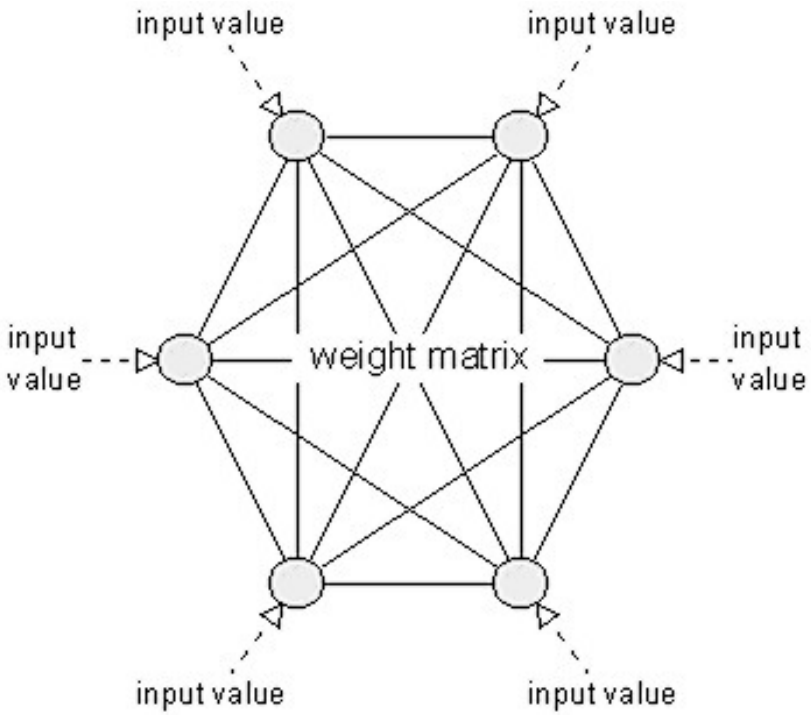
Model sieci	Zastosowanie
Wielowarstwowa z propagacją wsteczną	Klasyfikacja Predykcja
Sieć rekurencyjna (Hopfielda)	Pamięć skojarzeniowa
Sieć konkurencyjna samo-organizująca sieć (Kohonen)	PCA Analiza skupień

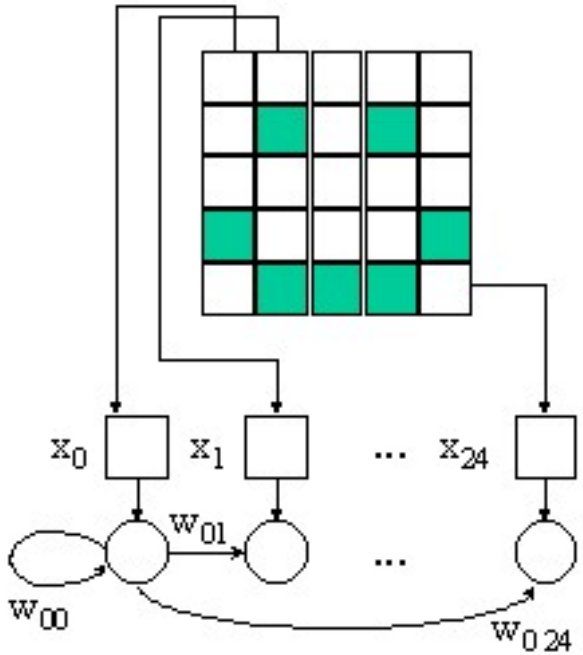
Outline

Idea:



Sieci Hopfielda





- Zbadane przez Hopfielda;
- Zwane również sieciami auto-asocjacyjnymi;
- Są to jedno-warstwowe sieci z pełnym połączeniem typu "każdy z każdym";
- Każdy neuron ma bipolarne wartość wejść i wyjść;

Operacje w sieciach Hopfielda:

- Zainicjuj wartości początkowe do sieci;
- Czekaj aż sieć się ustabilizuje;
  - To działa w czasie dyskretnym: $t_1, t_2, ..., t_N$;
  - wartości neuronów w chwili t_n zależy od wartości neuronów w chwili t_{n-1}:

$$u_i(t_n) = \sum_j w_{ij} y_j(t_{n-1})$$

$$y_i(t_n) = \begin{cases} 1 & \text{jeśli } u_i(t_n) > T_i \\ y_i(t_{n-1}) & \text{gdy } u_i(t_n) = T_i \\ -1 & \text{jeśli } u_i(t_n) < T_i \end{cases}$$

- Odczytaj wartości neuronów jako wynik obliczeń;
- Istnieją modele synchroniczne i asynchroniczne;

Dlaczego to działa?

