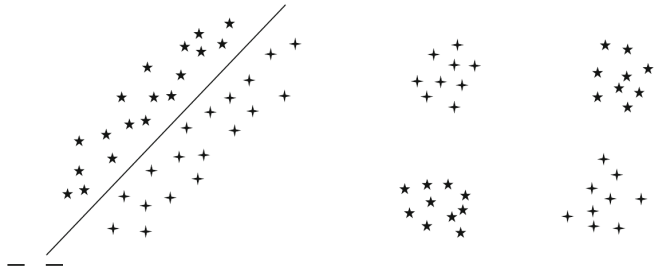
Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Lineární separabilita

- ukázali jsme si základní lineární klasifikační modely
- jejich výrazná nevýhoda je v tom, že umí pracovat jen s lineárně separabilními daty

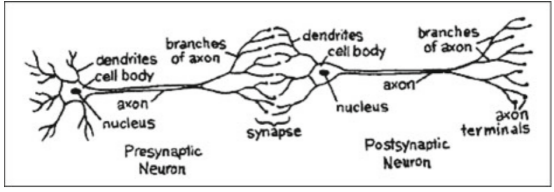


Lineární separabilita

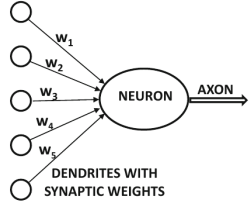
- to se dá někdy řešit vhodnou nelineární transformací dat
- najít takovou transformaci je ale často velice náročné
- tento problém efektivně vyřešily hluboké neuronové sítě
- v následující části přednášky se budeme zabývat právě těmito sítěmi
- nejprve se ale vrátíme k základnímu perceptronu

Perceptron

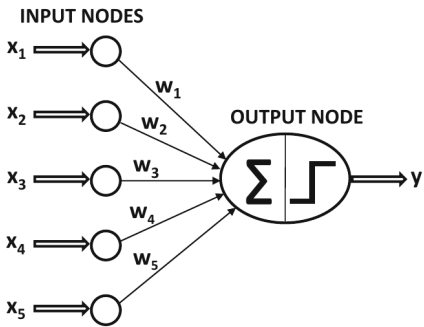
- perceptron je inspirován biologickými neurony



