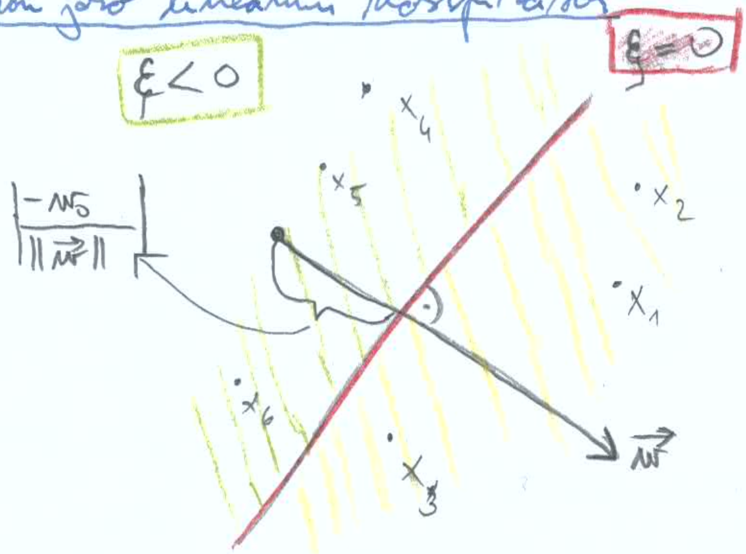
PV021 | Neuronové sítě – Důležitosti, důkazy a pomůcky

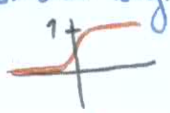
- **Neuron jako lineární klasifikátor**



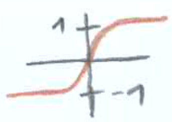
- přímka ($\xi = 0$) je definovaná vektorem $\vec{w}$ (váhy neuronu)
- x_1, x_2, x_3 (směr, kam ukazuje vektor $\vec{w}$) mají $\xi > 0$; a x_4, x_5, x_6 mají $\xi < 0$
- posunutí přímky (lineárního klasifikátoru) je definováno $\left|\frac{-w_0}{\|\vec{w}\|}\right|$, kde w_0 je bias

- **Aktivační funkce**
  - Unit step function: $\sigma(\xi) = \begin{cases} 1 & \xi \geq 0 \\ 0 & \xi < 0 \end{cases}$

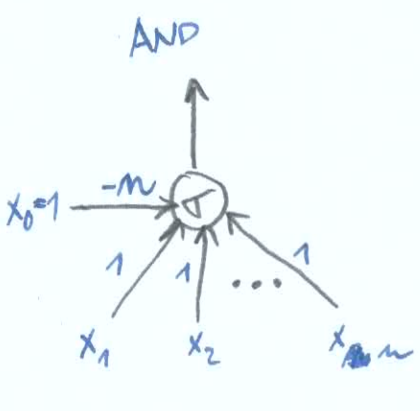
  - Logistická sigmoida: $\sigma(\xi) = \frac{1}{1 + e^{-\lambda \cdot \xi}}$; $\lambda \in \mathbb{R}$ parametr strmosti

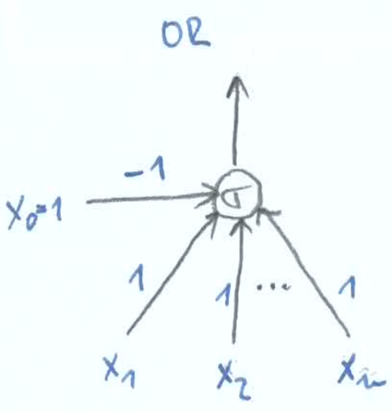


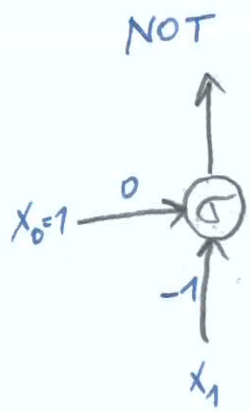
  - Hyperbolic tangens: $\sigma(\xi) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}}$



- **Logické funkce** (unit step function)







ADALINE

- Error function

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\vec{w}\cdot\vec{x}_k - d_k\right)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right)^2$$

$\Rightarrow$ cíl: najít $\vec{w}$ takové, které minimalizuje $E(\vec{w})$

- Gradient of the error function

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \dots, \frac{\partial E}{\partial w_n}(\vec{w})\right)$$

- vektor v prostoru vah, který míří ve směru nejstrmějšího svahu chybové fce.
- vektory $\vec{x}_k$ jsou fixované ← vstupy

$\Rightarrow$ $\nabla E(\vec{w}) = \vec{0} = (0, \dots, 0)$ znamená globální minimum chybové fce E

- $$\frac{\partial E}{\partial w_l}(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\frac{\delta}{\delta w_l}\left(\sum_{i=0}^{n} w_i x_{ki} - d_k\right)^2$$

$$= \sum_{k=1}^{p}\left(\vec{w}\cdot\vec{x}_k - d_k\right) x_{kl}$$

$$\Rightarrow \boxed{\nabla E(\vec{w})} = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \dots, \frac{\partial E}{\partial w_n}(\vec{w})\right) = \boxed{\sum_{k=1}^{p}\left(\vec{w}\cdot\vec{x}_k - d_k\right)\vec{x}_k}$$

- Learning algorithm

(Batch): $\boxed{\vec{w}(t+1)} = \vec{w}(t) - \varepsilon\cdot\nabla E(\vec{w}(t)) = \boxed{\vec{w}(t) - \varepsilon\cdot\sum_{k=1}^{p}\left(\vec{w}(t)\cdot\vec{x}_k - d_k\right)\cdot\vec{x}_k}$

(Online):

$$\boxed{\vec{w}(t+1) = \vec{w}(t) - \varepsilon(t)\cdot\left(\vec{w}(t)\cdot\vec{x}_k - d_k\right)\cdot\vec{x}_k}$$

MULTI LAYER PERCEPTRON

- **Notace**:
  - X ..... set vstupních neuronů (inputs)
  - Y .... set výstupních neuronů (outputs)
  - Z .... set všech neuronů ($X, Y \subseteq Z$)
  - $i, j, \dots$ indexy neuronů:
    - ξ_j .... vnitřní potenciál neuronu j $\left(\sum_{i \in j_{\leftarrow}} w_{ji}\, y_i\right)$ $\cancel{\vec{w}\cdot\vec{x}}$
    - y_j .... output neuronu j $(\sigma(\xi_j))$
  - w_{ji} .... váha z neuronu i do neuronu j
  - $j_{\leftarrow}$ .... set všech neuronů, které míří do j
  - $j^{\rightarrow}$ .... set všech neuronů, ~~které~~ na které míří j