- Stanem sieci nazywamy wektor wartości neuronów w danym momencie

$$\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \dots \\ y_n(t) \end{pmatrix} = (y_1(t), y_2(t), \dots, y_n(t))^T$$

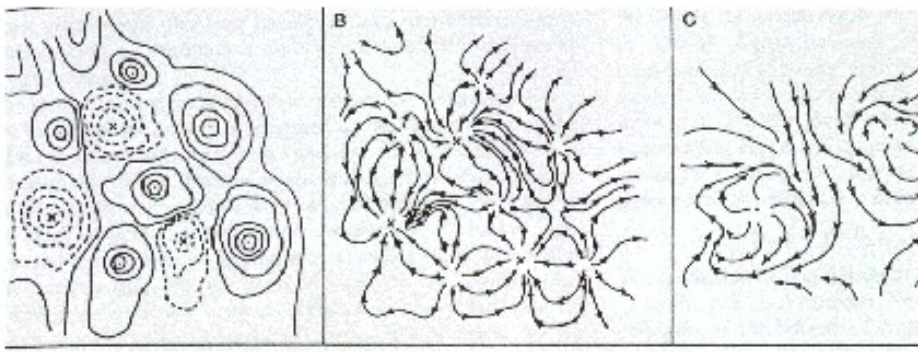
- Charakterystyczną własnością dla bieżącego stanu $\mathbf{y} = (y_1, \dots, y_n)^T$ sieci Hopfielda jest funkcja energii:

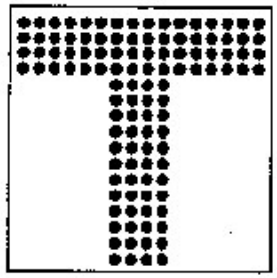
$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{T}^T\mathbf{y} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j + \sum_{i=1}^{n} T_i y_i$$

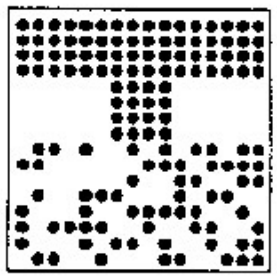
- Hopfield pokazał, że jeśli macierz wag jest symetryczna ($w_{ij} = w_{ji}$) oraz ma zerową przekątną ($w_{ii} = 0$), to funkcja energii jest nierosnąca:

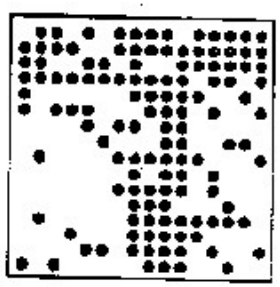
$$\Delta E = E(\mathbf{y}(t+1)) - E(\mathbf{y}(t)) \leq 0$$

- Istnieją więc stany o minimalnej energii (lokalnie);
- Takie stany możemy traktować jako *atraktory*, których zbiegają inne stany:




Original 'T'


half of image corrupted by noise


20% corrupted by noise (whole image)

- Sieć Hopfielda jest pewnym rodzajem *pamięci asocjacyjnej;*

Problem: jak zaprojektować sieć, która pamięta zadane wzorce?

Ustalenie wartości wag:

UWAGA: wagi się nie zmieniają w procesie uczenia się;

Maciecz wag musi spełniać warunki:

$$w_{ij} = w_{ji} \qquad \text{(symetryczność)}$$

$$w_{ii} = 0 \qquad \text{(zerowa przekątna)}$$

Zwykle chcemy, aby sieć ustabilizowała w jednym z wektorów: $\vec{v_1}, ..., \vec{v_P}$, gdzie

$$v_p = (x_1^p, ...x_n^p)$$

Aby to zapewnić możemy stosować reguły Hebba:

$$w_{ij} = \frac{1}{m} \sum_{p=1}^{P} x_i^p x_j^p$$

lub

$$\mathbf{W} = \frac{1}{P} \sum_{p=1}^{P} v_p v_p^T - \mathbf{I}$$

Przykład Niech

$$v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad v_2 = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

Wówczas

$$\mathbf{W} = \frac{1}{2}(v_1 v_1^T + v_2 v_2^T) - \mathbf{I} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Łatwo można sprawdzić, że jeśli $\mathbf{y}(t_{n-1}) = v_i$ to

$$\mathbf{W} v_1 = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

czyli $\mathbf{y}(t_n) = v_i$

- Hopfield pokazał, że pojemność sieci wynosi

$$P_{\max} = 0.15n$$

- Przy pewnych założeniach

$$P_{\max} = n/2\log n$$

Podsumowując:

pojemność $P_{\max}$ jest proporcjonalna do n, jeśli dopuszczamy mały procent bitów w każdym wzorcu, a proporcjonalna do $n/2\log n$, gdy nalegamy, aby większość lub prawie wszystkie wzorce były odtwarzane idealnie.