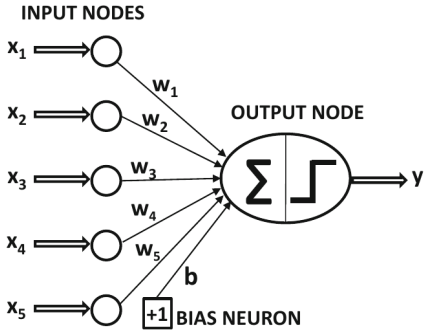
(a) Biological neural network



(b) Artificial neural network

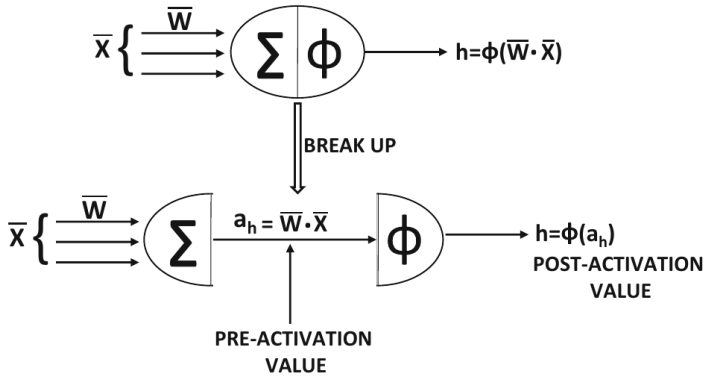


(a) Perceptron without bias



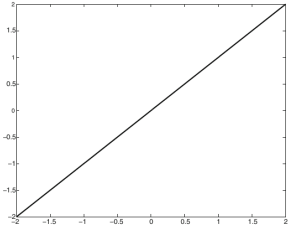
(b) Perceptron with bias

Perceptron

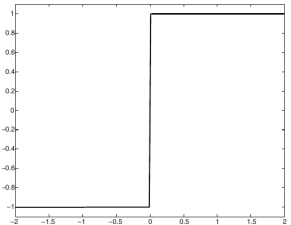


- důležitým prvkem perceptronu je aktivační funkce φ

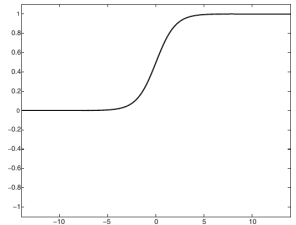
Perceptron



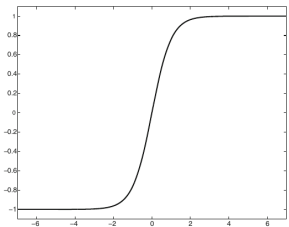
(a) Identity



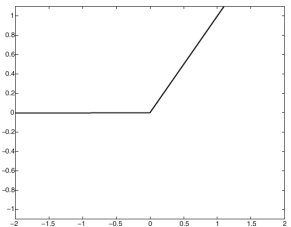
(b) Sign



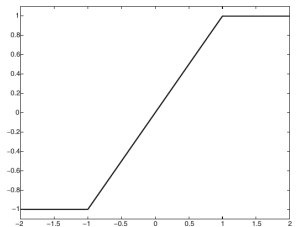
(c) Sigmoid



(d) Tanh



(e) ReLU



(f) Hard Tanh

- sigmoid

$$\varphi(v) = \frac{1}{1+e^{-v}}, \quad \varphi'(v) = \frac{e^{-v}}{(1+e^{-v})^2} = \varphi(v)\left(1-\varphi(v)\right)$$

- tanh

$$\varphi(v) = \frac{e^{2v}-1}{e^{2v}+1} = 2\cdot \text{sigmoid}(2v) - 1, \quad \varphi'(v) = \frac{4e^{2v}}{\left(e^{2v}+1\right)^2} = 1-\varphi(v)^2$$

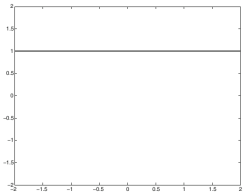
- ReLU = Rectified Linear Unit

$$\varphi(v) = \max\{v, 0\}$$

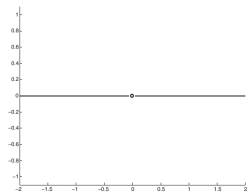
- Hard Tanh

$$\varphi(v) = \max\{\min[v, 1], -1\}$$

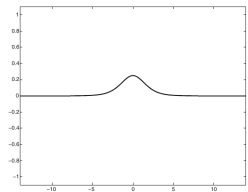
Perceptron



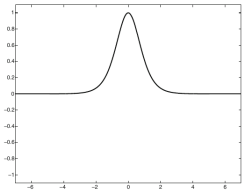
(a) Identity



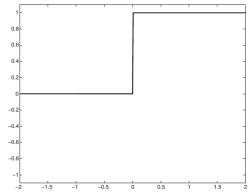
(b) Sign



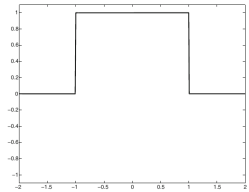
(c) Sigmoid



(d) Tanh



(e) ReLU



(f) Hard Tanh

- derivace aktivačních funkcí

- sigmoid

$$\varphi(v) = \frac{1}{1 + e^{-v}}$$

- tanh

$$\varphi(v) = \frac{e^{2v} - 1}{e^{2v} + 1} = 2 \cdot \text{sigmoid}(2v) - 1$$

- ReLU = Rectified Linear Unit

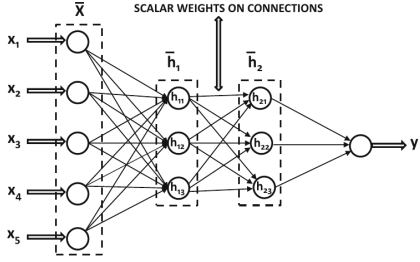
$$\varphi(v) = \max\{v, 0\}$$

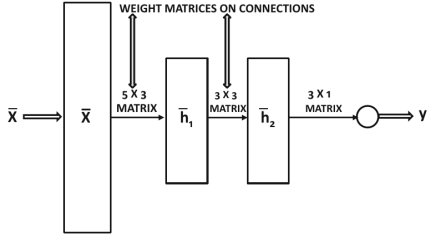
- Hard Tanh

$$\varphi(v) = \max\{\min[v, 1], -1\}$$

Vícevrstvé neuronové sítě

- vícevrstvé neuronové sítě se skládají z několika vrstev
- jednotlivé vrstvy jsou tvořeny perceptrony s různým počtem vstupů a výstupů
  - zde si můžeme všimnout podobnosti s perceptron pro klasifikaci do více tříd