- **Error function**

$\boxed{\{(\vec{x}_k, \vec{d}_k) \mid k = 1, \dots, p\}}$ – Tréninková sada

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$ ...... suma přes všechny tréninkové příklady

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} \left(y_j(\vec{w}, \vec{x}_k) - d_{kj}\right)^2$$

- **Learning algorithm (Batch)**

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

komponenta gradientu ∇E =>

$\Rightarrow \vec{w}(t+1) = \vec{w}(t) - \varepsilon(t) \cdot \nabla E(\vec{w}(t))$

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \boxed{\frac{\partial E}{\partial w_{ji}}(\vec{w}(t))}$$ ..... update w_{ji} v $t+1$ kroku

- **Error function gradient**
  - pro každé w_{ji}: (váhy)

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^{p} \boxed{\frac{\partial E_k}{\partial w_{ji}}}$$

  - a pro každé $j \in Z \setminus X$: (neurony)

$$\boxed{\frac{\partial E_k}{\partial y_j}} = y_j - d_{kj}$$ | pro $j \in Y$

  - kde pro každé k: (examples)

$$\boxed{\frac{\partial E_k}{\partial w_{ji}}} = \boxed{\frac{\partial E_k}{\partial y_j}} \cdot \sigma_j'(\xi_j) \cdot y_i$$

$$\boxed{\frac{\partial E_k}{\partial y_j}} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$ | pro $j \in Z \setminus (Y \cup X)$

... slovy:

- Změna chyby podle dané váhy je rovna součtu chyb všech tréninkových příkladů (podle dané váhy).
- Změna chyby příkladu (example) podle váhy je rovna změně chyby podle výstupu daného neuronu (do kterého váha vchází), vynásobené vnitřním potenciálem, na který je aplikovaná derivovaná aktivační fce, a výstupem neuronu, ~~pro dané váhy a výstupy~~ ~~váha vyjde~~ ze kterého daná váha vychází (o vrstvu níž)
- Změna chyby podle výstupu je rovna:
  - pro výstupní neurony: rozdíl reálného a očekávaného výstupu
  - pro ostatní neurony: • suma přes všechny neurony r, do kterých daný neuron j míří (o 1 vrstvu výš):
    - Změna chyby podle výstupu daného neuronu o vrstvu ~~níž~~ výš, krát váha mezi těmito 2 neurony, krát vnitřní potenciál neuronu r (vyšší vrstva), na který je aplikována derivovaná aktivační fce

- <u>Derivace aktivačních funkcí</u>

Logistická sigmoida: $\sigma_j'(\xi_j) = \lambda_j y_j (1 - y_j)$

Hyperbolický tangens: $\sigma_j(\xi_j) = a \cdot \tanh(b \cdot \xi_j)$ — modifikují tvar (strmost, rozložení) křivky

$$\sigma_j'(\xi_j) = \frac{b}{a}(a - y_j)(a + y_j)$$

- <u>Gradient Descent algorithm</u>

$\varepsilon_{ji} = 0$ (na konci $\varepsilon_{ji} = \frac{\partial E}{\partial w_{ji}}$)

for ($k = 1, \ldots, p$):

1) forward pass: – spočítej $y_j = y_j(\vec{w}; \vec{x}_k)$ pro ~~vš~~ všechny $j \in Z$
2) backward pass: – spočítej $\frac{\partial E_k}{\partial y_j}$ pro všechny $j \in Z$ pomocí $\boxed{\text{Backpropagation}}$
3) spočítej $\boxed{\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma_j'(\xi_j) \cdot y_i}$ pro všechny w_{ji}
4) $\varepsilon_{ji} = \varepsilon_{ji} + \frac{\partial E_k}{\partial w_{ji}}$

Výsledné $\varepsilon_{ji} = \frac{\partial E}{\partial w_{ji}}$

- **Backpropagation**

Spočítej $\frac{\partial E_k}{\partial y_j}$ pro všechna $j \in Z$:

- $j \in Y \Rightarrow \frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$

- $j \in Z \setminus (Y \cup X) \Rightarrow$ předpokl., že $\frac{\partial E_k}{\partial y_r}$ již známe:

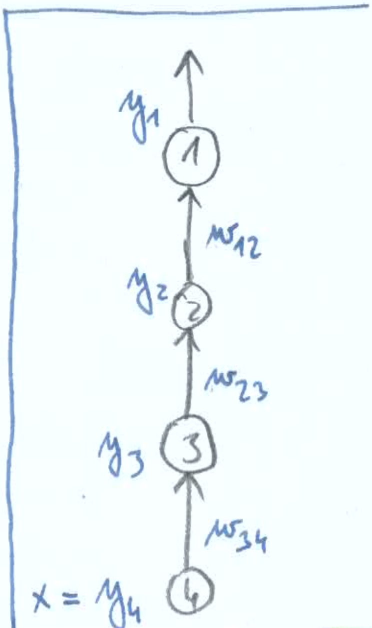
$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$

- **Chain rule**

$$\frac{\partial E_k}{\partial w_{12}} = \frac{\partial E_k}{\cancel{\partial y_1}} \cdot \frac{\cancel{\partial y_1}}{\cancel{\partial \xi_1}} \cdot \frac{\cancel{\partial \xi_1}}{\partial w_{12}} = \frac{\partial E_k}{\partial y_1} \cdot \sigma_1'(\xi_1) \cdot y_2$$

$$\frac{\partial E_k}{\partial w_{23}} = \dots = \frac{\partial E_k}{\partial y_2} \cdot \sigma_2'(\xi_2) \cdot y_3$$

$$\frac{\partial E_k}{\partial w_{34}} = \dots = \frac{\partial E_k}{\partial y_3} \cdot \sigma_3'(\xi_3) \cdot y_4$$



$y_1 = \sigma_1(\xi_1)$; $\xi_1 = w_{12} \cdot y_2$
$y_2 = \sigma_2(\xi_2)$; $\xi_2 = w_{23} \cdot y_3$
$y_3 = \sigma_3(\xi_3)$; $\xi_3 = w_{34} \cdot y_4$

- **Error functions**

– Square error: $E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$, kde

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} \left(y_j(\vec{w}, \vec{x}_k) - d_{kj} \right)^2$$

– Mean square error (mse):

$$E(\vec{w}) = \frac{1}{p} \sum_{k=1}^{p} E_k(\vec{w})$$

..... průměrná chyba objektů v daném batchi

HEURISTIKY - GRADIENT DESCENT

- Momentum

Problém: Moc velké kroky mohou po seběhnutí údolí (správným směrem) vyběhnout na druhé straně opět nahoru. ~~V příštím kroku nastane to samé a zase~~

