

## Organizační dynamika:

- ▶ Cyklická síť se symetrickými spoji (tj. libovolný neorientovaný graf)
- ▶ Množinu všech neuronů značíme  $N$
- ▶ označme  $\xi_j$  vnitřní potenciál a  $y_j$  výstup (stav) neuronu  $j$
- ▶ stav stroje:  $\vec{y} \in \{-1, 1\}^{|N|}$ .
- ▶ označme  $w_{ji}$  *reálnou* váhu spoje od neuronu  $i$  k neuronu  $j$ .
- ▶ žádný neuron nemá bias a předpokládáme  $w_{jj} = 0$  pro  $j \in N$ .

# Boltzmannův stroj - opakování

**Aktivní dynamika:** Stavby neuronů jsou iniciálně nastaveny na hodnoty z množiny  $\{-1, 1\}$ , tj.  $y_j^{(0)} \in \{-1, 1\}$  pro  $j \in N$ .

V  $t$ -tém kroku aktualizujeme náhodně vybraný neuron  $j \in N$  takto: nejprve vypočteme vnitřní potenciál

$$\xi_j^{(t-1)} = \sum_{i \in j_{\leftarrow}}^n w_{ji} y_i^{(t-1)}$$

a poté náhodně zvolíme hodnotu  $y_j^{(t)} \in \{-1, 1\}$  tak, že  $\mathbf{P}[y_j^{(t)} = 1] = \sigma(\xi_j^{(t-1)})$  kde

$$\sigma(\xi) = \frac{1}{1 + e^{-2\xi/T(t)}}$$

Parametr  $T(t)$  se nazývá **teplota** v čase  $t$ .

## Boltzmannův stroj - opakování

- ▶ Velmi vysoká teplota  $T(t)$  znamená, že  $\mathbf{P}[y_j^{(t)} = 1] \approx \frac{1}{2}$  a stroj se chová téměř náhodně.
- ▶ Velmi nízká teplota  $T(t)$  znamená, že buď  $\mathbf{P}[y_j^{(t)} = 1] \approx 1$  nebo  $\mathbf{P}[y_j^{(t)} = 1] \approx 0$  v závislosti na tom, jestli  $\xi_j^{(t)} > 0$  nebo  $\xi_j^{(t)} < 0$ . Potom se stroj chová téměř deterministicky (tj. jako Hopfieldova síť).

# Boltzmannův stroj - reprezentace rozložení

**Cíl:** Chceme sestrojít síť, která bude reprezentovat dané pravděpodobnostní rozložení na množině vektorů  $\{-1, 1\}^n$ .

**Velmi hrubá a nepřesná idea:** Boltzmannův stroj mění náhodně svůj stav z množiny  $\{-1, 1\}^n$ .

Když necháme B. stroj běžet dost dlouho s fixní teplotou, potom budou frekvence návštěv jednotlivých stavů nezávislé na iniciálním stavu.

Tyto frekvence budeme považovat za pravděpodobnostní rozložení na  $\{-1, 1\}^n$  reprezentované B. strojem.

V adaptivním režimu bude zadáno nějaké rozložení na stavech a cílem bude nalézt konfiguraci takovou, že rozložení reprezentované strojem bude odpovídat zadanému rozložení.

# Rovnovážný stav

Fixujeme teplotu  $T$  (tj.  $T(t) = T$  pro  $t = 1, 2, \dots$ ).

Boltzmannův stroj se po jisté době dostane do tzv. *termální rovnováhy*. Tj. existuje čas  $t^*$  takový, že pro libovolný stav stroje  $\gamma^* \in \{-1, 1\}^{|N|}$  platí

$$\mathbf{P}[\vec{y}^{(t^*)} = \gamma^*] \approx p_N(\gamma^*)$$

Zde  $p_N(\gamma^*) = \frac{1}{Z} e^{-E(\gamma^*)/T}$  kde

$$Z = \sum_{\gamma \in \{-1, 1\}^{|N|}} e^{-E(\gamma)/T}$$

tj. Boltzmannovo rozložení

**Pozn.:** Teorie Markovových řetězců říká, že  $\mathbf{P}[\vec{y}^{(t^*)} = \gamma^*]$  je také dlouhodobá frekvence návštěv stavu  $\gamma^*$ .

Toto platí *bez ohledu na iniciální nastavení neuronů!* Síť tedy reprezentuje rozložení  $p_N$ .

**Problém:** Tak jak jsme si jej definovali má Boltzmannův stroj omezenou schopnost reprezentovat daná rozložení.

Proto množinu neuronů  $N$  disjunktně rozdělíme na

- ▶ množinu **viditelných** neuronů  $V$
- ▶ množinu **skrytých** neuronů  $S$ .

Pro daný stav viditelných neuronů  $\alpha \in \{-1, 1\}^{|V|}$  označme

$$p_V(\alpha) = \sum_{\beta \in \{-1, 1\}^{|S|}} p_N(\alpha, \beta)$$

pravděpodobnost stavu viditelných neuronů  $\alpha$  v termálním ekvilibriu bez ohledu na stav skrytých neuronů.

Cílem bude adaptovat síť podle daného rozložení na  $\{-1, 1\}^{|V|}$ .

## Adaptivní dynamika:

Nechť  $p_d$  je pravděpodobnostní rozložení na množině stavů viditelných neuronů, tj. na  $\{-1, 1\}^{|V|}$ .

Cílem je nalézt konfiguraci sítě  $W$  takovou, že  $p_V$  odpovídá  $p_d$ .