Vícevrstvé neuronové sítě

- data vstupují do sítě přes **vstupní vrstvu** (*input layer*) ve formě vektoru $\vec{x}$
- po vstupní vrstvě následují tzv. **skryté**/**vnitřní vrstvy** (*hidden layers*) a nakonec je **výstupní vrstva** (*output layer*)
- data se nemohou vracet na předchozí vrstvu - proto se těmto sítím také říká *feed forward networks*
- uvažujme neuronovou síť s l vnitřními vrstvami
- označme n_i počet uzlů na i-té vrstvě, vstupní vrstvu označíme indexem 0

Vícevrstvé neuronové sítě

- pro první vrstvu dostáváme

$$\vec{h}_1 = \Phi_1(\mathbb{W}_1\vec{x}), \text{ kde } \vec{x} \in \mathbb{R}^{n_0}, \quad \mathbb{W} \in \mathbb{R}^{n_1,n_0}, \quad \vec{h}_1 \in \mathbb{R}^{n_1}$$

- pro další vrstvy pak platí

$$\vec{h}_{p+1} = \Phi_{p+1}\left(\mathbb{W}_{p+1}\vec{h}_p\right) \text{ pro } p = 1,\ldots,k, \text{ kde } \vec{h}_p \in \mathbb{R}^{n_p}, \quad \mathbb{W}_p \in \mathbb{R}^{n_p,n_{p-1}}$$

- pro výstupní vrstvu platí

$$\vec{o} = \Phi_{l+1}\left(\mathbb{W}_{l+1}\vec{h}_l\right), \text{ kde } \vec{y} \in \mathbb{R}^{l+1}, \mathbb{W}_{l+1} \in \mathbb{R}^{l+1,l}$$

- pokud by platilo, že Φ_i je identita pro $i = 1, \ldots, l+1$, tj. nemáme žádné nelinearity, pak je

$$\vec{o} = \mathbb{W}_{l+1}\mathbb{W}_l \ldots \mathbb{W}_1 \vec{x} = \mathbb{W}^* \vec{x}$$

pro

$$\mathbb{W}^* = \mathbb{W}_{l+1}\mathbb{W}_l \ldots \mathbb{W}_1$$

a celou síť lze vyjádřit pomocí jedné vrstvy

Vícevrstvé neuronové sítě

- **nelineární** aktivační funkce Φ_p jsou zpravidla stejné v rámci jedné vrstvy, často i v celé síti
- učení sítě spočívá ve správném nastavení váhových matic $\mathbb{W}_1, \ldots \mathbb{W}_{l+1}$
- to se provádí opět pomocí minimalizace vhodné ztrátové funkce a pomoci některé gradientní metody jako SGD
- pokud řešíme úlohu s numerickými výstupy typu regrese, lze použít jako ztrátovou funkci L_1/L_2-normu

- pokud řešíme klasifikační úlohu, volíme často na výstupní vrstvě *softmax* aktivační funkci

$$o_i = \frac{e^{v_i}}{\sum_{j=1}^{k} e^{v_j}}, \text{ pro } i = 1, \ldots, k$$

- takto se převádí vstup $(v_1, \ldots, v_k) \in \mathbb{R}^n$ na pravděpodobnosti $o_1, \ldots, o_k$
- softmax se s oblibou kombinuje s tzv. *cross-entropy* jako ztrátovou funkcí

$$L = -\sum_{i=1}^{k} y_i \log o_i$$

kde $y_i \in \{0, 1\}$ *one-hot* kódování pozorovaných tříd $C_1, \ldots, C_k$

- potom je, jak již víme

$$\frac{\partial L}{\partial v_i} = \sum_{i=1}^{k} \frac{\partial L}{\partial o_j} \frac{\partial o_j}{\partial v_i} = o_i - y_i$$

Example 1

Tensorflow Playground

Učení neuronových sítí

- učení spočívá v nalezení váhových matic $\mathbb{W}_i$ ve snaze minimalizovat ztrátovou funkci
- na rozdíl od lineárních klasifikátorů, nyní již nehledáme minimum konvexní úlohy
- téměř vždy se musíme spokojit s nalezením dostatečně dobrého lokálního minima
- funkce, které optimalizujeme při učení neuronových sítí bývají "velice divoké"
- k nalezení lokálního minima můžeme použít různé metody, téměř vždy jsou ale založeny na výpočtu gradientu vůči vahám neuronové sítě
- v následující části si ukážeme, jak tento gradient napočítat