Řešení: V každém kroku přičteme změnu provedenou minulým krokem (vynásobenou ~~A~~ faktorem α) → **HYBNOST**

$$\Delta \vec{w}(t) = -\varepsilon(t) \cdot \sum_{k \in T} \nabla E_k(\vec{w}^{(t)}) + \boxed{\alpha \cdot \Delta w_{ji}^{(t-1)}}$$

- Adaptující se rychlost učení (learning rate)

  - začínat s vyšším (např. $\varepsilon = 0.1$)
  - později snižovat (dohledávka lokálního minima)

~~Nebo~~ Případně:

- ~~[illegible]~~ Error ↓ => ε ↑ (chyba se může krátkodobě zvednout a však to nemusí vadit - minimum je schované za kopcem)
- Error ↑ => ε ↓

- Ada Grad

  - každá váha má svůj learning rate (ε), který se updatuje samostatně:
    - málo měnící se váhy => ~~[illegible]~~ velké ε
    - hodně měnící se váhy => malé ε

    } všechny váhy se mění ± stejně
  - problém: akumulace updatů během celého procesu => RMSProp minimalizuje historii (exponenciálně)

POŽADAVKY NA AKTIVAČNÍ FUNKCE

1) diferencovatelná .... většinou (ne nutně vždy - ReLU) → gradient descent počítá s derivací

~~2) non-linear~~

2) nelineární .... i více-vrstevná síť (lineární) je ekvivalentní jedno-vrstevné

3) monotónnost ..... bez lokálních extrémů

4) "lineární" ..... jednodušeji se učí → tradeoff
..... dnes se používají co nejvíc lineární

• **Input preprocessing**

- velké hodnoty mají výrazně větší vliv na trénink než malé

=> Standardizace :
- průměr = 0 (odečtení průměru)
- odchylka = 1 (podělení standardní odchylkou) "standard deviation"
  (variance)

- redukce dimenze : PCA (Principal Component Analysis)

• **Inicializace vah** interval $[-w; w]$; průměr = 0; odchylka = 1 (rozptyl); d = počet vstupů do neuronu

- příliš malá čísla => příliš lineární
- příliš velká čísla => ploché oblasti (rozseknutí)

Řešení: vybrat w takové, aby standardní odchylka je blízko přelomu mezi lineární a nelineární části fce. (ξ se tedy rozloží do ~~tohoto~~ tohoto intervalu)

$$\Rightarrow \boxed{w = \frac{\sqrt{3}}{\sqrt{d}}} \Rightarrow \left[-\frac{\sqrt{3}}{\sqrt{d}} ; \frac{\sqrt{3}}{\sqrt{d}}\right]$$

tedy rozloží vnitřní potenciál do intervalu $[-1; 1]$

• **Glorot & Bengio inicializace**

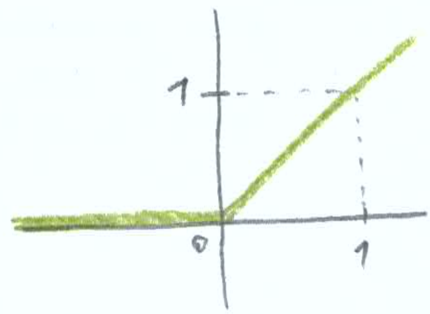
- pro lineární modely ideální
- lépe zohledňuje backward pass (řeší i počet neuronů nahoře, ne jen dole)

$$\boxed{w = \sqrt{\frac{6}{m+n}}}$$

• **Moderní aktivační funkce**

- Hidden layers: ReLU

$$\sigma(\xi) = \max\{0, \xi\}$$



- Output neurons:
  - regrese → lineární / sigmoida
  - klasifikace → binární → logistická sigmoida
    → tanh
  - → n-ární → Softmax
    (multi-class)

$$\boxed{\sigma_j(\xi_j) = \frac{e^{\xi_j}}{\sum_{i \in Y} e^{\xi_i}}}$$

... spolu se softmaxem se nejčastěji používá Cross-entropy jako error function

• Cross-entropy

$$E = -\frac{1}{p}\sum_{k=1}^{p}\sum_{j\in Y}\left[d_{kj}\ln(y_j) + (1-d_{kj})\ln(1-y_j)\right]$$

.... minimalizováním Cross-entropy maximalizujeme likelihood
"Maximum likelihood principle"

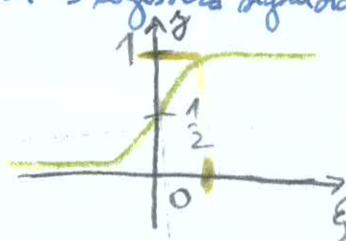
• Proč je Cross-entropy lepší než MSE (mean square error)?

Slovně: • Když se MSE plete, ~~…~~ zůstane relativně zaseklá ve špatném rozhodnutí => trénink je pomalý.

• Cross-entropy dělá sama od sebe velké kroky směrem od špatného rozhodnutí.

$\sigma' = \sigma(1-\sigma)$

Formálně: binární klasifikace - $\{0;1\}$; 1 output neuron → logistická sigmoida



$\boxed{\text{MSE:}}$ $$E_k^{mse} = \left(\sigma(wx_k) - d_k\right)^2$$

$$\frac{\partial E^{mse}}{\partial w} = \left(\sigma(wx_k) - d_k\right)\cdot\sigma'(wx_k)\cdot x_k =$$

$$= \left(\sigma(wx_k) - d_k\right)\cdot\boxed{\sigma(wx_k)}\left(1-\sigma(wx_k)\right)\cdot x_k$$

→ odečítáme tuto hodnotu od stávajících vah

$\boxed{d_k = 1}$:

• pokud ~~jsme ovšem špatně …~~ je ovšem vnitřní potenciál záporný (a my jej chceme dostat do kladných čísel), ~~…~~ $\frac{\partial E^{mse}}{\partial w}$ je malé číslo. Člen ● (wx_k je vnitřní potenciál => $\sigma(wx_k)$ je blízko 0).

⇓

Moc se nepohneme => Z tohoto špatného stavu se nedostaneme.