Projektowanie sieci Hopfielda

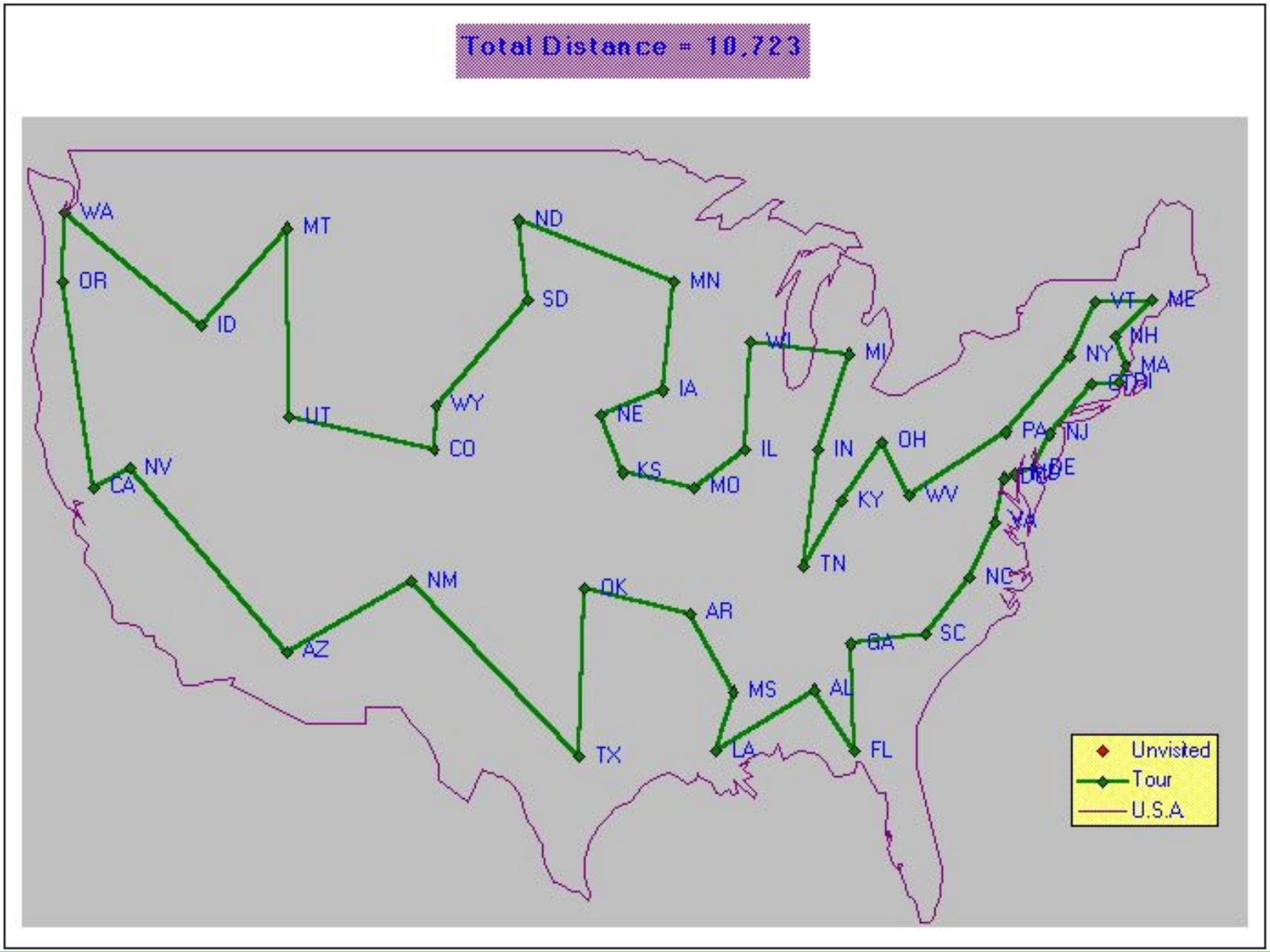
Zasada:

Działanie sieci Hopfielda sterowane jest przez dążenie do minimalizacji pewnej funkcji, którą zwyczajowo utożsamia się z „energią" sieci.

$$E(\mathbf{y}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j - \sum_{i=1}^{n} x_i(t)y_i(t) + \sum_{i=1}^{n} T_i y_i$$

- $x_i(t)$ jest dodatkowym sygnałem wejściowym, który jest niezbędny do wprowadzenia danych startowych i dla pobudzenia sieci do działań.
- Przypomnijmy, że „energia" $E(t)$ we wszystkich procesach w sieci Hopfielda zawsze maleć lub pozostawać bez zmian – nigdy nie może rosnąć.
- Oznacza to, że znając funkcję energii możemy przewidzieć trajektorie zmian sygnałów w całej sieci.