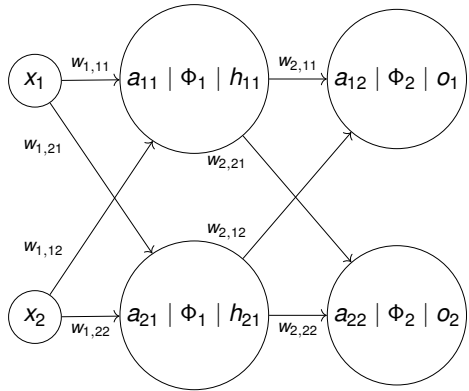
Učení neuronových sítí

Máme ztrátovou funkci $L = L(o_1, o_2)$ a

$$
\begin{aligned}
o_1 &= \phi_2(\underbrace{w_{2,11} h_{11} + w_{2,12} h_{21}}_{a_{12}}) \\
o_1 &= \phi_2(\underbrace{w_{2,21} h_{11} + w_{2,22} h_{21}}_{a_{22}}) \\
h_{11} &= \phi_1(\underbrace{w_{1,11} x_1 + w_{1,12} x_2}_{a_{11}}) \\
h_{21} &= \phi_1(\underbrace{w_{1,21} x_1 + w_{1,22} x_2}_{a_{21}})
\end{aligned}
$$

a tedy

$$
\begin{aligned}
o_1 &= \phi_2\left(w_{2,11}\phi_1\left(w_{1,11}x_1 + w_{1,12}x_2\right)\right. \\
&\quad \left. + w_{2,12}\phi_1\left(w_{1,21}x_1 + w_{1,22}x_2\right)\right) \\
o_2 &= \phi_2\left(w_{2,21}\phi_1\left(w_{1,11}x_1 + w_{1,12}x_2\right)\right. \\
&\quad \left. + w_{2,22}\phi_1\left(w_{1,21}x_1 + w_{1,22}x_2\right)\right)
\end{aligned}
$$



Učení neuronových sítí

- nyní chceme napočítat parciální derivace ztrátové funkce L vůči jednotlivým vahám $w_{*,**}$, tj.

$$\frac{\partial L}{\partial w_{*,**}} = \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial w_{*,**}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial w_{*,**}}$$

$$o_1 = \phi_2\left(w_{2,11}\phi_1(\underbrace{w_{1,11}x_1 + w_{1,12}x_2}_{a_{11}}) + w_{2,12}\phi_1(\underbrace{w_{1,21}x_1 + w_{1,22}x_2}_{a_{21}})\right)$$

$$\begin{aligned}
\frac{\partial o_1}{\partial w_{2,11}} &= \phi_2'(a_{12})\phi_1(a_{11}) \\
\frac{\partial o_1}{\partial w_{2,12}} &= \phi_2'(a_{12})\phi_1(a_{21}) \\
\frac{\partial o_1}{\partial w_{1,11}} &= \phi_2'(a_{12})w_{2,11}\phi_1'(a_{11})x_1 \\
\frac{\partial o_1}{\partial w_{1,12}} &= \phi_2'(a_{12})w_{2,11}\phi_1'(a_{11})x_2 \\
\frac{\partial o_1}{\partial w_{1,21}} &= \phi_2'(a_{12})w_{2,12}\phi_1'(a_{21})x_1 \\
\frac{\partial o_1}{\partial w_{1,22}} &= \phi_2'(a_{12})w_{2,12}\phi_1'(a_{21})x_2
\end{aligned}$$

- podobně pro o_2

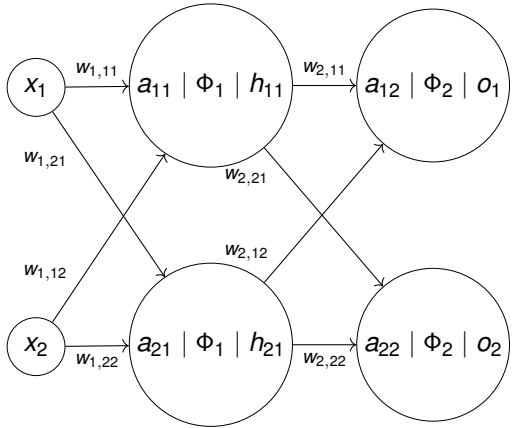
$$o_2 = \phi_2\left(w_{2,21}\phi_1(\underbrace{w_{1,11}x_1 + w_{1,12}x_2}_{a_{11}}) + w_{2,22}\phi_1(\underbrace{w_{1,21}x_1 + w_{1,22}x_2}_{a_{21}})\right)$$