$\boxed{\text{CROSS-ENTROPY:}}$

$$E_k^{cross}(w) = -d_k\ln(\sigma(wx_k)) + \underbrace{(1-d_k)}_{=0}\ln(1-\sigma(wx_k))$$

$\boxed{d_k = 1}$: $$\frac{\partial E_k^{cross}}{\partial w} = -\frac{1}{\sigma(wx_k)}\cdot\sigma'(wx_k)\cdot x_k = -\frac{1}{\sigma(wx_k)}\cdot\sigma(wx_k)(1-\sigma(wx_k))\cdot x_k =$$

$$= -(1-\sigma(wx_k))\cdot x_k = \boxed{(\sigma(wx_k) - d_k)\cdot x_k}$$

⇒ velké záporné číslo → odečítáme od vah → posuneme se „doprava" → zlepšíme se

(pro $d_k = 0$ získáme to stejné)

• ~~Cross-validation~~ Early stopping (kdy skončit s učením)

- rozdělit dataset na 3 části:
  - training set (60%) – k trénování použít pouze tyto data
  - validation (20%) – ke zjištění, kdy zastavit učení
  - test set (20%) – k otestování sítě (porovnání s jinými modely, ...)

→ validation set se tedy „ušpiní" => na opravdové testování musíme ! VŽDY ! použít čerstvý, nedotknutý dataset

• Dropout

- v každém kroku gradient descentu vybereme náhodně jen podmnožinu neuronů, které se použijí do učení. Zbytek se pro tohle kolo ignoruje

• Weight decay

- při každé úpravě vah, vynásobíme koeficientem $0 < \alpha < 1$, a tím snížíme všechny váhy
- silné (důležité) přežijí a slabé (nedůležité) spadnou na 0

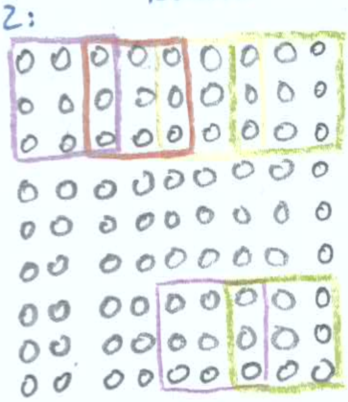
KONVOLUČNÍ SÍTĚ

• Zpracování obrazu používá téměř výhradně konvoluční sítě

• každý neuron ve skryté vrstvě (hidden layer) je napojen na čtverec $k \times k$ vstupních neuronů. Tento okno se s každým dalším neuronem ve skryté vrstvě posouvá (o hodnotu ~~shift~~ stride) a tvoří tak tzv. feature mapy.

stride ~~shift~~ = 2:

k = 3:



vstupní obrázek (~~pixely~~) (např. pixely)

feature map:

- všechny neurony ~~se~~ v jedné mapě spolu sdílí váhy
- mají pouze $k \times k$ (9) vah
- každá vrstva může mít několik takových feature map
- každá feature mapa „hledá v obrázku nějaký vzor"

Pooling

- redukce dimenze (každý neuron v pooling vrstvě je opět napojen na $n \times n$ čtverec ve feature mapě a abstrahuje je do 1 výstupu)
- nad feature mapou

→ Max-pooling: bere maximum

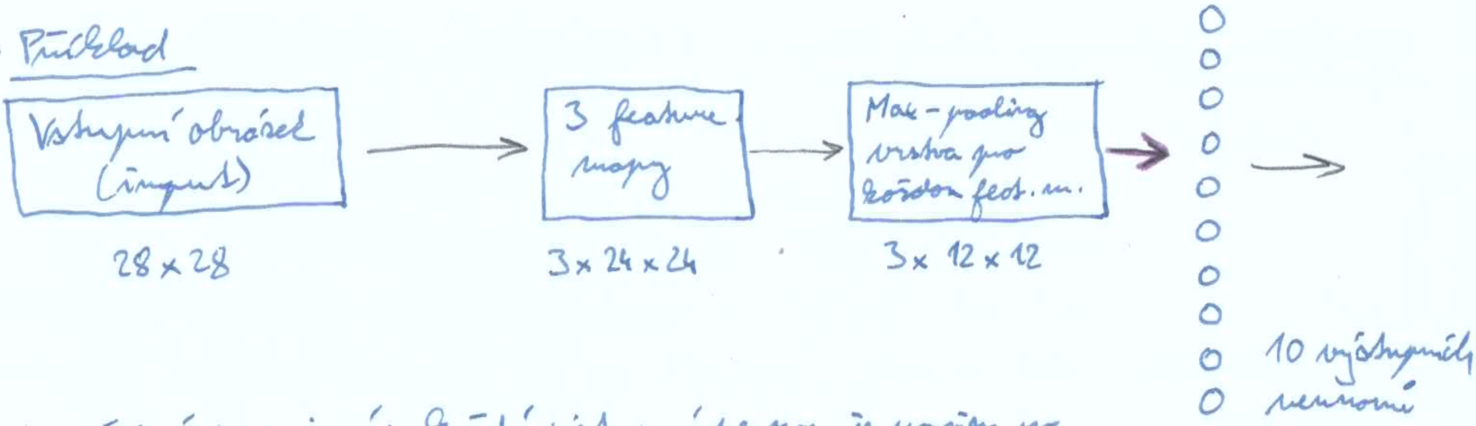
→ L2-pooling: odmocnina ze součtu 2. mocnin

→ Avg-pooling: ~~s~~ průměr

⋮

- nepřekrývají se (rozdíl od konvol. vrstvy)

• Příklad

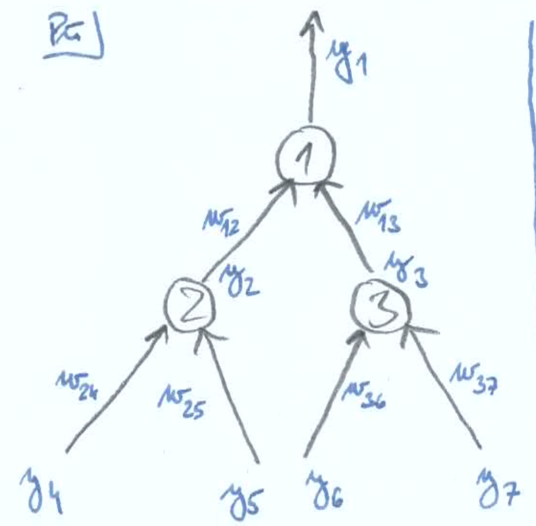


→ úplné propojení: každý výstupní neuron je napojen na úplně všechny neurony z pooling vrstev

⇒ Gradient descent pro trénink

• Jak vypočítat $\frac{\partial E_k}{\partial w_{ji}}$?

Př.)