Vhodnou mírou rozdílu mezi rozděleními  $p_V$  a  $p_d$  je relativní entropie zvážená pravděpodobnostmi vzorů (tzv. Kullback Leibler distance)

$$\mathcal{E}(\vec{w}) = \sum_{\alpha \in \{-1, 1\}^{|V|}} p_d(\alpha) \ln \frac{p_d(\alpha)}{p_V(\alpha)}$$

# Boltzmannův stroj - učení

$\mathcal{E}(\vec{w})$  budeme minimalizovat pomocí gradientního sestupu, tj. budeme počítat poslounost vektorů vah  $\vec{w}^{(0)}, \vec{w}^{(1)}, \dots$

- ▶ váhy v  $\vec{w}^{(0)}$  jsou inicializovány náhodně blízko 0
- ▶ v  $t$ -tém kroku (zde  $t = 1, 2, \dots$ ) je  $\vec{w}^{(t)}$  vypočteno takto:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)}$$

kde

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial \mathcal{E}}{\partial w_{ji}}(\vec{w}^{(t-1)})$$

je změna váhy  $w_{ji}$  v  $t$ -tém kroku a  $0 < \varepsilon(t) \leq 1$  je rychlost učení v  $t$ -tém kroku.

Zbývá spočítat (odhadnout)  $\frac{\partial \mathcal{E}}{\partial w_{ji}}(\vec{w})$ .



# Boltzmannův stroj - učení

Formálním derivováním funkce  $\mathcal{E}$  lze ukázat, že

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = -\frac{1}{T} \left( \langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{fixed} - \langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{free} \right)$$

- ▶  $\langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{fixed}$  je průměrná hodnota  $y_j^{(t^*)} y_i^{(t^*)}$  v termální rovnováze za předpokladu, že hodnoty viditelných neuronů jsou fixovány na počátku výpočtu dle rozložení  $p_d$ .
- ▶  $\langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{free}$  je průměrná hodnota  $y_j^{(t^*)} y_i^{(t^*)}$  v termální rovnováze bez fixace viditelných neuronů.

Celkově

$$\begin{aligned} \Delta w_{ji}^{(t)} &= -\varepsilon(t) \cdot \frac{\partial \mathcal{E}}{\partial w_{ji}} (\vec{w}^{(t-1)}) \\ &= \frac{\varepsilon(t)}{T} \left( \langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{fixed} - \langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{free} \right) \end{aligned}$$

# Boltzmannův stroj - učení

Pro výpočet  $\langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{fixed}$  proved' následující:

▶ Polož  $\mathcal{Y} := 0$  a proved' následující akce  $q$  krát:

1. fixuj náhodně hodnoty viditelných neuronů dle rozložení  $p_d$  (tj. v průběhu následujících kroků je neaktualizuj)

2. simuluj stroj po  $t^*$  kroků

[ Lepší způsob: simuluj stroj tak dlouho, dokud nedosáhne termální rovnováhy při nízké teplotě (zde je možné použít simulované žíhání, tedy pomalé snižování teploty). ]

3. přičti aktuální hodnotu  $y_j y_i$  k proměnné  $\mathcal{Y}$ .

▶  $\mathcal{Y}/q$  bude dobrým odhadem  $\langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{fixed}$  za předpokladu, že  $q$  je dostatečně velké číslo.

$\langle y_j^{(t^*)} y_i^{(t^*)} \rangle_{free}$  se odhadne podobně, pouze se nefixují viditelné neurony (tj. v kroku 1. se zvolí libovolný startovní stav a v následném výpočtu se mohou aktualizovat všechny neurony).

# Boltzmannův stroj - učení

Pro upřesnění analytická verze:

$$\begin{aligned} \langle y_i^{(t^*)} y_j^{(t^*)} \rangle_{fixed} &= \\ &= \sum_{\alpha \in \{-1,1\}^{|V|}} p_d(\alpha) \sum_{\beta \in \{-1,1\}^{|S|}} \frac{p_N(\alpha, \beta)}{p_V(\alpha)} y_j^{\alpha\beta} y_i^{\alpha\beta} \end{aligned}$$

kde  $y_j^{\alpha\beta}$  je výstup neuronu  $j$  ve stavu  $(\alpha, \beta)$ .

$$\langle y_i^{(t^*)} y_j^{(t^*)} \rangle_{free} = \sum_{\gamma \in \{-1,1\}^{|M|}} p_N(\gamma) y_j^\gamma y_i^\gamma$$

# Vektorová kvantizace

Obecná formulace problému:

- ▶ Dána hustota pravděpodobnosti  $p(\vec{x})$  na vstupních vektorech  $\vec{x} \in \mathbb{R}^n$ .  
Tj. předpokládáme, že „něco“ náhodně generuje vstupní vektory podle rozložení s hustotou  $p(\vec{x})$ .
- ▶ Cílem je aproximovat hustotu  $p(\vec{x})$  pomocí konečně mnoha **reprezentantů**  $\vec{w}_i \in \mathbb{R}^n$  kde  $i = 1, \dots, h$ .  
Velmi zhruba: v oblasti s větší hustotou by mělo být více reprezentantů, v oblastech s menší hustotou méně.
- ▶ Formálněji: danému vstupnímu vektoru  $\vec{x}$  přiřadíme reprezentanta  $\vec{w}_c$ , který je mu nejbližší:

$$c = \arg \min_{i=1, \dots, h} \{ \|\vec{x} - \vec{w}_i\| \}$$

a poté minimalizujeme chybu

$$E = \int \|\vec{x} - \vec{w}_c\|^2 p(\vec{x}) d\vec{x}$$

Pozor! Zde  $c$  závisí na  $\vec{x}$ .