$$\begin{aligned}
\frac{\partial o_2}{\partial w_{2,11}} &= \phi_2'(a_{22})\phi_1(a_{11}) \\
\frac{\partial o_2}{\partial w_{2,12}} &= \phi_2'(a_{22})\phi_1(a_{21}) \\
\frac{\partial o_2}{\partial w_{1,11}} &= \phi_2'(a_{22})w_{2,21}\phi_1'(a_{11})x_1 \\
\frac{\partial o_2}{\partial w_{1,12}} &= \phi_2'(a_{22})w_{2,21}\phi_1'(a_{11})x_2 \\
\frac{\partial o_2}{\partial w_{1,21}} &= \phi_2'(a_{22})w_{2,22}\phi_1'(a_{21})x_1 \\
\frac{\partial o_2}{\partial w_{1,22}} &= \phi_2'(a_{22})w_{2,22}\phi_1'(a_{21})x_2
\end{aligned}$$

Učení neuronových sítí

Nyní např. dostáváme

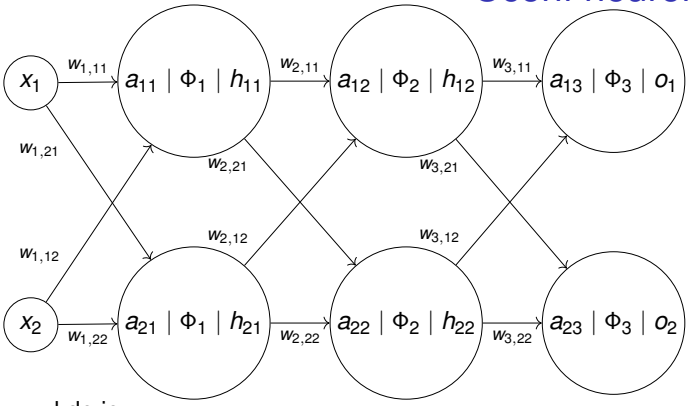
$$\begin{aligned}
\frac{\partial L}{\partial w_{1,11}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial w_{1,11}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial w_{1,11}} \\
&= \frac{\partial L}{\partial o_1}\phi_2'(a_{12})w_{2,11}\phi_1'(a_{11})x_1 + \\
&\quad \frac{\partial L}{\partial o_2}\phi_2'(a_{22})w_{2,21}\phi_1'(a_{11})x_1 \\
\frac{\partial L}{\partial w_{1,22}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial w_{1,22}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial w_{1,22}} \\
&= \frac{\partial L}{\partial o_1}\phi_2'(a_{12})w_{2,12}\phi_1'(a_{21})x_2 + \\
&\quad \frac{\partial L}{\partial o_2}\phi_2'(a_{22})w_{2,22}\phi_1'(a_{21})x_2
\end{aligned}$$



Učení neuronových sítí

- vidíme, že derivace jsou dány **všemi cestami v grafu** vedoucími k hraně s danou vahou
- zejména v sítích s více vrstvami by byl výpočet gradientu velmi náročný
- s rostoucím počtem vrstev totiž roste exponenciálně počet výrazů, které je potřeba vyčíslit
- odvodíme efektivnější algoritmus založený na principu *dynamického programování*
- tomuto algoritmu se říká **zpětná propagace**, anglicky *backpropagation*
- předvedeme si ho na příkladu větší neuronové sítě

Učení neuronových sítí - aktivace



$$
\begin{aligned}
a_{11} &= w_{1,11}x_1 + w_{1,12}x_2, \\
a_{21} &= w_{1,21}x_1 + w_{1,22}x_2, \\
a_{12} &= w_{2,11}h_{11} + w_{2,12}h_{21}, \\
a_{22} &= w_{2,21}h_{11} + w_{2,22}h_{21}, \\
a_{13} &= w_{3,11}h_{12} + w_{3,12}h_{22}, \\
a_{23} &= w_{3,21}h_{12} + w_{3,22}h_{22},
\end{aligned}
$$

kde je

$$
h_{11} = \Phi_1(a_{11}), \quad h_{21} = \Phi_1(a_{21}), \quad h_{12} = \Phi_2(a_{12}), \quad h_{22} = \Phi_2(a_{22}),
$$
$$
o_1 = \Phi_3(a_{13}), \quad o_2 = \Phi_3(a_{23})
$$