$w_{24} = w_{36} = w_a$

$w_{25} = w_{37} = w_b$

$y_1 = \sigma_1(w_{12} y_2 + w_{13} y_3)$

$y_2 = \sigma_2(w_{24} y_4 + w_{25} y_5)$

$y_3 = \sigma_3(w_{36} y_6 + w_{37} y_7)$

1 feature mapa: 2, 3

1 input vrstva: y_4, y_5, y_6, y_7 (stride 2)

• pohnutí s w_a má vliv na y_2 i y_3 ⇒ sčítá se přes všechny neurony, které to ovlivní (které sdílí váhy)

$$\boxed{\frac{\partial E_k}{\partial w_a}} = \frac{\partial E_k}{\partial y_2} \cdot \frac{\partial y_2}{\partial w_a} + \frac{\partial E_k}{\partial y_3} \cdot \frac{\partial y_3}{\partial w_a} = \boxed{\frac{\partial E_k}{\partial y_2} \cdot \sigma_2' \cdot y_4 + \frac{\partial E_k}{\partial y_3} \cdot \sigma_3' \cdot y_6}$$

$$\frac{\partial E_k}{\partial w_{ji}} = \sum_{r \in j \text{ shora}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot y_r$$

proč tu má být y_r? – ve slajdech to tak je (str. 189), ale příklad z tabule ukazuje něco jiného

• Backpropagation

– pro $j \in Y$: $\frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$ (pro MSE)

– pro $j \in Z \setminus Y$, kde $j^{\rightarrow}$ je konvoluční vrstva nebo dense vrstva (úplně propojená):

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$

- pro $j \in Z \setminus Y$, kde $j^{\rightarrow}$ je max-pooling vrstva, pak $j^{\rightarrow} = \{i\}$ pro každý 'max' neuron:

$$\frac{\partial E_k}{\partial y_j} = \begin{cases} \frac{\partial E_k}{\partial y_i} & \text{pokud } j \text{ je ten maximální neuron} \\ 0 & \text{jinak} \end{cases}$$

DEEP MLP

PRO: - lepší výsledky
- těžší problémy
- 1 vrstva může být neefektivní

PROTI: - vanishing gradient
- rychlé přeučení (overfit)

· Vanishing gradient

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \boxed{\sigma_r'(\xi_r)} \cdot w_{rj}$$

pro ~~se~~ standardní aktivační fce (logistická sigmoida, tanh, ...) je ~~~~ menší než 1 => ~~~~ opakovaným násobením gradient stále snižujeme => v nižších vrstvách už není vůbec patrný

• Předtrénování MLP ~~~~ pomocí RBM

- každé 2 sousední vrstvy MLP můžeme považovat za RBM (Restricted Boltzman Machine) ..... B_i odpovídá vrstvě i a $i-1$ (B_1 = input + 1. skrytá)
- 1. fáze učení (unsupervised pretraining): ℓ = počet vrstev
  - bereme jen vstupy (ignorujeme očekávané výstupy)
  - na náhodné podmnožině trénovací sady natrénujeme B_i (pro $i = 1, \dots, \ell$) pomocí RBM algoritmu
  - tímto vlastně inicializujeme váhy MLP tak, že již v sobě obsahují jistou dávku informace (vzorů) z dat. - „nejdůležitější kousky skládačky" / „nejdříve trochu pochopíme doménu"
  - postupujeme od spodu - každou předtrénovanou podčást sítě (1. vrstva, 1. dvě vrstvy, ...) použijeme k transformaci dat pro následující vrstvu (2., 3., ...)

REKURENTNÍ SÍTĚ

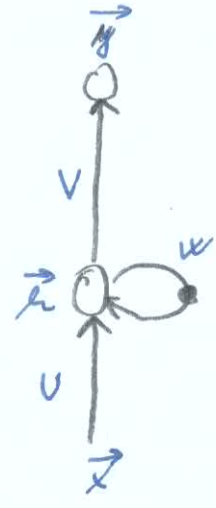
• Rekurentní MLP (se smyčkou):

- vstupy (input): $\vec{x}$
- stav: $\vec{h}$
- output: $\vec{y}$
- matice: U, W, V
  váhy

z minulého kroku - staré hodnoty

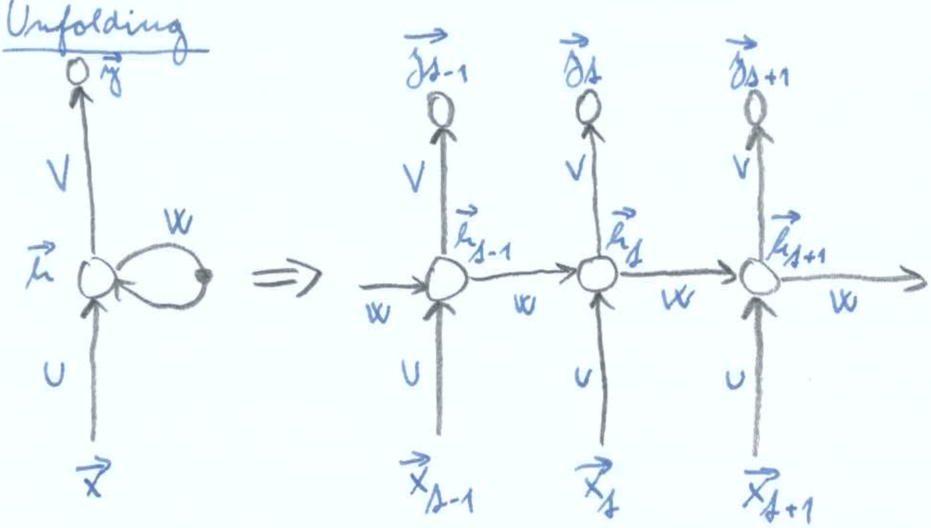
$\vec{h} = \sigma(U\vec{x} + W\vec{h})$ .... sigmoidal / ReLU

$\vec{y} = \sigma(V\vec{h})$ .... softmax / sigmoidal



=> lze zpracovávání sekvence vstupů => časové řady atd. (zpracování řeči)

• Unfolding



+1 dimenze

vstup = sekvence vektorů

výstup = sekvence vektorů

$\vec{h}_t$ je paměť sítě

$\vec{h}_t = \sigma(U\vec{x}_t + W\vec{h}_{t-1})$

$\vec{y}_t = \sigma(V\vec{h}_t)$

matice U, V, W jsou pro všechny stejné

• Learning

- training example (1 příklad): (x, d)

$X = \vec{x}_1, \dots, \vec{x}_T$ (sekvence vstupů)

$\vec{x}_t = (x_{t1}, \dots, x_{tN})$

(to stejné pro d - výstupy)

- rozbalení (unfolding) pro dané x vytvoří sekvenci vnitřních stavů:

$\vec{h}_0, \vec{h}_1, \vec{h}_2, \dots, \vec{h}_T$ kde $\vec{h}_t = (h_{t1}, \dots, h_{tH})$ $\boxed{\vec{h}_0 = (0, \dots, 0)}$

- sekvence výstupů: $\vec{y}_1, \dots, \vec{y}_T$ kde $\vec{y}_t = (y_{t1}, \dots, y_{tM})$