# Vektorová kvantizace

V praxi často máme často k dispozici tréninkovou množinu

$$\mathcal{T} = \{\vec{x}_j \in \mathbb{R}^n \mid j = 1, \dots, \ell\}$$

z níž jsou vzory rovnoměrně vytahovány. Chyba potom odpovídá

$$E = \frac{1}{\ell} \sum_{j=1}^{\ell} \|\vec{x}_j - \vec{w}_c\|^2$$

Pokud byla množina  $\mathcal{T}$  volena náhodně (dle rozložení  $p(\vec{x})$ ) a  $\ell$  je dost velké, pak

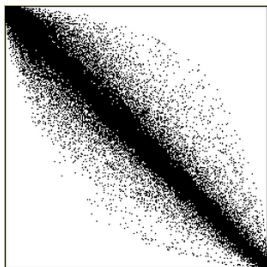
$$\frac{1}{\ell} \sum_{j=1}^{\ell} \|\vec{x}_j - \vec{w}_c\|^2 \approx \int \|\vec{x} - \vec{w}_c\|^2 p(\vec{x}) d\vec{x}$$

## Příklad - komprese obrázků

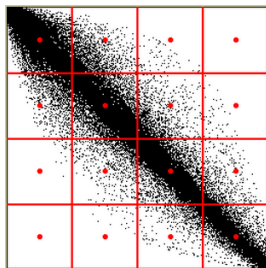


- ▶ každý pixel má 256 stupňů šedi.
- ▶ každá dvojice sousedních pixelů je dvourozměrným vektorem z  $\{0, \dots, 255\} \times \{0, \dots, 255\}$
- ▶ komprese najde malou množinu reprezentantů, kteří budou kódovat stupně šedi *dvojic* pixelů (pro danou dvojici vezmeme nejbližšího reprezentanta)
- ▶ obrázek se potom zakóduje jednoduchým průchodem při němž se dvojice pixelů nahradí reprezentanty

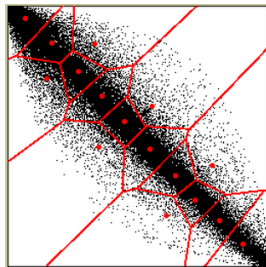
# Příklad - komprese obrázků



distribuce dvojic



naivní kvantizace



chytřejší kvantizace

# Lloydův algoritmus

Přepodkládáme konečnou tréninkovou množinu:

$$\{\vec{x}_j \mid \vec{x}_j \in \mathbb{R}^n \mid j = 1, \dots, \ell\}$$

Algoritmus postupně posouvá reprezentanty k tréninkovým vzorům.

V kroku  $t$  vypočte  $\vec{w}_1^{(t)}, \dots, \vec{w}_h^{(t)}$  takto:

- ▶ pro každé  $k = 1, \dots, h$  spočítej množinu  $\mathcal{T}_k$  vektorů z  $\mathcal{T}$ , které jsou nejbližší  $\vec{w}_k^{(t-1)}$ :

$$\mathcal{T}_k = \left\{ \vec{x}_j \in \mathcal{T} \mid k = \arg \min_{i=1, \dots, h} \left\{ \left\| \vec{x}_j - \vec{w}_i^{(t-1)} \right\| \right\} \right\}$$

- ▶ spočítej  $\vec{w}_k^{(t)}$  jako těžiště množiny  $\mathcal{T}_k$ :

$$\vec{w}_k^{(t)} = \frac{1}{|\mathcal{T}_k|} \sum_{\vec{x} \in \mathcal{T}_k} \vec{x}$$

Výpočet ukončíme např. ve chvíli, kdy je chyba  $E$  pro aktuální reprezentanty dostatečně malá.



Nevýhoda Lloydova algoritmu: není online! (mimo jiné ...)

Následující Kohonenův algoritmus je online (připouští obě verze vstupů: náhodně generované i dané tréninkovou množinou).

V kroku  $t$  po předložení vstupu  $\vec{x}_t$  vypočteme  $\vec{w}_k^{(t)}$  takto:

**Jestliže** je  $\vec{w}_k^{(t-1)}$  nejbližší  $\vec{x}_t$ , tj.  $k = \arg \min_i \left\| \vec{x}_t - \vec{w}_i^{(t-1)} \right\|$  **pak**

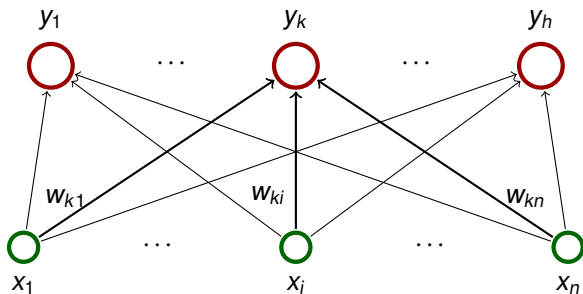
$$\vec{w}_k^{(t)} = \vec{w}_k^{(t-1)} + \theta \cdot (\vec{x}_t - \vec{w}_k^{(t-1)})$$

**jinak**  $\vec{w}_k^{(t)} = \vec{w}_k^{(t-1)}$

Parametr  $0 < \theta \leq 1$  určuje míru posunu reprezentanta ve směru ke vstupnímu vzoru. Na začátku učení se obvykle volí blízko 1 a postupně se zmenšuje.