Učení neuronových sítí - zpětná propagace

- nejprve provedeme aktivaci sítě, tj. napočítání hodnot a_{ij}
- všimneme si, že aktivace probíhá zleva doprava
- zpětná propagace probíhá v opačném směru, tj. od poslední vrstvy
- využijeme při ní právě napočítané aktivační hodnoty

Učení neuronových sítí - zpětná propagace

Pro gradienty vah na poslední vrstvě dostáváme:

$$\frac{\partial L(o_1,o_2)}{\partial w_{3,11}} = \frac{\partial L(o_1,o_2)}{\partial o_1}\frac{\partial o_1}{\partial w_{3,11}} + \frac{\partial L(o_1,o_2)}{\partial o_1}\underbrace{\frac{\partial o_1}{\partial w_{3,11}}}_{=0} = \frac{\partial L(o_1,o_2)}{\partial o_1}\frac{\partial \Phi_3(a_{13})}{\partial w_{3,11}} = \frac{\partial L(o_1,o_2)}{\partial o_1}\Phi_3'(a_{13})h_{12},$$

$$\frac{\partial L(o_1,o_2)}{\partial w_{3,21}} = \frac{\partial L(o_1,o_2)}{\partial o_1}\underbrace{\frac{\partial o_1}{\partial w_{3,21}}}_{=0} + \frac{\partial L(o_1,o_2)}{\partial o_2}\frac{\partial o_2}{\partial w_{3,21}} = \frac{\partial L(o_1,o_2)}{\partial o_2}\frac{\partial \Phi_3(a_{23})}{\partial w_{3,21}} = \frac{\partial L(o_1,o_2)}{\partial o_2}\Phi_3'(a_{23})h_{12},$$

$$\frac{\partial L(o_1,o_2)}{\partial w_{3,12}} = \frac{\partial L(o_1,o_2)}{\partial o_1}\frac{\partial o_1}{\partial w_{3,12}} + \frac{\partial L(o_1,o_2)}{\partial o_1}\underbrace{\frac{\partial o_1}{\partial w_{3,12}}}_{=0} = \frac{\partial L(o_1,o_2)}{\partial o_1}\frac{\partial \Phi_3(a_{13})}{\partial w_{3,12}} = \frac{\partial L(o_1,o_2)}{\partial o_1}\Phi_3'(a_{13})h_{22},$$

$$\frac{\partial L(o_1,o_2)}{\partial w_{3,22}} = \frac{\partial L(o_1,o_2)}{\partial o_1}\underbrace{\frac{\partial o_1}{\partial w_{3,22}}}_{=0} + \frac{\partial L(o_1,o_2)}{\partial o_2}\frac{\partial o_2}{\partial w_{3,22}} = \frac{\partial L(o_1,o_2)}{\partial o_2}\frac{\partial \Phi_3(a_{23})}{\partial w_{3,22}} = \frac{\partial L(o_1,o_2)}{\partial o_2}\Phi_3'(a_{23})h_{22},$$

Učení neuronových sítí - zpětná propagace

Zavedeme substituce:

$$b_{13} = \frac{\partial L}{\partial o_1}\Phi_3'(a_{13}), \quad b_{23} = \frac{\partial L}{\partial o_2}\Phi_3'(a_{23})$$

Pak lze psát:

$$\begin{aligned}
\frac{\partial L}{\partial w_{3,11}} &= b_{13}h_{12}, \\
\frac{\partial L}{\partial w_{3,21}} &= b_{23}h_{12}, \\
\frac{\partial L}{\partial w_{3,12}} &= b_{13}h_{22}, \\
\frac{\partial L}{\partial w_{3,22}} &= b_{23}h_{22},
\end{aligned}$$

Učení neuronových sítí - zpětná propagace

- pro gradienty vah na předposlední vrstvě dostáváme

$$\begin{aligned}
\frac{\partial L}{\partial w_{2,11}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial h_{12}}\frac{\partial h_{12}}{\partial w_{2,11}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial h_{12}}\frac{\partial h_{12}}{\partial w_{2,11}} \\
&= \underbrace{\left(\frac{\partial L}{\partial o_1}\frac{\partial \Phi_3(a_{13})}{\partial h_{12}} + \frac{\partial L}{\partial o_2}\frac{\partial \Phi_3(a_{23})}{\partial h_{12}}\right)}_{\frac{\partial L}{\partial h_{12}}}\frac{\partial h_{12}}{\partial w_{2,11}} \\
&= \underbrace{\left(\frac{\partial L}{\partial o_1}\Phi_3'(a_{13})w_{3,11} + \frac{\partial L}{\partial o_2}\Phi_3'(a_{23})w_{3,21}\right)}_{\frac{\partial L}{\partial h_{12}}}\Phi_2'(a_{12})h_{11},
\end{aligned}$$