Chybová fce:

$$\boxed{E(x,d) = \sum_{t=1}^{T} \sum_{k=1}^{M} (y_{tj} - d_{tj})^2}$$

→ online verze (při minibatchi musíme přidat ještě jednu sumu)

• Backpropagation

$\nabla E_{(x,d)}(\vec{w}^{(t)})$ ..... vektor všech parciálních derivací $\frac{\partial E_{(x,d)}}{\partial w_{ji}}$

1) ~~pro~~ nahradit derivace podle w_{ji} za derivace podle y_j:

$$\frac{\partial E_{(x,d)}}{\partial w_{ji}} = \sum_{r \in j^{\text{shora}}} \frac{\partial E_{(x,d)}}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot y_i$$

2) pro $j \in Y$: (MSE)

$$\frac{\partial E_{(x,d)}}{\partial y_j} = y_j - d_j$$

3) pro $j \in Z \setminus Y$:

$$\frac{\partial E_{(x,d)}}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_{(x,d)}}{\partial y_r} \cdot \sigma_r'(\xi_r) \cdot w_{rj}$$

$\boxed{\text{RNN:}}$

- $$\frac{\partial E_{(x,d)}}{\partial h_{tk}} = \sum_{k'=1}^{N} \frac{\partial E_{(x,d)}}{\partial y_{tk'}} \cdot \sigma' \cdot V_{kk'} + \sum_{k'=1}^{H} \frac{\partial E_{(x,d)}}{\partial h_{(t+1)k'}} \cdot \boxed{\sigma' \cdot W_{kk'}}$$

- $$\frac{\partial E_{(x,d)}}{\partial V_{kk'}} = \sum_{t=1}^{T} \frac{\partial E_{(x,d)}}{\partial y_{tk}} \cdot \sigma' \cdot h_{tk}$$

- $$\frac{\partial E_{(x,d)}}{\partial W_{kk'}} = \sum_{t=1}^{T} \frac{\partial E_{(x,d)}}{\partial h_{tk}} \cdot \sigma' \cdot h_{(t-1)k}$$

- $$\frac{\partial E_{(x,d)}}{\partial U_{kk'}} = \sum_{t=1}^{T} \frac{\partial E_{(x,d)}}{\partial h_{tk}} \cdot \sigma' \cdot x_{tk}$$

• zpátky v čase = vanishing gradient

nebo exploding

$\Rightarrow$ řešení: LTSM

chceme co nejblíže 1, jinak

LSTM

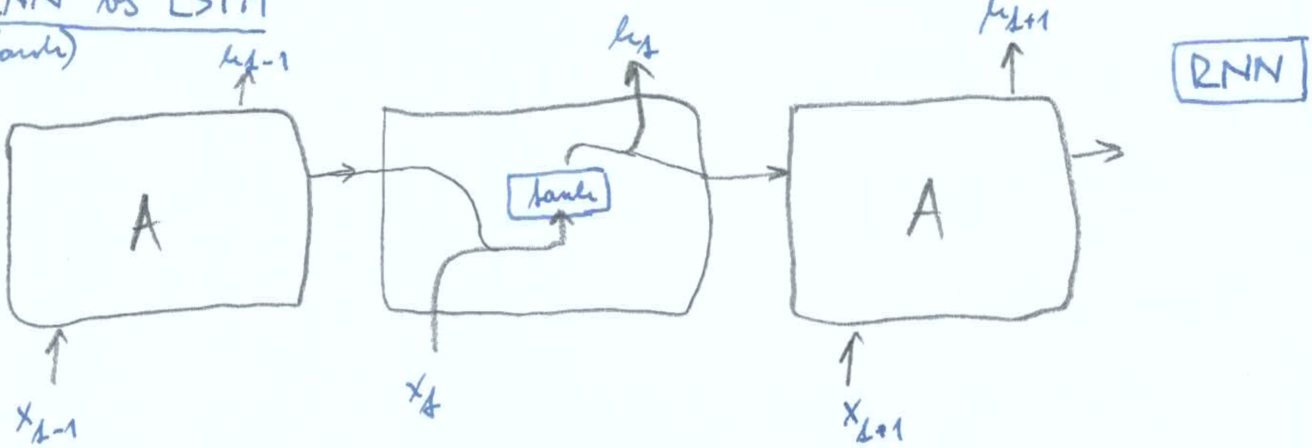
– každý neuron je nahrazen komplexní 'mašinou'

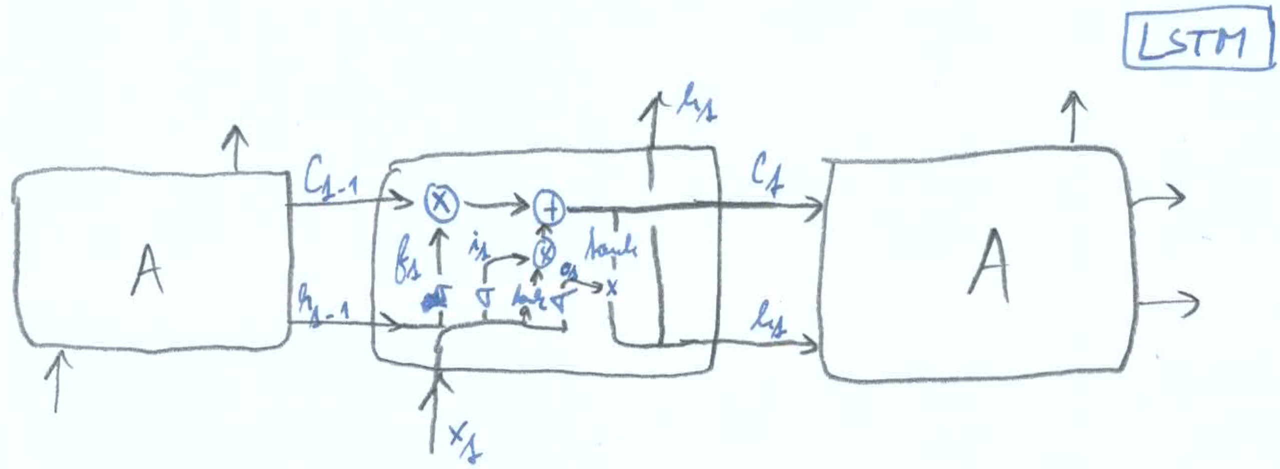
$\circ$ ..... hadamardův produkt (násobení po komponentách, ne dot product)

$\cancel{(\vec{y}_t =) \ \vec{h}_t = \vec{o}_t \circ \sigma_h}$

• RNN vs LSTM

(tanh)





$(\vec{y}_t =) \ \vec{h}_t = \vec{o}_t \circ \sigma_h(\vec{c}_t)$ output

$\vec{c}_t = \vec{f}_t \circ \vec{c}_{t-1} + \vec{i}_t \circ \tilde{c}_t$ memory

$\tilde{c}_t = \sigma_c(W_c \cdot \vec{h}_{t-1} + U_c \cdot x_t)$ new memory contents

$\vec{o}_t = \sigma_g(W_o \cdot \vec{h}_{t-1} + U_o \cdot \vec{x}_t)$ output gate

$\vec{f}_t = \sigma_g(W_f \cdot \vec{h}_{t-1} + U_f \cdot \vec{x}_t)$ forget gate

$\vec{i}_t = \sigma_g(W_i \cdot \vec{h}_{t-1} + U_i \cdot \vec{x}_t)$ input gate

– kontroluje, kolik informace má převzít z historie, kolik má zahodit, kolik má brát z nových dat, kolik poslat dál, atd.

