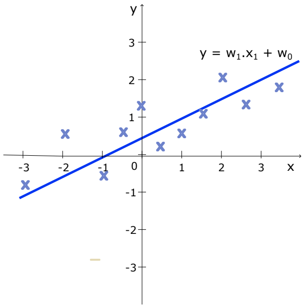
Co jsme probírali minule

1. **Přehled různých přenosových funkcí**

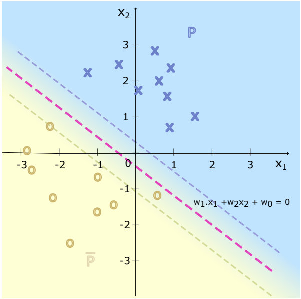
- lineární

Úloha lineární regrese



- sigmoida, hyperbolický tangens

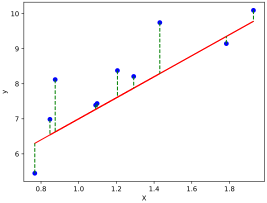
Úloha lineární klasifikace



Co jsme probírali minule

2. **Úloha lineární regrese a její řešení metodou nejmenších čtverců**

- chceme aby se skutečný výstup neuronu y_p se co nejméně lišil od požadovaného d_p
- minimalizujeme chybovou funkci SSE (součet čtverců) v prostoru vah:



$$E_{SSE}(\vec{w}) = \frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2$$

Co jsme probírali minule

3. **Úloha lineární regrese a její řešení metodou nejmenších čtverců**

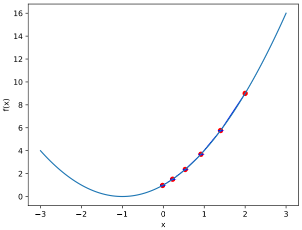
- Učení lineární regrese přímým výpočtem (metoda LSQ)

$$\vec{w} = (X^T X)^{-1} X^T \vec{d}$$

popř.:

$$\vec{w} = \lim_{\lambda \to 0+} (X^T X + \lambda I)^{-1} X^T \vec{d}$$

- Gradientní metoda učení



$$\vec{w}(t+1) = \vec{w}(t) - \alpha \nabla E_{SSE}(\vec{w}) = \\ = \vec{w}(t) + \alpha(d_t - y_t)\vec{x}_t^T$$

Co jsme probírali minule

4. **Gradientní metoda pro libovolnou spojitou (a diferencovatelnou) přenosovou funkci f**

Chybová funkce SSE:

$$E(\vec{w}) = \sum_{p=1}^{N} E_p(\vec{w}) = \frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2 = \frac{1}{2}\sum_{p=1}^{N}\left(d_p - f(\sum_{i=0}^{n} w_i.x_{pi})\right)^2$$

Parciální derivace:

$$\frac{\partial E_p}{\partial w_i} = \left(d_p - f(\sum_{i=0}^{n} w_i.x_{pi})\right) f'(\xi_p)(-x_{pi}) = -(d_p - y_p)f'(\xi_p)x_{pi}$$

Adaptační pravidlo (po předložení p-tého vzoru v čase t)

$$w_i(t+1) = w_i(t) - \alpha\frac{\partial E_p}{\partial w_i} = w_i(t) + \alpha f'(\xi_p)(d_p - y_p)x_{pi}$$

pro vektor vah:

$$\vec{w}(t+1) = \vec{w}(t) - \alpha\nabla E_p(\vec{w}) = \vec{w}(t) + \alpha f'(\xi_p)(d_p - y_p)\vec{x}_p$$

Co jsme probírali minule

Obecné schéma gradientní metody (GD, gradient descent)

1. Inicializuj váhy malými náhodnými reálnými hodnotami ....
   $\vec{w}(0) = (w_0, w_1, ..., w_n)^T$
   Inicializuj parametr učení α_0.... $1 > \alpha_0 > 0$
2. Předlož další trénovací vzor $(\vec{x}_t, d_t)$ a spočti potenciál a skutečný výstup neuronu:
   $\xi_t = \vec{x}_t \vec{w}$
   $y_t = f(\xi_t)$
3. Adaptuj váhy:
   $$\vec{w}(t+1) = \vec{w}(t) + \alpha_t f'(\xi_t)(d_t - y_t)\vec{x}_t^T$$
4. Případně aktualizuj parametr učení : $\alpha_t \rightarrow \alpha_{t+1}$
5. Pokud není konec, přejdi ke kroku 2.