Učení neuronových sítí - zpětná propagace

- celkem tedy máme

$$\frac{\partial L}{\partial w_{2,11}} = \underbrace{\left(\frac{\partial L}{\partial o_1}\Phi_3'(a_{13})w_{3,11} + \frac{\partial L}{\partial o_2}\Phi_3'(a_{23})w_{3,21}\right)}_{\frac{\partial L}{\partial h_{12}}}\Phi_2'(a_{12})h_{11},$$

$$\frac{\partial L}{\partial w_{2,12}} = \underbrace{\left(\frac{\partial L}{\partial o_1}\Phi_3'(a_{13})w_{3,11} + \frac{\partial L}{\partial o_2}\Phi_3'(a_{23})w_{3,21}\right)}_{\frac{\partial L}{\partial h_{12}}}\Phi_2'(a_{12})h_{21},$$

$$\frac{\partial L}{\partial w_{2,21}} = \underbrace{\left(\frac{\partial L}{\partial o_1}\Phi_3'(a_{13})w_{3,12} + \frac{\partial L}{\partial o_2}\Phi_3'(a_{23})w_{3,22}\right)}_{\frac{\partial L}{\partial h_{22}}}\Phi_2'(a_{22})h_{11},$$

$$\frac{\partial L}{\partial w_{2,22}} = \underbrace{\left(\frac{\partial L}{\partial o_1}\Phi_3'(a_{13})w_{3,12} + \frac{\partial L}{\partial o_2}\Phi_3'(a_{23})w_{3,22}\right)}_{\frac{\partial L}{\partial h_{22}}}\Phi_2'(a_{22})h_{21},$$

Učení neuronových sítí - zpětná propagace

- to lze psát jako

$$\frac{\partial L}{\partial w_{2,11}} = b_{12} h_{11}, \quad \frac{\partial L}{\partial w_{2,12}} = b_{12} h_{21}, \quad \frac{\partial L}{\partial w_{2,21}} = b_{22} h_{11}, \quad \frac{\partial L}{\partial w_{2,22}} = b_{22} h_{21},$$

- pro

$$\begin{aligned} b_{12} = \frac{\partial L}{\partial h_{12}} &= \frac{\partial L}{\partial o_1} \Phi_3'(a_{31}) w_{3,11} \Phi_2'(a_{12}) + \frac{\partial L}{\partial o_2} \Phi_3'(a_{32}) w_{3,21} \Phi_2'(a_{12}) \\ &= \Phi_2'(a_{12}) \left(b_{13} w_{3,11} + b_{23} w_{3,21} \right), \\ b_{22} = \frac{\partial L}{\partial h_{22}} &= \frac{\partial L}{\partial o_1} \Phi_3'(a_{31}) w_{3,12} \Phi_2'(a_{22}) + \frac{\partial L}{\partial o_2} \Phi_3'(a_{32}) w_{3,22} \Phi_2'(a_{22}) \\ &= \Phi_2'(a_{22}) \left(b_{13} w_{3,12} + b_{23} w_{3,22} \right) \end{aligned}$$

Učení neuronových sítí - zpětná propagace

- ve výrazech pro b_{12} a b_{22} se vyskytují vztahy

$$b_{13}w_{3,11} + b_{23}w_{3,21}$$
$$b_{13}w_{3,12} + b_{23}w_{3,22}$$

- označíme-li

$$\begin{pmatrix} b^*_{12} \\ b^*_{22} \end{pmatrix} = \begin{pmatrix} b_{13}w_{3,11} + b_{23}w_{3,21} \\ b_{13}w_{3,12} + b_{23}w_{3,22} \end{pmatrix} = \mathbb{W}_3^T \vec{b}_3,$$