Tento algoritmus lze snadno formulovat v jazyce neuronových sítí ...

## Organizační dynamika: Jednovrstvá síť



**Aktivní dynamika:** Pro vstup  $\vec{x} \in \mathbb{R}^n$  a  $k = 1, \dots, h$ :

$$y_k = \begin{cases} 1 & k = \arg \min_{i=1, \dots, h} \|\vec{x} - \vec{w}_i\| \\ 0 & \text{jinak} \end{cases}$$

## Adaptivní dynamika (online algoritmus, Kohonenovo učení):

V kroku  $t$  po předložení vstupu  $\vec{x}_t$  adaptujeme každé  $\vec{w}_k$  takto:

**Jestliže**  $k = \arg \min_{i=1, \dots, h} \left\| \vec{x}_t - \vec{w}_i^{(t-1)} \right\|$  **pak**

$$\vec{w}_k^{(t)} = \vec{w}_k^{(t-1)} + \theta(t) \cdot (\vec{x}_t - \vec{w}_k^{(t-1)})$$

**jinak**  $\vec{w}_k^{(t)} = \vec{w}_k^{(t-1)}$

Parametr  $0 < \theta \leq 1$  určuje míru změny vah ve směru ke vstupnímu vzoru. Na začátku učení se obvykle volí blízko 1 a postupně se zmenšuje.

- ▶ Funguje dobře pokud většina vstupů zhruba rovnoměrně pokrývá jednu konvexní oblast.
- ▶ V případě dvou a více oddělených shluků nemusí hustota reprezentantů odpovídat hustotě rozložení:
  - ▶ Příklad: dvě oddělené oblasti se stejnou hustotou.
  - ▶ Předpoklad, že vzory jsou iniciálně rozmístěny v jedné oblasti.
  - ▶ Druhá potom přitáhne jednoho reprezentanta, který poté vždy vyhraje soutěž.
  - ▶ Výsledek: jedna oblast bude reprezentována jedním reprezentantem, druhá zbytkem reprezentantů (hustota reprezentantů tedy neodpovídá hustotě pravděpodobnosti).

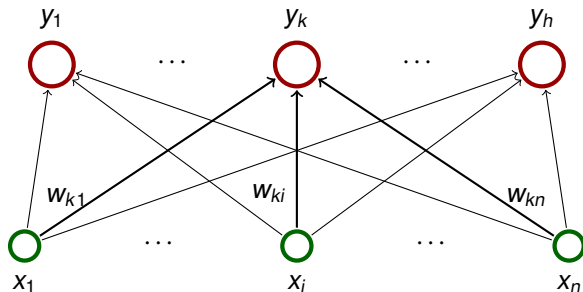
# Částečná zlepšení Kohonenova učení

Tyto nedostatky lze částečně odstranit např.

- ▶ přidáním šumu k tréninkovým vzorům - obvykle se přidávají náhodně generované vektory k tréninkovým vzorům, na počátku učení náhodné vektory převažují a postupně je jejich počet snižován
- ▶ radiální růst
  - ▶ začínáme s váhovými vektory blízko  $\vec{0}$
  - ▶ všechny tréninkové vzory násobíme faktorem  $\beta$ , který je na počátku blízko 0 a postupně roste (váhy se musí postupně vzdalovat od  $\vec{0}$ )
- ▶ lokální paměť: Cílem je, aby pravděpodobnost vítězství každého z  $h$  neuronů (reprezentantů) byla zhruba  $1/h$ .

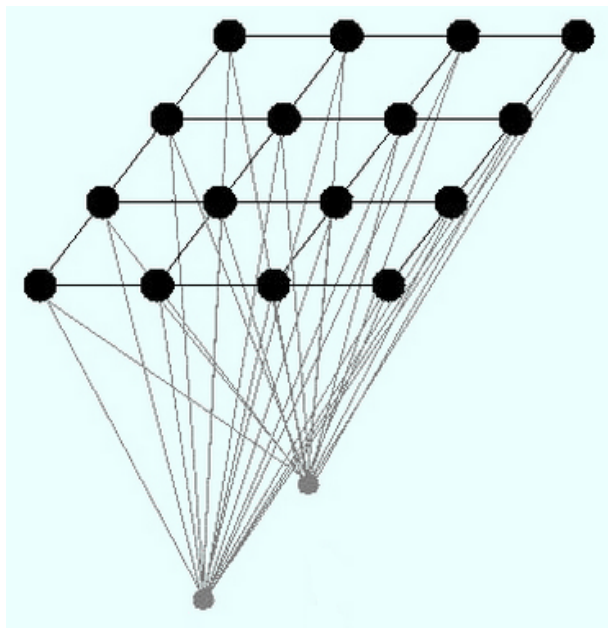
Každý neuron si udržuje přehled o tom kolikrát vyhrál v soutěži. Pokud je četnost výher příliš vysoká, na nějakou dobu vypadne ze soutěže.