Diskuse gradientní metody

Co ovlivňuje, jak velký bude přírůstek vah?

$$\triangle\vec{w}(t) + \alpha_t f'(\xi_t)(d_t - y_t)\vec{x}_t^T$$

- $(d_t - y_t)$ ... rozdíl mezi skutečným a požadovaným výstupem
- α ... parametr učení (learning rate)
- $\vec{x}_t^T$ ... vstupní vzor $\rightarrow$ důležitost normalizace
- $f'(\xi_t)$ ... pro sigmoidu / hyperbolický tangens klesá s rostoucí vzdáleností vzoru od dělící nadroviny
  $\rightarrow$ riziko **saturace** neuronu

Jak se vyhnout saturaci neuronu?

- důležitost normalizace vstupních dat a inicializace vah ,,kolem nuly"
- popřípadě pro sigmoidu / hyperbolický tangens použít jinou chybovou funkci (např. cross-entropy)

Dnešní hodina

1. **Neuronová síť s jednou vrstvou neuronů**
2. Vícevrstvá (vrstevnatá) neuronová síť (MLP)
3. Algoritmus zpětného šíření (backpropagation)

Neuronová síť s jednou vrstvou neuronů

Motivace: příklad klasifikace do více tříd

Velikost	**Srst**	**Mluví?**	**Třída**
Malá	Krátká	Ne	Kočka
Velká	Dlouhá	Ne	Pes
Malá	Žádná	Ano	Papoušek
Střední	Krátká	Ne	Kočka
Malá	Střední	Ne	Pes
...			...

Jak na to?

- nejprve hodnoty příznaků převedeme z kategoriálních hodnot na numerické

Neuronová síť s jednou vrstvou neuronů

Motivace: příklad klasifikace do více tříd

Velikost	Srst	Mluví?	Třída
-1	0	-1	1 (Kočka)
1	1	-1	2 (Pes)
-1	-1	1	3 (Papoušek)
0	0	-1	1 (Kočka)
...			...

Kategorie → numerické hodnoty

1. převedeme hodnoty jednoduše na -1 a 1 (binární hodnoty)
2. převedeme každou kategorii na číslo a normalizujeme na interval $[-1, 1]$ (pokud kategorie lze uspořádat)
3. převedeme jeden příznak na k příznaků s hodnotami -1 a 1 (pokud kategorie nelze uspořádat)
4. převedeme každou kategorii na číslo (hodnoty 1...k pro k kategorií), v případě neuronové sítě nepraktické

Neuronová síť s jednou vrstvou neuronů

Motivace: příklad klasifikace do více tříd

Velikost	Srst	Mluví?	Třída
-1	0	-1	1 (Kočka)
1	1	-1	2 (Pes)
-1	-1	1	3 (Papoušek)
0	0	-1	1 (Kočka)
...			...

Jak úlohu naučíme?

1. pro každou kategorii naučíme jeden neuron (jeden po druhém) a slepíme výsledky ...

Velikost	Srst	Mluví?	Kočka
-1	0	-1	1
0	1	-1	-1
-1	-1	1	-1
0	0	-1	1
...			...

Neuronová síť s jednou vrstvou neuronů

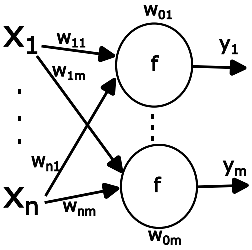
Motivace: příklad klasifikace do více tříd

Velikost	Srst	Mluví?	Kočka	Pes	Papoušek
-1	0	-1	1	-1	-1
0	1	-1	-1	1	-1
-1	-1	1	-1	-1	1
0	0	-1	1	-1	-1
...			...		

Jak úlohu naučíme?