KOHONENOVY MAPY

- **Vector quantization**

  - pro každý input $\vec{x}$, určíme nejbližší střed $\vec{w}_c(\vec{x})$:

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

  - následně minimalizujeme chybu:

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

→ chceme nahradit všechny objekty $\vec{x}$ středy $\vec{w}$ tak, abychom snížili dimenzi a zároveň dobře popsali dataset

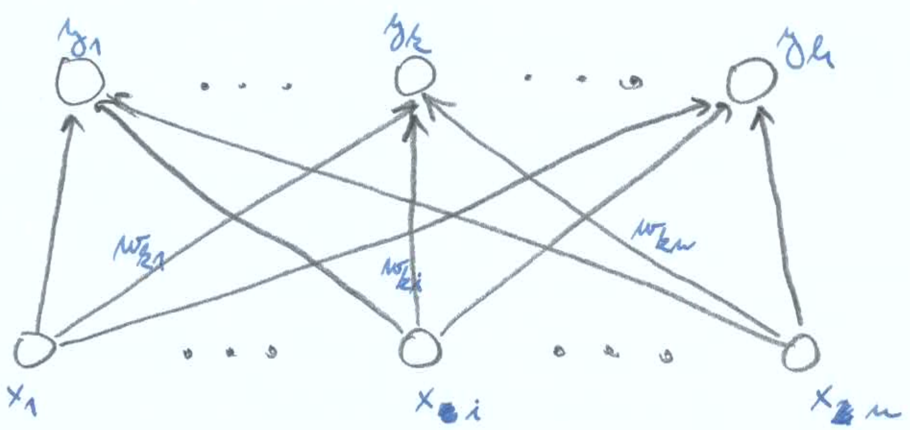
→ co nejvíc středů v hustých oblastech, co nejméně v řídkých

jak to vyřešit?

		ONLINE	
⇒	k-means (Lloyd's algorithm)	NE :(	– potřebujeme celý dataset
⇒	Kohonenovy mapy	ANO :)	

- **Z pohledu NN**

1 vrstva:



vstupní objekty: x

výsledné souřadnice středů: w

neurony Kohonenovy mapy jsou spolu pospojené a váhy propojení nezávisí na standardním pospojování (váhy, ...)

---

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

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

$\boxed{\text{Activity}}$

- **Learning**
  - bereme v úvahu topologii sítě, která je „nad" klasickou strukturou
    → propojení neuronů do mřížky

$d(c,k)$ = nejkratší cesta z c do k

topological neighbourhood neuronu c o velikosti s:

$$N_s(c) = \{ k \mid d(c,k) \le s \}$$

- **Krok učícího algoritmu**

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

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

$$\left. \begin{array}{l} \theta \in \mathbb{R} \\ s \in \mathbb{N}_0 \end{array} \right\} \text{ parametry, které se během tréninku mohou měnit}$$

- **V praxi**

2 fáze:
- coarse phase:
  - cca 1000 kroků
  - learning rate θ: od 0.1 do 0.01 (postupně)
  - sousedství neuronů: začínat s velkým okolím, postupně zmenšovat
- fine tuning
  - cca 500 × počet neuronů
  - θ blízko 0,01 (jinak je pravděpodobnost defektů)
  - jen malé sousedství neuronů

- **Klasifikace pomocí Kohonenových map**

1) Natrénuj mapu na vstupních datech (ignoruj třídy)
2) Označ neurony třídami: – každé třídě vyber ten neuron, který zastupuje nejvíc jejích instancí (obráceně)
3) dodatečný trénink (fine tune) za pomocí LVQ (později)

Použití: – spočítej pro daný vstup $\vec{x}$ nejbližší střed a vrať jeho třídu

• LVQ

pro každý trénovací příklad:

- najdi nejbližší neuron:
  - pokud ~~je správně vítěz~~ třída správně, posuň střed blíže k novému examplu
  - pokud je třída špatně, posuň ~~k~~ střed kousek od nového examplu

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

HOPFIELD NETWORK

- Klika - kompletní graf
- všechny neurony jsou zároveň vstupní i výstupní
- váhy jsou symetrické $w_{ji} = w_{ij}$

- Pokud síť dostane naučený vstup, měla by zastavit a dál se neměnit.
- Pokud síť dostane neznámý vstup, měla by se dostat do stavu odpovídajícímu nejbližšímu naučenému příkladu (vstupu)
- Training set je jen množina vektorů $\vec{x}_k \in \{-1;1\}^n$
- Hebb's rule: učíme síť, které dvojice neuronů mívají často stejnou hodnotu

$$w_{ji} = \sum_{k=1}^{p} x_{kj} x_{ki} \qquad 1 \le j \neq i \le n$$

.... pokud jsou x_{kj} a x_{ki} stejná (obě 1 nebo obě -1), vyjde veliké positivní číslo => jejich spojení by mělo být silné.