## Organizační dynamika: Jednovrstvá síť

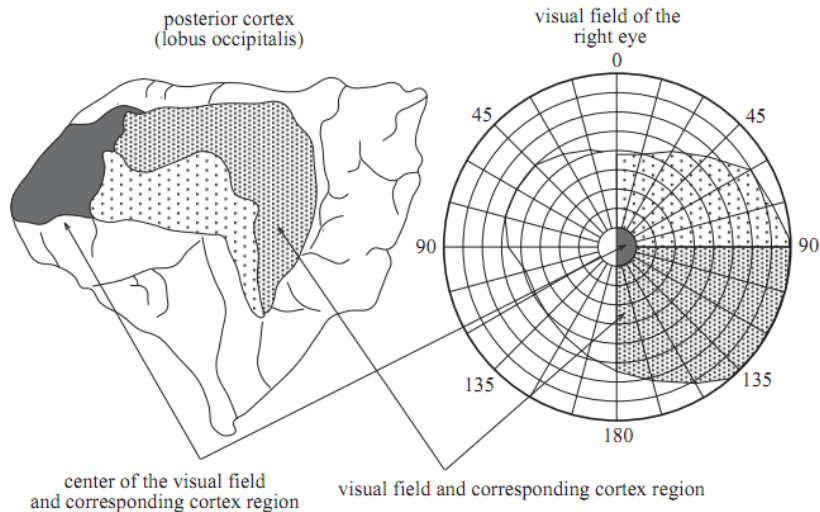


- ▶ Mezi neurony je navíc zavedena **topologická struktura** (tj. neurony tvoří uzly neorientovaného grafu).
- ▶ V drtivé většině případů je tato struktura buď jednorozměrná řada jednotek nebo dvojrozměrná mřížka.

## Kohonenova mapa - ilustrace



# Kohonenova mapa - bio odbočka



**Fig. 15.2.** Mapping of the visual field on the cortex

Zdroj obrázku: Neural Networks - A Systematic Introduction, Raul Rojas, Springer, 1996



**Aktivní dynamika:** Pro vstup  $\vec{x} \in \mathbb{R}^n$  a  $k = 1, \dots, h$ :

$$y_k = \begin{cases} 1 & k = \arg \min_{i=1, \dots, h} \|\vec{x} - \vec{w}_i\| \\ 0 & \text{jinak} \end{cases}$$

**Adaptivní dynamika:** V adaptivním režimu využijeme topologickou strukturu.

- ▶ Označme  $d(c, k)$  délku nejkratší cesty z neuronu  $c$  do neuronu  $k$  v topologické struktuře.
- ▶ Pro neuron  $c$  a dané  $s \in \mathbb{N}_0$  definujeme **okolí** neuronu  $c$  velikosti  $s$  takto:  $N_s(c) = \{k \mid d(c, k) \leq s\}$

V kroku  $t$  po předložení vstupu  $\vec{x}_t$  adaptujeme každé  $\vec{w}_k$  takto:

$$\vec{w}_k^{(t)} = \begin{cases} \vec{w}_k^{(t-1)} + \theta \cdot (\vec{x}_t - \vec{w}_k^{(t-1)}) & k \in N_s(c) \\ \vec{w}_k^{(t-1)} & \text{jinak} \end{cases}$$

kde  $c = \arg \min_{i=1, \dots, h} \|\vec{x}_t - \vec{w}_i^{(t-1)}\|$  a kde  $\theta \in \mathbb{R}$  a  $s \in \mathbb{N}_0$  jsou parametry, které se mohou v průběhu učení měnit.

## Obecnější formulace adaptivní dynamiky:

$$\vec{w}_k^{(t)} = \vec{w}_k^{(t-1)} + \Theta(c, k) \cdot (\vec{x} - \vec{w}_k^{(t-1)})$$

kde  $c = \arg \min_{i=1, \dots, h} \|\vec{x}_t - \vec{w}_i^{(t-1)}\|$ . Předchozí případ potom odpovídá

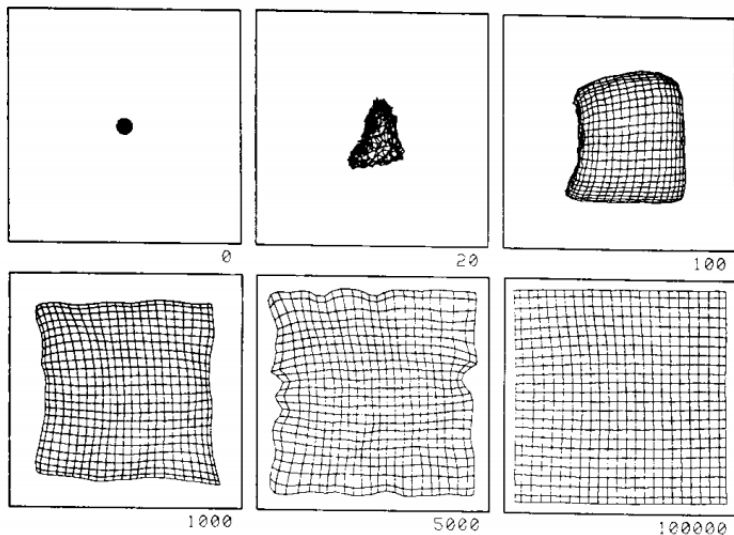
$$\Theta(c, k) = \begin{cases} \theta & k \in N_s(c) \\ 0 & \text{jinak} \end{cases}$$

Obvykle se používá plynulejší přechod mezi nenulovými a nulovými hodnotami, např.

$$\Theta(c, k) = \theta_0 \cdot \exp\left(\frac{-d(c, k)^2}{\sigma^2}\right)$$

kde  $\theta_0 \in \mathbb{R}$  určuje maximální míru změny vah a  $\sigma \in \mathbb{R}$  je šířka (oba parametry se mohou v průběhu měnit).