2. **lépe:** sestrojíme neuronovou síť s jednou vrstvou neuronů a naučíme ji najednou

Neuronová síť s jednou vrstvou neuronů

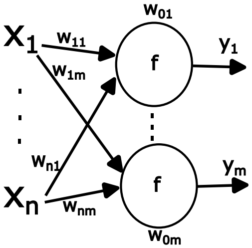


- Model je reprezentován maticí vah

$$W = \begin{pmatrix} w_{01} & w_{02} & \dots & w_{0m} \\ \dots & \dots & & \dots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{pmatrix}$$

  každému neuronu odpovídá jeden sloupec matice
- mají-li jednotlivé neurony přenosovou funkci $f : R \to R$, definujeme $f(\vec{\xi}) = (f(\xi_1), ..., f(\xi_N))^T$

Neuronová síť s jednou vrstvou neuronů



- Výstup modelu:

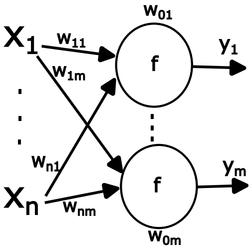
$$\vec{y} = f(\vec{\xi}) = f(\vec{x}W)$$

- Trénovací množina je ve tvaru $T = (X, D)$

$x_{10} = 1$	x_{11}	$\dots$	x_{1n}	d_{11}	$\dots$	d_{1m}
$\dots$	$\dots$		$\dots$	$\dots$		$\dots$
$x_{N0} = 1$	x_{N1}	$\dots$	x_{Nn}	d_{N1}	$\dots$	d_{Nm}

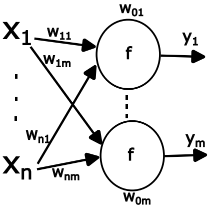
$$Y = f(\Xi) = f(XW)$$

Neuronová síť s jednou vrstvou neuronů



1. Neurony s lineární přenosovou funkcí
   → **mnohorozměrná lineární regrese**
   - lineární neuronová síť
2. Neurony se skokovou nebo tanh přenosovou funkcí (popř. logsig)
   → **lineární klasifikace do více tříd (úloha rozpoznávání vzorů)**
   - jednovrstvý perceptron

Lineární neuronová síť



- tvořena jednou vrstvou lineárních neuronů (více vrstev by nepřineslo žádný benefit)
- mnohorozměrná lineární regrese
- výstup modelu:

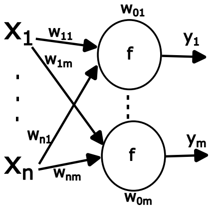
$$Y = XW$$

Učení metodou LSQ

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

Učení gradientní metodou

Jednovrstvá neuronová síť se spojitou přenosovou funkcí



Učení gradientní metodou

1. SGD (stochastická gradientní metoda) - v každém kroku adaptuji váhy jen jednoho náhodně zvoleného neuronu
2. váhové vektory všech neuronů adaptuji současně

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Učení gradientní metodou ... chybová funkce SSE

$$E_{SSE}(W) = \sum_{j=1}^{m} E_{SSE}(\vec{w}_j) = \frac{1}{2}\sum_{j=1}^{m}\sum_{p=1}^{N}(d_{pj} - y_{pj})^2$$

$$= \frac{1}{2}\sum_{j=1}^{m}\sum_{p=1}^{N}\left(d_{pj} - f(\sum_{i=0}^{n} w_{ij}x_{pi})\right)^2 = \sum_{p=1}^{N} E_p(\vec{w}_j)$$

Parciální derivace:

$$\frac{\partial E_p}{\partial w_{ij}} = \left(d_{pj} - f(\sum_{i=0}^{n} w_{ij}x_{pi})\right) f'(\xi_{pj})(-x_{pi}) = -(d_{pj} - y_{pj}) f'(\xi_{pj}) x_{pi}$$

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Učení gradientní metodou

- Adaptační pravidlo pro jednu váhu:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha_t (d_{tj} - y_{tj}) f'(\xi_{tj}) x_{ti}$$

- Adaptační pravidlo pro jeden neuron s vektorem vah $\vec{w}_j$:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha_t (d_{tj} - y_{tj}) f'(\xi_{tj}) \vec{x}_t^t$$

- Adaptační pravidlo pro jednu vrstvu neuronů s maticí vah W:

$$W(t+1) = W(t) + \alpha_t \vec{x}_t^T [f'(\vec{\xi}_t) \circ (\vec{d}_t - \vec{y}_t)]$$