Sieć Hopfielda w rozwiązywaniu problemu kombinatorycznego



Sieć Hopfielda w rozwiązywaniu problemu kombinatorycznego

Kluczem do sukcesu przy stosowaniu sieci neuronowej w problemie TSP jest odnalezienie odpowiedniej reprezentacji danych. W opisanym przez Tanka rozwiązaniu problemu każde miasto reprezentowane jest za pomocą wiersza zawierającego n neuronów. W takim wierszu **dokładnie jeden** neuron powinien przyjmować wartość „1", a **wszystkie pozostałe** mają sygnały wyjściowe odpowiadające wartości „0". Pozycja (od 1 do n), na której występuje neuron sygnalizujący „1" odpowiada kolejności, w jakiej to właśnie miasto ma być odwiedzone przez wędrownego sprzedawcę. Tak więc jedynka na pierwszej pozycji oznacza miasto, które komiwojażer powinien odwiedzić jako pierwsze, jedynka na drugiej pozycji sygnalizuje drugie w kolejności odwiedzane miasto itd. Oczywiście z opisu tego wynika, że liczba wierszy musi odpowiadać liczbie rozważanych miast (czyli musi wynosić n), zatem łączna liczba potrzebnych neuronów wynosi n^2.

Opisując funkcję „energii" minimalizowanej przez rozważaną sieć trzeba brać pod uwagę cztery jej składniki:

$$E_1 = A/2 \sum_x \sum_i \sum_{i \neq j} y_{xi}\, y_{xj}$$

$$E_2 = B/2 \sum_i \sum_x \sum_{z \neq x} y_{xi}\, y_{zj}$$

$$E_3 = C/2 \left[\left(\sum_x \sum_i y_{xi} \right) - n \right]^2$$

$$E_4 = D/2 \sum_x \sum_{z \neq x} \sum_i d_{xz}\, y_{xi}\, (y_{z,i+1} + y_{z,i-1})$$

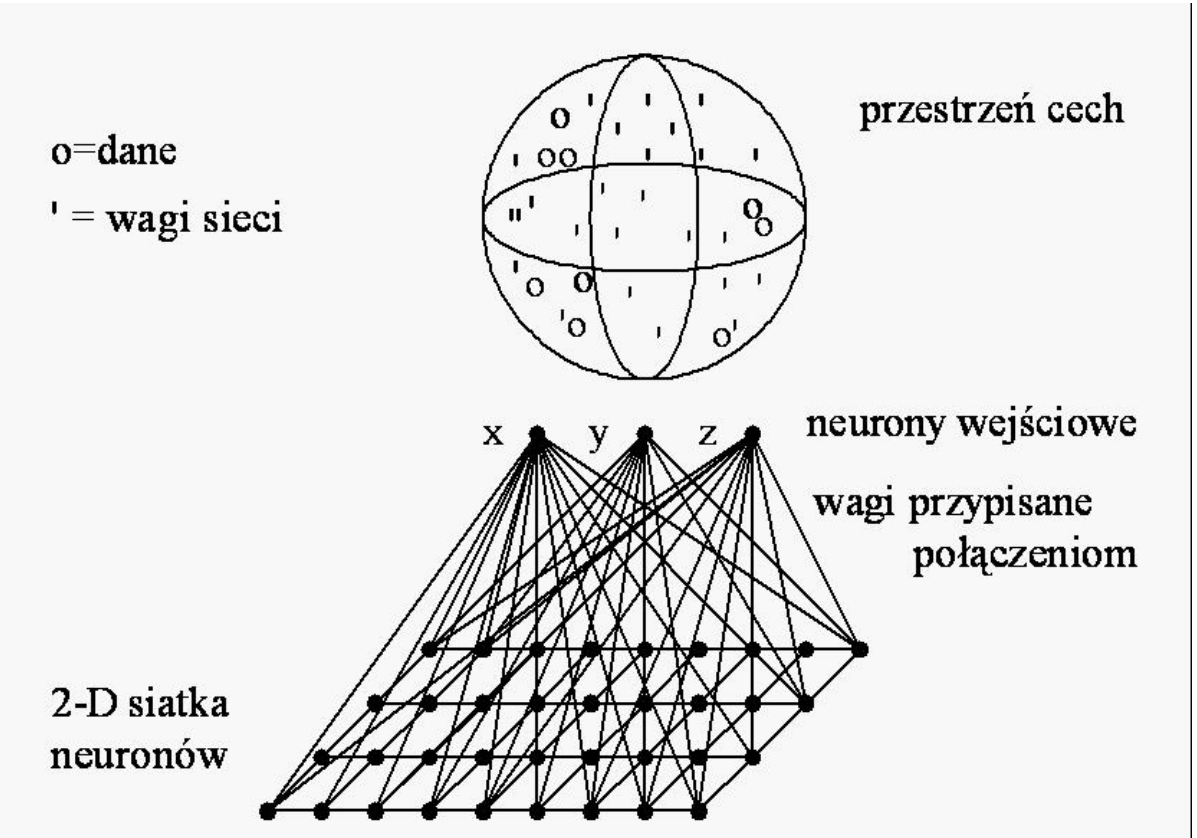
Outline

Modele samoorganizacji

- SOM (Self-Organized Maps) - samorganizująca się mapa.
- Mapy topograficzne powstaje przez połączenia lokalne: neuron silnie pobudzany przez pobliskie połączenie, słabo przez odległe, hamowany przez neurony pośrednie.
- Historia: von der Malsburg i Willshaw (1976), uczenie konkurencyjne, mechanizm Hebbowski, wzbudzenie typu „Meksykańskiego kapelusza", model układu wzrokowego. Amari (1980) - model ciągłej tkanki neuronowej. Kohonen (1981) - uproszczenie, bez hamowania; dwie fazy - konkurencja i kooperacja.