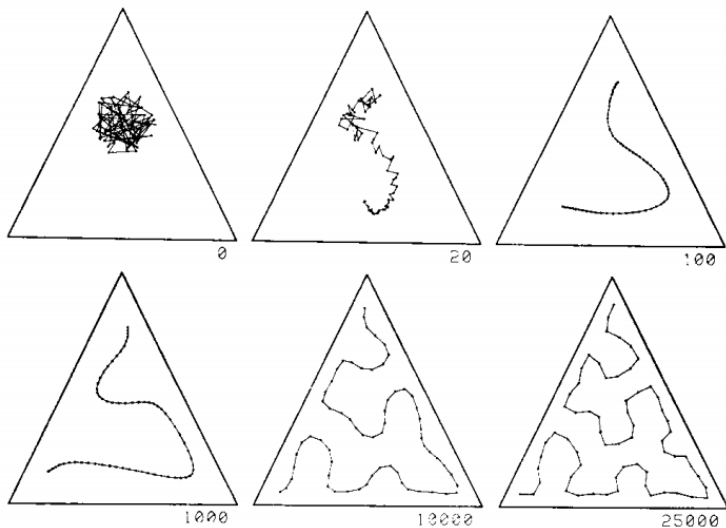
# Příklad 1



Vstupy jsou uniformně rozloženy ve čtverci.

Zdroj obrázku: Neural Networks - A Systematic Introduction, Raul Rojas, Springer, 1996

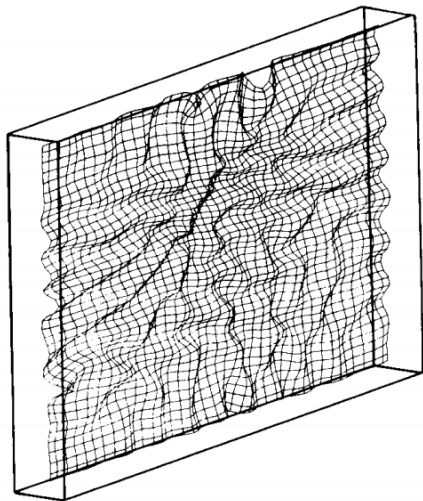
## Příklad 2



Vstupy jsou uniformně rozloženy v trojúhelníku.

Zdroj obrázku: Neural Networks - A Systematic Introduction, Raul Rojas, Springer, 1996

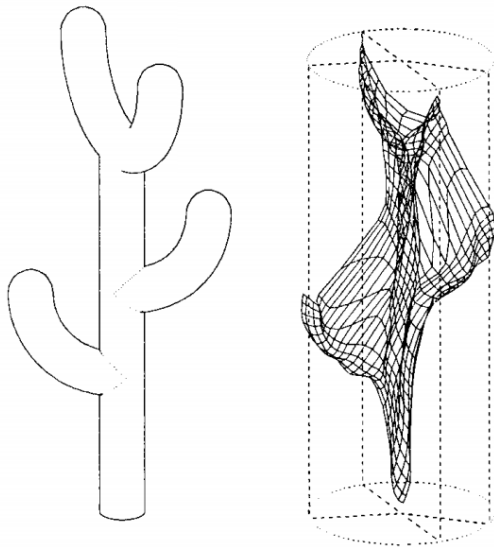
## Příklad 3



Vstupy jsou uniformně rozloženy v kvádru.

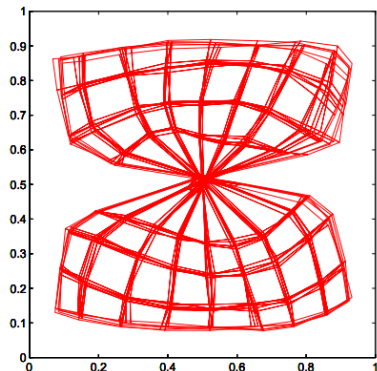
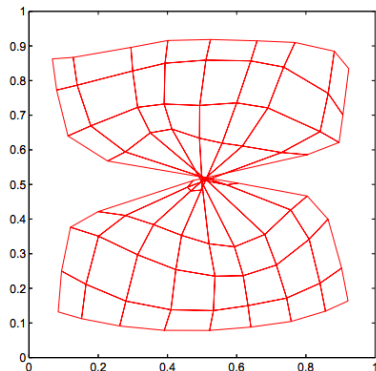
Zdroj obrázku: Neural Networks - A Systematic Introduction, Raul Rojas, Springer, 1996

## Příklad 4



Vstupy jsou uniformně rozloženy v kaktusu.

# Příklad - defekt



Překroucená síť - topologický defekt.

Zdroj obrázku: Neural Networks - A Systematic Introduction, Raul Rojas, Springer, 1996

Iniciální váhy (prý) nejsou příliš důležité, pokud jsou různé.

Učení se obvykle sestává ze dvou fází:

*hrubé učení:*

- ▶ obvykle cca 1000 kroků
- ▶ rychlost učení  $\theta$  by měla začít kolem 0.1 a postupně klesat k 0.01
- ▶ okolí každého neuronu (dané parametrem  $s$  nebo případně šířkou  $\sigma$ ) by mělo zpočátku obsahovat většinu neuronů a postupně se zmenšovat tak, aby ke konci této fáze obsahovalo nejvýše několik neuronů

*dolad'ování:*

- ▶ počet kroků je obvykle volen jako 500 násobek počtu neuronů v síti
- ▶  $\theta$  by se měla držet kolem 0.01 jinak hrozí topologický defekt (zmačkaná síť)
- ▶ okolí každého neuronů by mělo obsahovat jen pár dalších neuronů (nebo jen ten jeden)



# Kohonenova mapa - pohled teoretika

Předpokládáme náhodně generované vstupy s hustotou pravděpodobnosti  $p(\vec{x})$ . Platí  $\int p(\vec{x})d\vec{x} = 1$ .