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

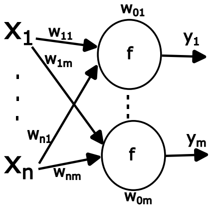
Obecné schéma gradientního algoritmu (GD, gradient descent)

1. Inicializuj váhy malými náhodnými reálnými hodnotami ....
   $W(0)$ tvaru $(n+1) \times m$
   Inicializuj parametr učení α_0.... $1 > \alpha_0 > 0$
2. Předlož další trénovací vzor $(\vec{x}_t, \vec{d}_t)$ a spočti potenciál a skutečný výstup modelu: $\vec{\xi}_t = \vec{x}_t W$
   $\vec{y}_t = f(\vec{\xi}_t)$
3. Adaptuj váhy:

$$W(t+1) = W(t) + \alpha_t \vec{x}_t^T [f'(\vec{\xi}_t) \circ (\vec{d}_t - \vec{y}_t)]$$

4. Případně aktualizuj parametr učení : $\alpha_t \to \alpha_{t+1}$
5. Pokud není konec, přejdi ke kroku 2.

Lineární klasifikace do více tříd (rozpoznávání vzorů)



- Bipolární neurony se skokovou nebo tanh přenosovou funkcí

Příklad klasifikační úlohy

x_1	x_2	x_3	**Třída**
1.5	-2.6	3.7	Třída 1
2.1	3.2	-0.5	Třída 2
-1.0	1.8	2.9	Třída 1
-2.2	0.5	2.0	Třída 3
...	...	...	...

Lineární klasifikace do více tříd (rozpoznávání vzorů)

Příklad klasifikační úlohy

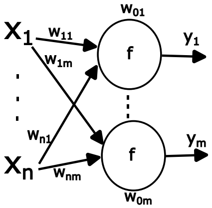
x_1	x_2	x_3	**Třída**
1.5	-2.6	3.7	Třída 1
2.1	3.2	-0.5	Třída 2
-1.0	1.8	2.9	Třída 1
-2.2	0.5	2.0	Třída 3
...	...	...	...

- pokud by vstupní příznaky nebyly normalizované, měly by se normalizovat

$\rightarrow$ **Trénovací množina pro bipolární model:**

x_0	x_1	x_2	x_3	d_1	d_2	d_3
1	1.5	-2.6	3.7	1	-1	-1
1	2.1	3.2	-0.5	-1	1	-1
1	-1.0	1.8	2.9	1	-1	-1
1	-2.2	0.5	2.0	-1	-1	-1

Lineární klasifikace do více tříd (rozpoznávání vzorů)



- pro každou třídu naučíme jeden perceptron (gradientní metodou nebo Rosenblattovým algoritmem v závislosti na zvolené přenosové funkci)

Jak zjistím vítěznou třídu?

- argmax

$$k_{max} = argmax_k y_k$$

- softmax - pouze pro spojitou přenosovou funkci

$$softmax(y_k) = \frac{e^{y_k}}{\sum_{j=1}^{m} e^{y_j}}$$

Dnešní hodina

1. Neuronová síť s jednou vrstvou neuronů
2. **Vícevrstvá (vrstevnatá) neuronová síť (MLP)**
3. Algoritmus zpětného šíření (backpropagation)

Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Neuronová síť s hierarchickou architekturou:

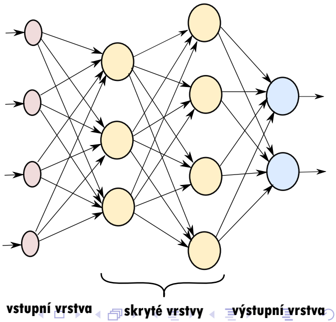
- neurony jsou uspořádány do vrstev
- všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

Speciální vstupní vrstva:

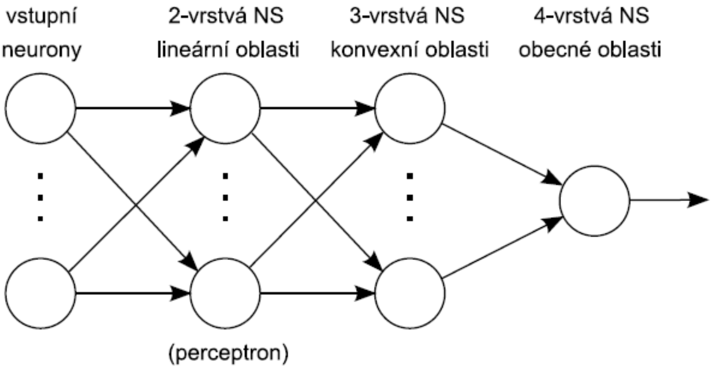
- odpovídá vstupům neuronové sítě

Výstupní vrstva

- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů

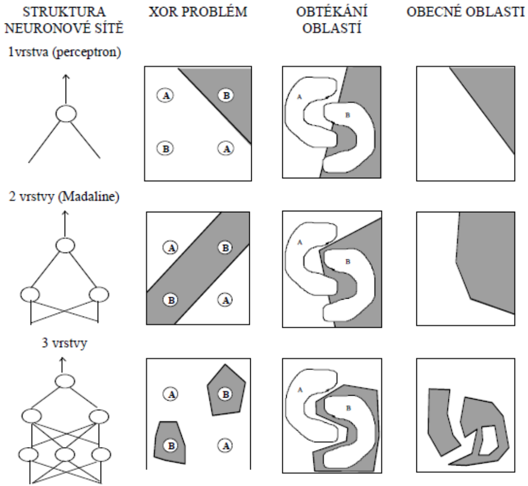


Motivace: Vícevrstvý perceptron



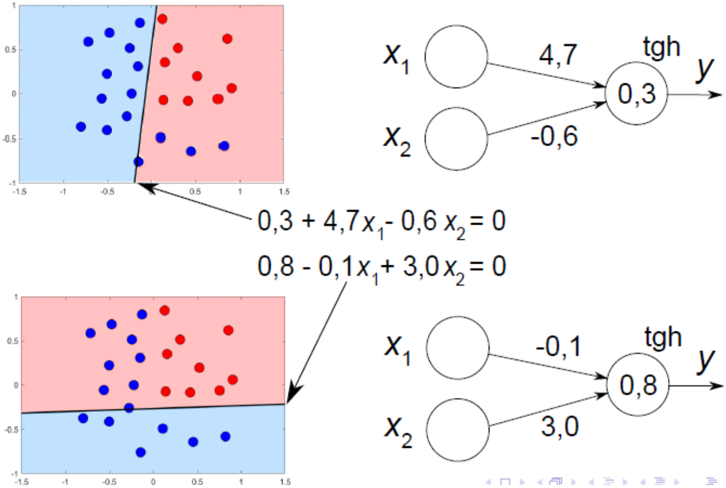
Zdroj: E.Volná: Neuronové sítě 1, Ostrava, 2008, Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

Motivace: Vícevrstvý perceptron



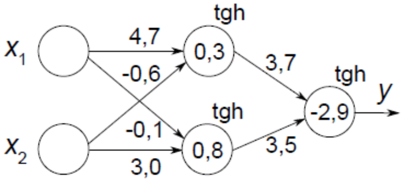
Zdroj: E.Volná: Neuronové sítě 1, Ostrava, 2008

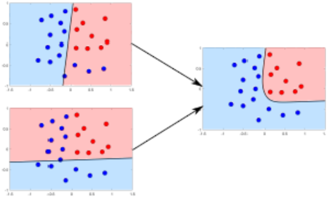
Motivace: Vícevrstvý perceptron



Zdroj: Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

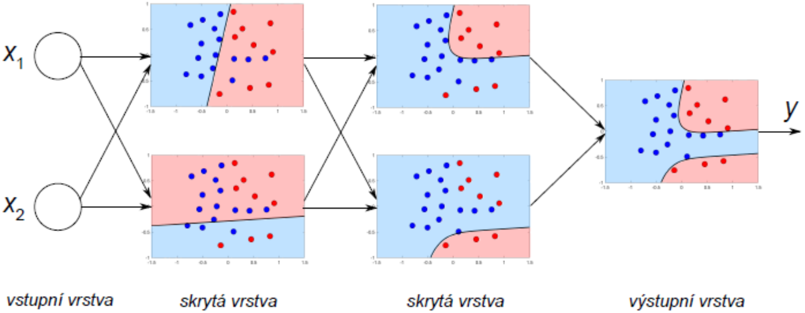
Motivace: Vícevrstvý perceptron





Zdroj: Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

Motivace: Vícevrstvý perceptron



Zdroj: Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Neuronová síť s hierarchickou architekturou:

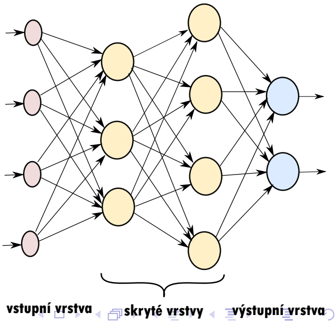
- neurony jsou uspořádány do vrstev
- všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

Speciální vstupní vrstva:

- odpovídá vstupům neuronové sítě

Výstupní vrstva

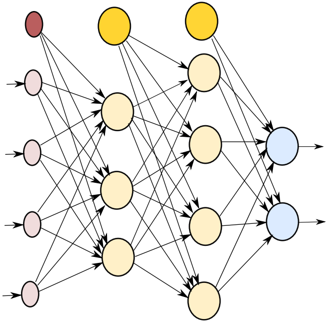
- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů



Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Fiktivní neurony

- ke každé skryté vrstvě (podobně jako ke vstupní) přidáme jeden fiktivní neuron s konstantním výstupem 1, který bude reprezentovat prahy (biasy) neuronů v následující vrstvě

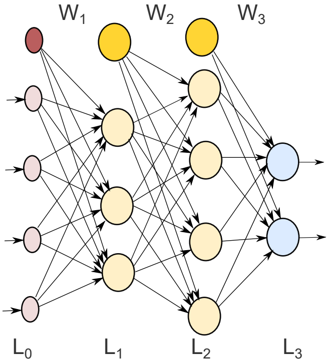


Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Maticová reprezentace vrstevnaté neuronové sítě:

- mějme vrstevnatou neuronovou síť s vrstvami L_0 (vstupní),...,L_{max} (výstupní)
- váhy všech neuronů můžeme reprezentovat pomocí matic $W_1, \ldots, W_{L_{max}}$
- W_L ... matice vah mezi vrstvou $L-1$ a L o rozměrech $(n_{L-1}+1) \times n_L$

(podobně jako pro jednovrstvou neuronovou síť)



Vrstevnatá neuronová síť (multi-layer neural network, MLP)

Konfigurace neuronové sítě:

- vektor vah všech neuronů v síti $\vec{w} \sim$ vektor všech parametrů modelu

Jak vrstevnatou neuronovou síť budeme učit?

- můžeme použít gradientní metodu pro zvolenou chybovou funkci E a vektor všech parametrů modelu $\vec{w}$:

$$\vec{w}(t+1) = \vec{w}(t) - \alpha \nabla E(\vec{w})$$

- výpočet parciálních derivací a proces adaptace vah budou složitější než u jedné vrstvy neuronů
- výpočet značně zjednoduší algoritmus zpětného šíření chyby (backpropagation)

Algoritmus zpětného šíření (Backpropagation)

(Werbos, Rumelhart, 1974-1986)

Máme k dispozici

- Trénovací množina T s N trénovacími vzory $(\vec{x}_p, \vec{d}_p)$.
  - $x^p = (x_1^p, ..., x_n^p)$ ... vstupní vzor
  - $d^p = (d_1^p, ..., d_m^p)$ ... požadovaný výstup

$x_{10} = 1$	x_{11}	...	x_{1n}	d_{11}	...	d_{1m}
...	...		...	...		...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_{N1}	...	d_{Nm}

- Vrstevnatá neuronová síť s danou architekturou a s $n+1$ vstupními a m výstupními neurony. Neurony musí mít spojitou, diferencovatelnou přenosovou funkci.

Cíl

- Nastavit váhy všech neuronů v síti tak, aby byl skutečný výstup sítě stejný jako požadovaný.