Algorytm SOM

- Neurony są umieszczone (ale nie połączone ze sobą) na siatce 1,2 lub 3-wymiarową;
- Każdy neuron ma N wag. Neuron i-ty ma wagi $\vec{W}_i(t) = (w_{i1}, ..., w_{iN})$ (t - czas dyskretny);
- Wektor wejściowy $X = (x_1, ..., x_N)$



Algorytm:

1. Inicjalizacja: przypadkowe $W_i(0)$ dla $i = 1 \ldots K$. Wyznacz otoczenie $Os(r_c, \sigma(t))$ neuronu r_c i o promieniu $\sigma(t)$.
2. Dla każdego $i = 1 \cdots K$ oblicz odległości

$$d(X, W_i) = ||X - W_i|| = \sqrt{\sum_j (x_j - w_{ij})^2}$$

3. Znajdź neuron-zwycięzcę: $c = \text{argmin}_i ||X - W_i||$ z wagami W_c najbardziej podobnymi do X.
4. Zmień wagi wszystkich neuronów r_i w sąsiedztwie $Os(r_c, \sigma(t))$

$$W_i(t+1) = W_i(t) + h(r_i, r_c)(X - W_i(t))$$

gdzie $h(r_i, r_c) = h_0(t) \cdot e^{-||r - r_c||^2 / \sigma_c^2(t)}$

5. Powoli zmniejszaj siłę $h_0(t)$ i promień $\sigma(t)$.
6. Iteruj aż ustaną zmiany.

Efekt: podział (tesselacja) na wieloboki Voronoia.

Własności SOM

- Brak dowodu o zbieżności lub punktach stacjonarnych dla SOM:
- Wyniki analityczne tylko w 1D dla ciągłego czasu, proces Markova: wartości wag wzdłuż prostej porządkują się. Powolna zbieżność: $10^4 - 10^6$ iteracji.
- Sąsiednie neurony kodują sąsiednie obszary, ale niekoniecznie odwrotnie Skręcone konfiguracje przy zbyt szybkiej redukcji sąsiedztwa. Złożoność O(KNn) dla K neuronów i n danych N-wymiarowych: konieczne porównanie wszystkich odległości; niezbyt duże mapy.
- Na komputerach wieloprocesorowych szukanie min z K będzie powolne.
- SOM działa jak metoda klasteryzacji k-średnich jeśli $\sigma = 0$.

Modyfikacje SOM

- Próba wprowadzenia funkcji błędu (Luttrell; Heskes i Kappen).
- Błąd lokalny neuronu i jest sumą po wszystkich neuronach:

$$E_i(X;t) = \frac{1}{2}\sum_j h(|r_i - r_j|, t) \cdot ||X - W_i(t)||^2$$

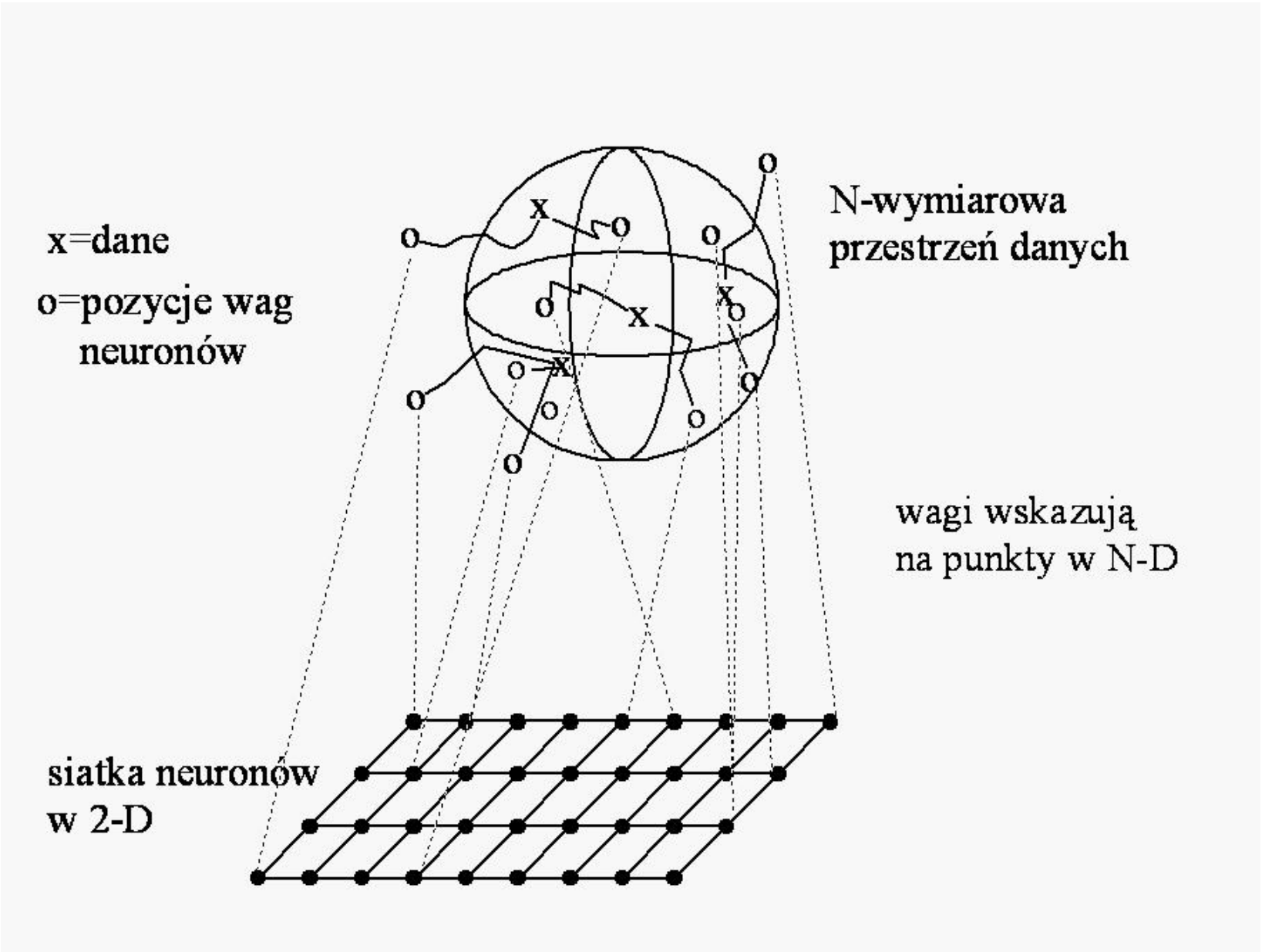
gdzie

$$h(|r_i - r_j|, t) = \exp -\frac{|r_i - r_j|^2}{2\sigma^2(t)}; \quad \sigma(t) = \sigma_0 e^{-2\sigma_0 t/t_{\max}}$$

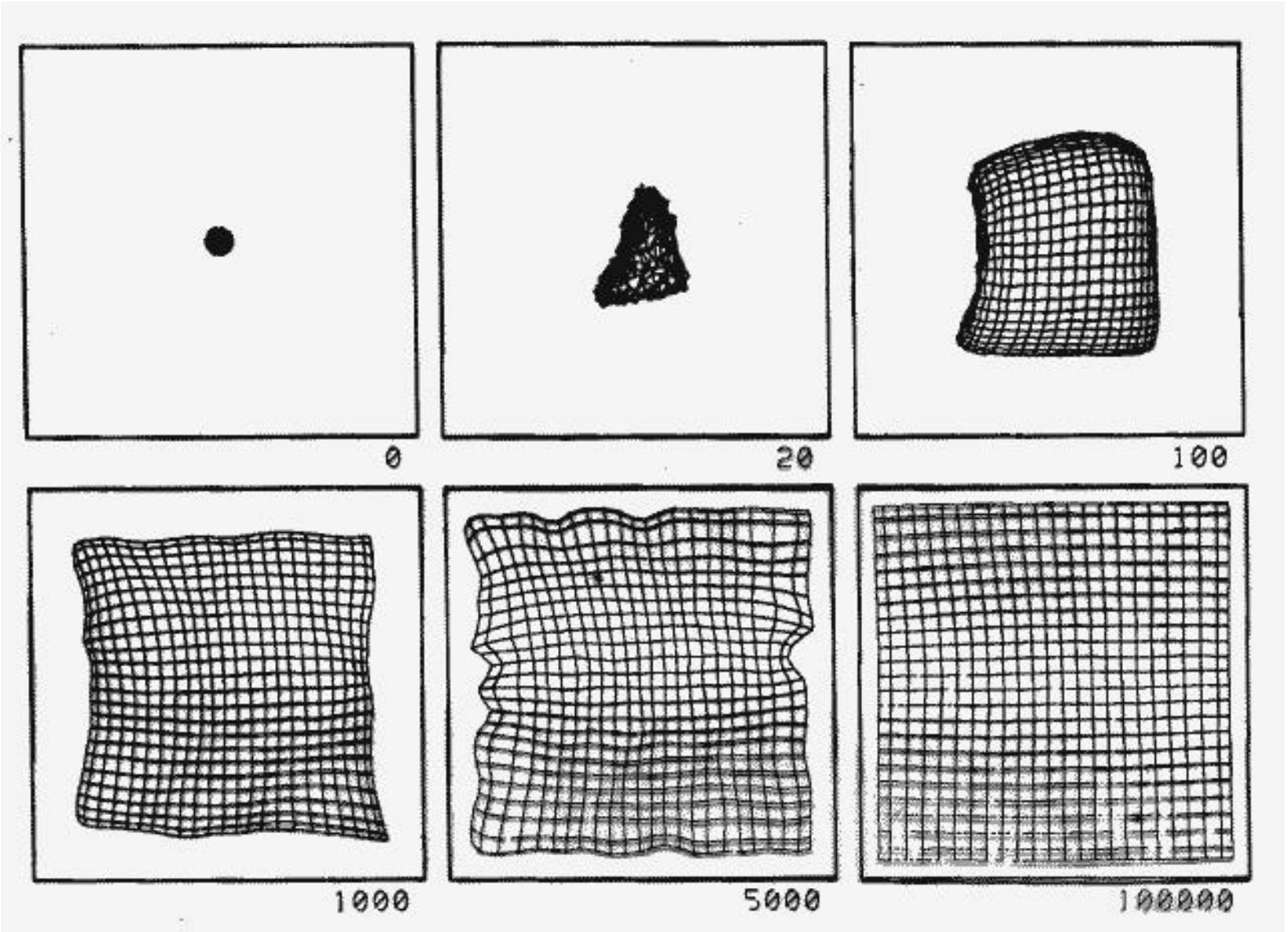
- Neuron-zwycięzca ma najmniejszy błąd lokalny:

$$c = \text{argmin}_i E_i(X;t)$$

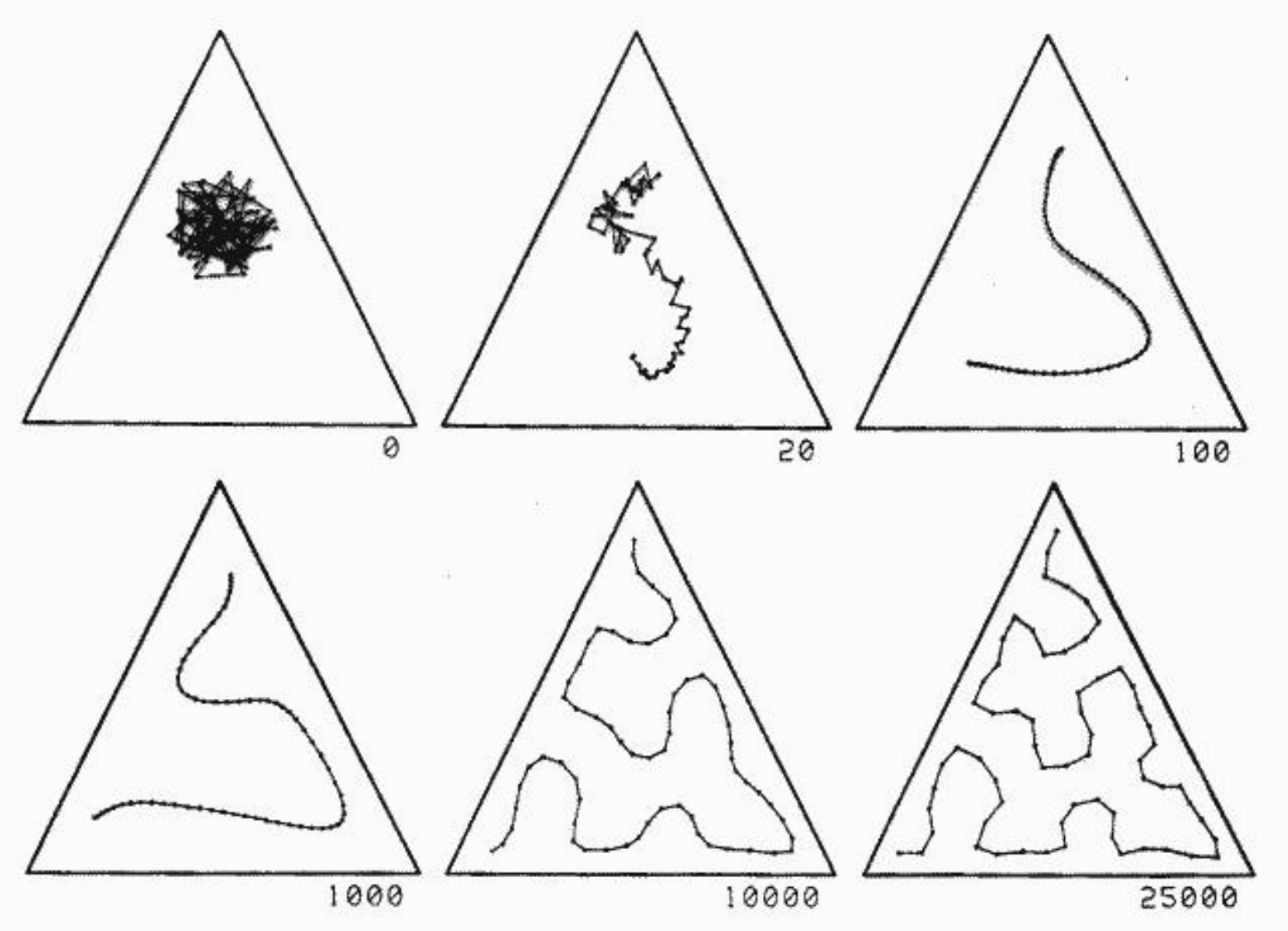
Uczenie sieci 2D



Uczenie kwadratu w sieci 2D

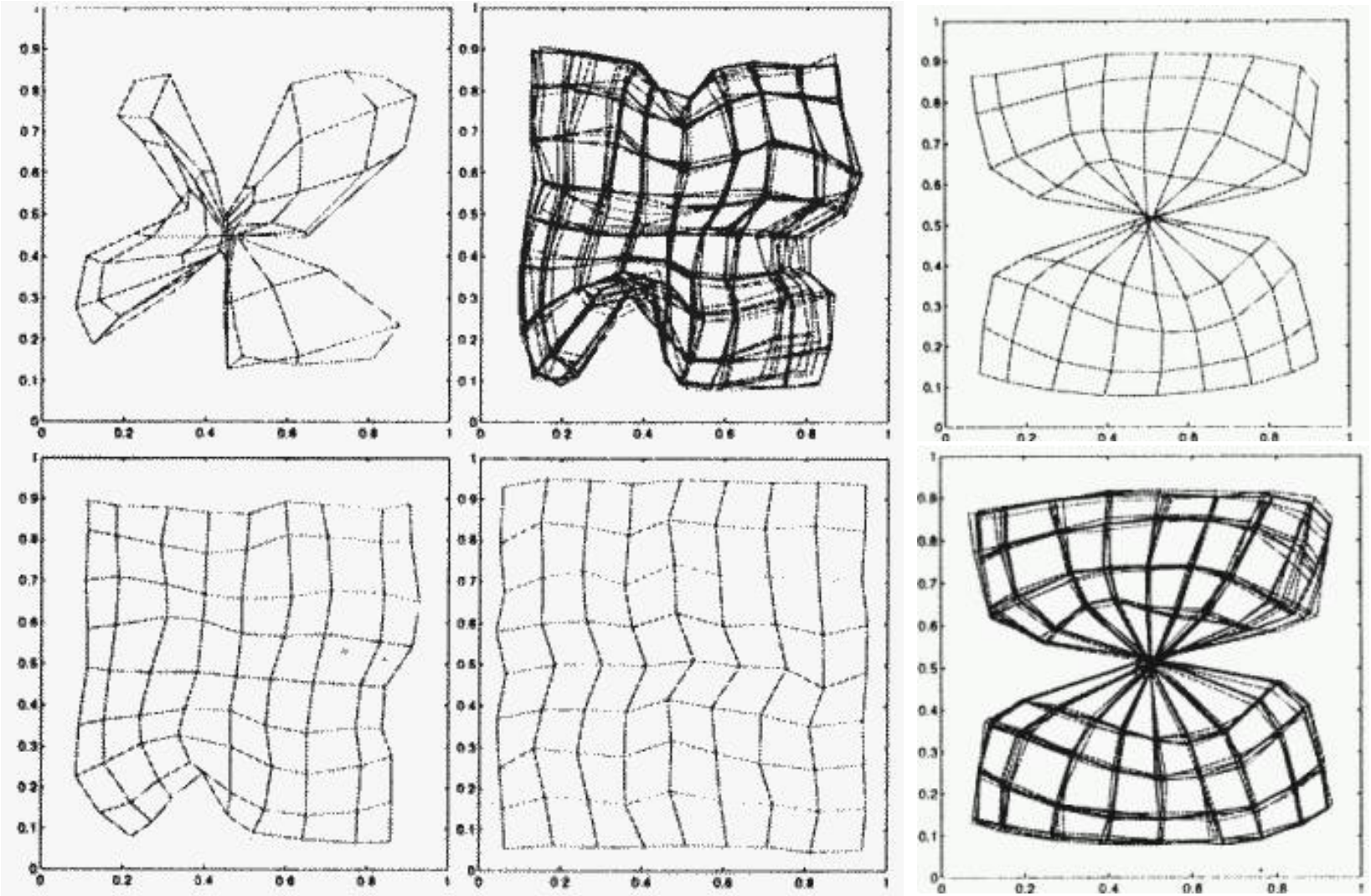


Uczenie trojkąta w sieci 1D



Tworzy się fraktalne krzywe Peano.

Zniekształcenie



Początkowe zniekształcenia mogą zniknąć lub pozostać.

WEBSOM websom.hut.fi/websom/