.... pokud jsou rozdílná => vyjde veliké negativní číslo => jejich spojení by mělo být slabé

- Activity: - nejprve nastavíme neurony na vstupy sítě
  - cyklicky upravujeme ~~s~~ stavy neuronů:
    1) vnitřní potenciál:

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

2)

$$y_j^{(t+1)} = \begin{cases} 1 & \xi_j^{(t)} > 0 \\ y_j^{(t)} & \xi_j^{(t)} = 0 \\ -1 & \xi_j^{(t)} < 0 \end{cases}$$

→ tento ~~p~~ proces zastaví, na konečném vstupu

$$y_j^{(t^*+n)} = y_j^{(t^*)} \qquad (j = 1, \dots, n)$$

• **Energie** (potenciální) pro každý stav $\vec{y} \in \{-1;1\}^n$

$$\boxed{E(\vec{y}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ji}\, y_j\, y_i}$$

- stavy s malou energií jsou stabilní (málo neuronů „chce změnit" svůj stav)
- stavy s velikou energií jsou málo stabilní => large (positive) $w_{ji} y_j y_i$ je stabilní
  small (negative) $w_{ji} y_j y_i$ není stabilní

• Energie se během procesu nezvyšuje

• **Phantoms**

= lokální minimum energetické fce, které nekoresponduje s tréninkovým příkladem

– mohou být "odnaučeni" (inverzní Hebb's rule):

$$w'_{ji} = w_{ji} - x_i x_j$$

• **Kapacita Hopfieldovy sítě**

$p \le n/(4 \log n)$

p = počet tréninkových příkladů schopných se naučit
n = počet neuronů

• **Boltzman activity**

– místo chození cyklicky dokola, v každém kroku vybereme náhodně neuron:

inner potencial:

$$\xi_j^{(t)} = \sum_{i=1}^{n} w_{ji}\, y_i^{(t)}$$

poté vyber náhodně 1 nebo −1 ($y_j^{(t+1)} \in \{-1,1\}$)

$$P[y_j^{(t+1)} = 1] = \sigma(\xi_j^{(t)})$$

kde

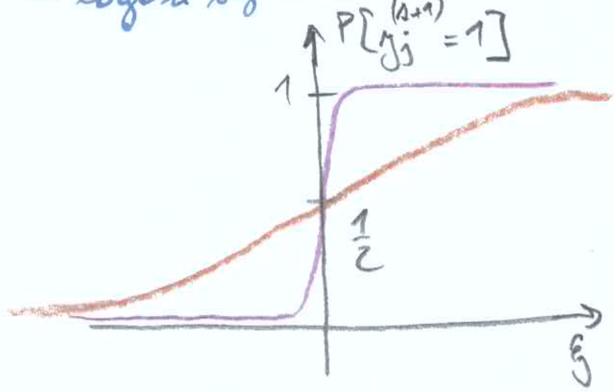
$$\boxed{\sigma(\xi) = \frac{1}{1 + e^{-2\xi/T(t)}}}$$

$T(t)$ je teplota v čase t

logistická sigmoida

• vysoká teplota => náhodné chování

• nízká teplota => stabilní chování



• Boltzman Activity = Hopfield network + random noise

• Energie může postoupit na vyšší úroveň, v závislosti na teplotě

• **Hopfield network – příklad**

$\boxed{\text{2 neurony:}}$

$$(1) \overset{w_{12}}{\text{———}} (2)$$

$\boxed{\text{4 stavy:}}$

$$\begin{tikzcd}
(1,1) \arrow[r, leftrightarrow, "a"] \arrow[d, leftrightarrow] \arrow[loop left] & (-1,1) \arrow[d, leftrightarrow] \arrow[loop above, "b"] \\
(1,-1) \arrow[r, leftrightarrow] \arrow[loop left] & (-1,-1) \arrow[loop below]
\end{tikzcd}$$

- pravděpodobnost přechodu a:

$P[a] = \frac{1}{2} \cdot P_1^a$ pst., že vybereme neuron 1

$\xi_1^{(t)} = w_{12} \cdot 1 \Rightarrow P_1^a = \frac{1}{1+e^{-2w_{12}}}$

P_1^a je pravděpodobnost 1 v neuronu 1

teplota je 1

$\Rightarrow$ velké váhy → neurony chtějí mít stejné hodnoty

$\Rightarrow$ malé váhy → neurony chtějí mít různé hodnoty

• **Simulated annealing**

– k lepšímu dosažení glob. minima energie

• začít s vysokou teplotou a postupně ji snižovat

$\boxed{\text{BOLTZMAN MACHINE}}$

• **Architektura**

- NN s cykly ~~acykl~~ a neorientovanými vahami
- jinak vlastně stejné jako Hopfield N.

• **Aktivita:**

- v každém ~~kr~~ kroku $t+1$:
  1) vyber náhodně neuron
  2) spočítej vnitřní potenciál
  $$\xi_j^{(t)} = \sum_{i \in j_{\leftarrow}}^{n} w_{ji} y_i^{(t)}$$
  3) vyber $y_j^{(t+1)} \in \{-1;1\}$ náhodně ~~tak~~:
  $$P[y_j^{(t+1)} = 1] = \sigma(\xi_j^{(t)}) \quad \ldots\ldots \quad \sigma = \boxed{18}$$

$\Rightarrow$ definuje pravděpodobnostní rozložení vektorů $\{-1;1\}^{|N|}$

→ když necháme BM běžet dostatečně dlouho, relativní ~~...~~ četnost jednotlivých stavů bude nezávislá na iniciálním stavu

→ tyto četnosti bereme jako pravděpodobnosti jednotlivých stavů

→ během učení získáme pravděpod. rozložení stavů $\{-1;1\}^{|N|}$ a podle nich adaptujeme váhy sítě, aby odpovídaly dané pravděp.

- Equilibrium
  - fixovaná teplota
  - dál tomu vůbec nerozumím :(

- Hidden neurons
  - rozdělíme N na 2 množiny:
    - visible V
    - hidden H

pravděp., že stav viditel. neuronů v termín. equil. je α

$\alpha \in \{-1;1\}^{|V|}$:

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

- ~~α je pst, že stav viditelných neuronů je v termálním equilibrium~~

given target prob.

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

current learned prob.

- Maximum likelyhood

$p_d(\alpha) = \#(\alpha, T) / m$, kde $\#(a,T)$ je počet výskytů α v T, $m = |T|$

naším cílem je najít konfiguraci sítě W takovou, že $p_V \approx p_d$

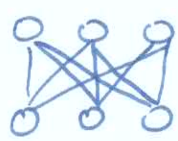
$\boxed{\text{likelyhood:}}$

$$L(T) = \prod_{i=1}^{m} P_V(\vec{x}_i)$$ ← toto chceme maximalizovat

- Gradient descent

RESTRICTED BOLTZMAN MACHINE

- kompletní bipartitní graf



V

H

- stavy nejsou -1 a 1, ale $\vec{y} \in \{0,1\}^{|N|}$
- bias: w_{j0} ---- y_0 je stále 0

- Activity
  - v každém sudém kroku: náhodně vyber hodnoty všech hidden neuronů
  - v každém lichém kroku: náhodně vyber hodnoty všech visible neuronů

- pro každé $j \in H$:

$$P\left[y_j^{(t+1)} = 1\right] = \frac{1}{1 + \exp\left(-w_{j0} - \sum_{i \in V} w_{ji}\, y_i^{(t)}\right)}$$

- pro $j \in V$:

to stejné, ale $\sum_{i \in H}$