Algoritmus zpětného šíření (Backpropagation)

Cílová (chybová) funkce

- místo SSE se často používá MSE (mean squared error) ... průměrná chyba přes všechny vzory $E_{MSE} = E_{SSE}/N$
  - pro jeden trénovací vzor:

$$E_p(\vec{w}) = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$$

  - pro celou trénovací množinu:

$$E(\vec{w}) = \frac{1}{N}\sum_{p=1}^{N} E_p = \frac{1}{2N}\sum_{p=1}^{N}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$$

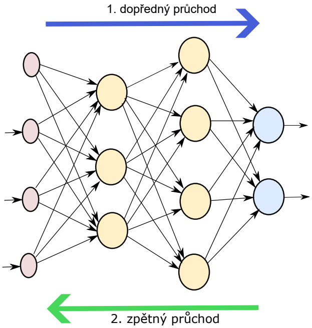
Cíl algoritmu zpětného šíření

- minimalizace chybové funkce E na dané trénovací množině T
- obecně lze ale použít i jinou chybovou funkci (např. cross-entropy)

Algoritmus zpětného šíření (Backpropagation)

Základní princip

1. Spočteme skutečnou odezvu sítě pro daný trénovací vzor.
   - od vstupní vrstvy směrem k výstupní
2. Porovnáme skutečnou a požadovanou odezvu sítě.
3. Adaptujeme váhy a prahy:
   - proti směru gradientu chybové funkce
   - od výstupní vrstvy směrem ke vstupní



Výpočet odezvy neuronové sítě

3. Předložíme vstupní vzor $\vec{x}_t$
4. Postupujeme ve směru od vstupní vrstvy k výstupní a pro každý neuron j spočteme (a zapamatujeme si) jeho výstup y_{tj} (využijeme k tomu aktivity neuronů v předchozí vrstvě, včetně fiktivního)
$$y_{tj} = f(\xi_{tj}) = f(\sum_i w_{ij} y_{ti})$$
(i je index přes neurony ve vrstvě předcházející neuronu j)
5. Výstup sítě $\vec{y} = (y_1, \dots, y_m)$ tvoří výstupy neuronů ve výstupní vrstvě

Výpočet odezvy neuronové sítě maticově I.

1. Předložíme vstupní vzor $\vec{x}$
2. Pro vrstvy $L = L_0, L_1, \ldots, L_{max}$ postupně spočítáme vektory výstupů těchto vrstev $\vec{y}_0, \ldots, \vec{y}_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $\vec{y}_0 = \vec{x}$
   - Pro $L = L_1 \ldots, L_{max}$:

$$\vec{z}_L = f(\vec{\xi}_L) = f(\vec{y}_{L-1} W_L)$$

$$\vec{y}_L = (1|\vec{z}_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $\vec{y} = (y_1, \ldots, y_m) = \vec{y}_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě

Výpočet odezvy neuronové sítě maticově II.

1. Předložíme matici vstupních vzorů X
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme matice výstupů těchto vrstev $Y_0, ..., Y_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $Y_0 = X$
   - Pro $L = L_1, .., L_{max}$:

$$Z_L = f(\xi_L) = f(Y_{L-1} W_L)$$

$$Y_L = (1|Z_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $Y = Y_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě

Backpropagation - Adaptační pravidla

- Chybová funkce pro jeden trénovací vzor $(\vec{x}_t, \vec{d}_t)$ a skutečný výstup sítě $\vec{y}_t$:

$$E_t = \frac{1}{2}\sum_{j=1}^{m}(d_{tj} - y_{tj})^2$$

- Parciální derivace:

$$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \frac{\partial \xi_{tj}}{\partial w_{ij}}$$

- Adaptační pravidlo pro váhu z neuronu i do neuronu j v čase t:

$$w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t),$$

- $\triangle w_{ij}(t)$ je přírůstek váhy w_{ij} přispívající k minimalizaci E_t

$$\triangle w_{ij}(t) = -\alpha \frac{\partial E_t}{\partial w_{ij}} = -\alpha \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \frac{\partial \xi_{tj}}{\partial w_{ij}}$$

- α je parametr učení (learning rate)

Backpropagation - Adaptační pravidla

Označme

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = \frac{\partial E_t}{\partial \xi_{tj}} \text{ ... chybový člen pro nejuron } j$$

Pak

$$\triangle w_{ij}(t) = -\alpha \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \frac{\partial \xi_{tj}}{\partial w_{ij}} = -\alpha \delta_{tj} \frac{\partial \xi_{tj}}{\partial w_{ij}} = -\alpha \delta_{tj} y_{ti}$$