Definujeme hustotu bodů  $m(\vec{x}) = \#(\vec{x})/d\vec{x}$  kde  $\#(\vec{x})$  je počet neuronů jejichž váhové vektory leží v malém okolí  $d\vec{x}$  bodu  $\vec{x}$  ve vstupním prostoru. Platí

$$\int m(\vec{x})d\vec{x} = h$$

kde  $h$  je počet neuronů (reprezentantů).

Pokud je dimenze mapy rovna jedné (tj. mapa je řetězcem neuronů) pak po natrénování

$$m(\vec{x}) \propto p^{2/3}(\vec{x})$$

# Kohonenova mapa - pohled teoretika

- ▶ Konvergenci k uspořádanému stavu se podařilo dokázat pouze pro jednorozměrnou topologii a různá rozložení vstupů (první výsledky byly pro uniformní rozložení) a fixní okolí velikosti 1 a konstantní rychlost učení

Lze nalézt velice jednoduché protipříklady pokud tyto podmínky nejsou splněny.

- ▶ Ve vícerozměrném případě žádné obecné výsledky neexistují, konvergence do uspořádaného stavu je ovlivněna mnoha faktory:
  - ▶ iniciální rozložení neuronů
  - ▶ velikost okolí
  - ▶ rychlost učení
- ▶ Jakou dimenzi topologie zvolit? Drtivá většina aplikací je založena na jedné nebo dvoudimenzionální mapě. Experimenty prokázaly, že ideální dimenze topologie odpovídá (Hausdorfově) dimenzi dat.

Přepodkládejme, že máme náhodně generované vzory tvaru  $(\vec{x}_t, d_t)$  kde  $\vec{x}_t \in \mathbb{R}^n$  je **vektor vlastností** a  $d_t \in \{C_1, \dots, C_q\}$  určuje jednu z  $q$  **tříd**.

Cílem je klasifikovat vstupy do tříd na základě znalosti vlastností, tj. každému  $\vec{x}_t$  chceme přiřadit třídu tak, abychom minimalizovali pravděpodobnost chyby.

Př.: Po pásu jede náhodně „naházené“ ovoce dvou druhů: jablka a meruňky. Námi sledovaná data budou  $(\vec{x}_t, d_t)$  kde

- ▶  $\vec{x}_t \in \mathbb{R}^2$ , první vlastnost je hmotnost, druhá je průměr
- ▶  $d_t$  je buď J nebo M v závislosti na tom, jestli je daný kus jablko nebo meruňka

Připouštíme možnost, že se najdou dvě jablka se stejnými mírami.

Cílem je třídít ovoce na základě hmotnosti a průměru.

# LVQ - klasifikace pomocí Kohonenovy mapy

Využijeme vektorovou kvantizaci (Kohonenovu mapu) takto:

1. Natrénujeme mapu na vektorech vlastností  $\vec{x}_t$  kde  $t = 1, \dots, \ell$ .
2. Jednotlivé neurony označíme třídami. Třidu  $v_c$  neuronu  $c$  nalezneme takto:

Pro každý neuron  $c$  a třídu  $C_i$  spočítáme četnost vzorů třídy  $C_i$ , které jsou reprezentovány neuronem  $c$ .

Toto lze provést jedním průchodem přes vzory

Neuronu  $c$  přiřadíme třídu s maximální četností.

3. Doladíme síť pomocí algoritmu LVQ (viz. dále)

Klasifikaci pomocí natrénované sítě provádíme takto: Na vektor vlastností  $\vec{x}$  aplikujeme natrénovanou síť v aktivním režimu. Právě jeden neuron, řekněme  $c$ , bude mít výstup 1. Potom vstup zařadíme do třídy  $v_c$ .

Postupně projdi tréninkové vzory. Pro vzor  $(\vec{x}_t, d_t)$  urči nejbližší neuron  $c$

$$c = \arg \min_{i=1, \dots, h} \|\vec{x}_t - \vec{w}_i\|$$

Potom uprav váhy neuronu  $c$  takto:

$$\vec{w}_c^{(t)} = \begin{cases} \vec{w}_c^{(t-1)} + \alpha(\vec{x}_t - \vec{w}_c^{(t-1)}) & d_t = v_c \\ \vec{w}_c^{(t-1)} - \alpha(\vec{x}_t - \vec{w}_c^{(t-1)}) & d_t \neq v_c \end{cases}$$

Parametr  $\alpha$  by měl být od počátku malý (cca 0.01 – 0.02) a postupně klesnout k 0.

Hranice mezi třídami vytvořená pomocí LVQ1 je poměrně dobrou aproximací *bayesovské rozhodovací hranice*.

Co to je??

# Bayesovský klasifikátor

Mějme pouze dvě třídy  $C_0$  a  $C_1$  (např. J a M).

Nechť  $P(C_i | \vec{x})$  je pravděpodobnost, že je vstup z třídy  $C_i$  za předpokladu, že má vlastnost  $\vec{x}$ .

(např.  $P(C_0 | (a, b))$  vyjadřuje pravděpodobnost, že ovoce s hmotností  $a$  a průměrem  $b$  je jablko)

Bayesovský klasifikátor přiřadí vstupu s vektorem vlastností  $\vec{x}$  třídu  $C_i$ , která splňuje  $P(C_i | \vec{x}) \geq P(C_{1-i} | \vec{x})$ .