Pro výstupní vrstvu:

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = -(d_{tj} - y_{tj}) f'(\xi_{tj})$$

$$\triangle w_{ij}(t) = \alpha (d_{tj} - y_{tj}) f'(\xi_{tj}) y_{ti}$$

Back-propagation - Adaptační pravidla

Pro skryté vrstvy:

$$
\begin{aligned}
\delta_{tj} &= \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \\
&= \left(\sum_k \frac{\partial E_t}{\partial \xi_{tk}} \frac{\partial \xi_{tk}}{\partial y_{tj}}\right) f'(\xi_{tj}) \\
&= \left(\sum_k \frac{\partial E_t}{\partial \xi_k} w_{jk}\right) f'(\xi_{tj}) \\
&= \left(\sum_k \delta_{tk} w_{jk}\right) f'(\xi_{tj})
\end{aligned}
$$

(index k jde přes všechny neurony v následující vrstvě)

Back-propagation - Adaptační pravidla

Celkem:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \delta_{tj} y_{ti},$$

kde

- pro výstupní neuron j:

$$\delta_{tj} = f'(\xi_{tj})(y_{tj} - d_{tj}).$$

- pro skrytý neuron j:

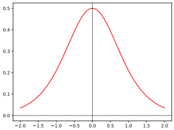
$$\delta_{tj} = f'(\xi_{tj}) \sum_k (\delta_{tk} w_{jk}).$$

Připomenutí: Výpočet derivace přenosové funkce

Sigmoidální ... logsig

$$y = f(\xi) = \frac{1}{1 + e^{-\lambda\xi}}$$

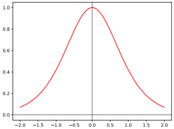
$$f'(\xi) = \lambda y(1 - y)$$



Hyperbolický tangens ... tanh

$$y = f(\xi) = \frac{1 - e^{-2\lambda\xi}}{1 + e^{-2\lambda\xi}}$$

$$f'(\xi) = \lambda^2(1 + y)(1 - y)$$



Algoritmus zpětného šíření (online varianta)

1. **Inicializace sítě**
   Inicializujeme váhy a prahy malými náhodnými hodnotami
2. **Předložíme další trénovací vzor** ve tvaru $(\vec{x}_t, \vec{d}_t)$
3. **Dopředný výpočet:**
   Postupujeme ve směru od vstupní vrstvy k výstupní a pro každý neuron j spočteme (a zapamatujeme si) jeho výstup y_{tj} (využijeme k tomu aktivity neuronů v předchozí vrstvě, včetně fiktivního)
   $$y_{tj} = f(\xi_{tj}) = f(\sum_i w_{ij} y_{ti})$$
   (i je index přes neurony ve vrstvě předcházející neuronu j)

Algoritmus zpětného šíření (online varianta)

4. **Zpětný výpočet**
Spočteme chybový člen δ_{tj} pro každý neuron j a aktualizujeme všechny váhy v síti (ve směru od výstupní vrstvy k vstupní):
    - Pro výstupní neurony j spočteme:

$$\delta_{tj} = f'(\xi_{tj})(y_{tj} - d_{tj})$$

    - Pro skryté neurony j spočteme:

$$\delta_{tj} = f'(\xi_{tj}) \sum_{k} (\delta_{tk} w_{jk}).$$

      (k je index přes neurony ve vrstvě následující po vrstvě obsahující neuron j)
    - Pro váhu každé hrany (z nějakého neuronu i včetně fiktivního) do neuronu j:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \delta_{tj} y_{ti}$$

Algoritmus zpětného šíření (online varianta)

5. **Ukončující podmínka**
   Pokud není splněna ukončující podmínka, vrátíme se zpět ke kroku 2.
   - Předem daný maximální počet epoch.
   - Časový limit.
   - Jakmile je průměrná chyba na trénovací množině dostatečně malá ... $E < E_{min}$
   - Jakmile přestane klesat průměrná chyba na validační množině dat .... early stopping
   - Jakmile je přírůstek vah Δw moc malý ... $|\Delta w| < \Delta_{min}$