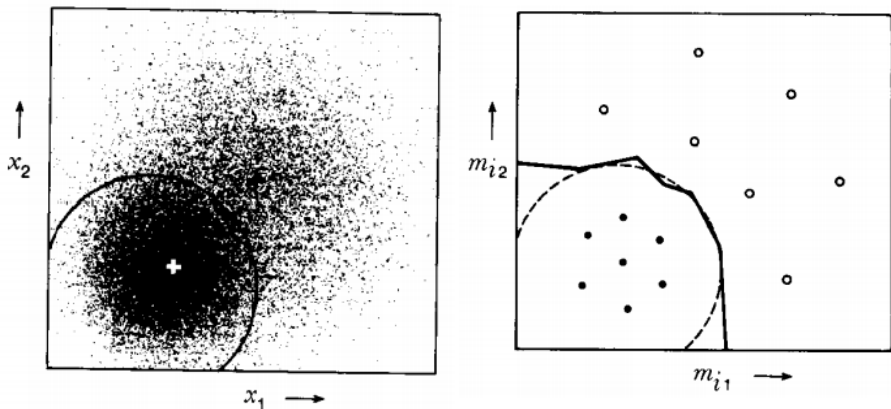
Označme  $R_0$  množinu všech  $\vec{x}$ , která splňují  $P(C_0 | \vec{x}) \geq P(C_1 | \vec{x})$  a  $R_1 = \mathbb{R}^n \setminus R_0$ .

Bayesovský klasifikátor minimalizuje pravděpodobnost chyby:

$$P(\vec{x} \in R_0 \wedge C_1) + P(\vec{x} \in R_1 \wedge C_0)$$

Bayesovská hranice je hranicí mezi množinami  $R_0$  a  $R_1$ .

# Bayesovská hranice a LVQ1 - příklad



Zdroj obrázku: The Self-Organizing Map, Teuvo Kohonen, IEEE, 1990

Aproximaci bayesovské hranice je možné zlepšit následujícím způsobem:

Pro vzor  $(\vec{x}_t, d_t)$  určíme dva nejbližší neurony, řekněme  $i$  a  $j$  s aktuálními váhami  $\vec{w}_i^{(t-1)}$  a  $\vec{w}_j^{(t-1)}$ .

Jestliže jsou splněny následující dvě podmínky

- ▶  $\vec{x}_t$  se nachází v **okně** okolo nadroviny umístěné ve středu spojnice  $\vec{w}_i^{(t-1)}$  a  $\vec{w}_j^{(t-1)}$  a
- ▶ jeden z neuronů  $i$  a  $j$  klasifikuje správně a druhý špatně

pak aktualizujeme váhy takto (bez újmy na obecnosti předpokládejme, že neuron  $i$  klasifikuje správně):

$$\vec{w}_i^{(t)} = \vec{w}_i^{(t-1)} + \alpha(\vec{x}_t - \vec{w}_i^{(t-1)})$$

$$\vec{w}_j^{(t)} = \vec{w}_j^{(t-1)} - \alpha(\vec{x}_t - \vec{w}_j^{(t-1)})$$



Okno je obvykle definováno takto: Řekneme, že  $\vec{x}_t$  se nachází v okně relativní šířky  $q$  mezi  $\vec{w}_i^{(t-1)}$  a  $\vec{w}_j^{(t-1)}$  pokud

$$\min \left\{ \frac{d_i}{d_j}, \frac{d_j}{d_i} \right\} > \frac{1-q}{1+q}$$

kde  $d_i = \left\| \vec{x}_t - \vec{w}_i^{(t-1)} \right\|$  a  $d_j = \left\| \vec{x}_t - \vec{w}_j^{(t-1)} \right\|$ .

Obvykle se volí  $q$  mezi 0.1 až 0.3.

LVQ2 zlepšuje pozici rozhodovací hranice tím, že ji posunuje k bayesovské hranici, ale jen do určitého počtu kroků. Později se neurony začínají vzdalovat (typicky se používá kolem 10 000 iterací).

LVQ3 se snaží odstranit defekt LVQ2 tím, že dobře klasifikující dvojici nejbližších neuronů posunuje směrem k vzoru (LVQ2 v takovém případě nedělá nic):

Nejprve „LVQ2 část“:

$$\vec{w}_i^{(t)} = \vec{w}_i^{(t-1)} + \alpha(\vec{x}_t - \vec{w}_i^{(t)})$$

$$\vec{w}_j^{(t)} = \vec{w}_j^{(t-1)} - \alpha(\vec{x}_t - \vec{w}_j^{(t)})$$

pokud  $i$  a  $j$  je pár výstupních neuronů nejbližše k  $\vec{x}_t$ ,  $v_i = d_t$ ,  $v_j \neq d_t$  a  $\vec{x}_t$  patří do okna mezi  $\vec{w}_i^{(t)}$  a  $\vec{w}_j^{(t)}$ .

Navíc:

$$\vec{w}_r^{(t)} = \vec{w}_r^{(t-1)} + \varepsilon\alpha(\vec{x}_t - \vec{w}_r^{(t)})$$

pokud  $r \in \{i, j\}$  a  $v_i = v_j = d_t$ .

$\varepsilon$  se doporučuje nastavit dle šířky okna mezi 0.1 až 0.5 a nechat konstantní.