- pak lze psát

$$b_{12} = \Phi'_2(a_{12})b^*_{12},$$
$$b_{22} = \Phi'_2(a_{22})b^*_{22}.$$

- při zpětné propagaci tedy násobíme **transponovanými** maticemi vah

Učení neuronových sítí - zpětná propagace

- pro gradienty vah na první vrstvě

$$\begin{aligned}
\frac{\partial L}{\partial w_{1,11}} &= \frac{\partial L}{\partial h_{12}}\frac{\partial h_{12}}{\partial w_{1,11}} + \frac{\partial L}{\partial h_{22}}\frac{\partial h_{22}}{\partial w_{1,11}} = \frac{\partial L}{\partial h_{12}}\frac{\partial h_{12}}{\partial h_{11}}\frac{\partial h_{11}}{\partial w_{1,11}} + \frac{\partial L}{\partial h_{22}}\frac{\partial h_{22}}{\partial h_{11}}\frac{\partial h_{11}}{\partial w_{1,11}} \\
&= \underbrace{\left(\frac{\partial L}{\partial h_{12}}\frac{\partial h_{12}}{\partial h_{11}} + \frac{\partial L}{\partial h_{22}}\frac{\partial h_{22}}{\partial h_{11}}\right)}_{\frac{\partial L}{\partial h_{11}}}\frac{\partial h_{11}}{\partial w_{1,11}} \\
&= \left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,11} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,21}\right)\Phi_1'(a_{11})x_1 \\
\frac{\partial L}{\partial w_{1,21}} &= \underbrace{\left(\frac{\partial L}{\partial h_{12}}\frac{\partial h_{12}}{\partial h_{21}} + \frac{\partial L}{\partial h_{22}}\frac{\partial h_{22}}{\partial h_{21}}\right)}_{\frac{\partial L}{\partial h_{21}}}\frac{\partial h_{21}}{\partial w_{1,21}} \\
&= \left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,12} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,22}\right)\Phi_1'(a_{21})x_1
\end{aligned}$$

Učení neuronových sítí - zpětná propagace

- celkem pak máme

$$\frac{\partial L}{\partial w_{1,11}} = \underbrace{\left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,11} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,21}\right)}_{\frac{\partial L}{\partial h_{11}}}\Phi_1'(a_{11})x_1,$$

$$\frac{\partial L}{\partial w_{1,12}} = \underbrace{\left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,11} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,21}\right)}_{\frac{\partial L}{\partial h_{11}}}\Phi_1'(a_{11})x_2,$$

$$\frac{\partial L}{\partial w_{1,21}} = \underbrace{\left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,12} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,22}\right)}_{\frac{\partial L}{\partial h_{21}}}\Phi_1'(a_{21})x_1,$$

$$\frac{\partial L}{\partial w_{1,22}} = \underbrace{\left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,12} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,22}\right)}_{\frac{\partial L}{\partial h_{21}}}\Phi_1'(a_{21})x_2$$

Učení neuronových sítí - zpětná propagace

- zavedeme opět značení

$$\begin{aligned} b_{11} &= \Phi_1'(a_{11})\left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,11} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,21}\right) \\ &= \Phi_1'(a_{11})\left(b_{12}w_{2,11} + b_{22}w_{2,21}\right), \\ b_{21} &= \Phi'(a_{21})\left(\frac{\partial L}{\partial h_{12}}\Phi_2'(a_{12})w_{2,12} + \frac{\partial L}{\partial h_{22}}\Phi_2'(a_{22})w_{2,22}\right) \\ &= \Phi_1'(a_{21})\left(b_{12}w_{2,12} + b_{22}w_{2,22}\right) \end{aligned}$$

- a lze psát

$$\frac{\partial L}{\partial w_{1,11}} = b_{11}x_1, \frac{\partial L}{\partial w_{1,12}} = b_{11}x_2, \frac{\partial L}{\partial w_{1,21}} = b_{21}x_1, \frac{\partial L}{\partial w_{1,22}} = b_{21}x_2,$$

Učení neuronových sítí - zpětná propagace

- mějme nyní obecnou síť s k vrstvami
- síť má n_0 vstupů, na j-té vrstvě n_j neuronů a má n_{k+1} výstupů
- na j-té vrstvě jsou použity nelinearity Φ_j
- aktivaci pak lze popsat takto

procedure AKTIVACE($\vec{x}$)
  $\vec{a}_0 = \vec{h}_0 = \vec{x}$
  for $j = 1, \ldots, k$ **do**
    $\vec{a}_j := \mathbb{W}_j \vec{h}_{j-1}$
    for $i = 1, \ldots, n_j$ **do**
      $h_{i,j} = \Phi_j(a_{i,j})$
    end for
  end for
  $\vec{o} = \vec{h}_k$
end procedure

Učení neuronových sítí - zpětná propagace

- zpětnou propagaci pak lze zapsat takto

procedure BACKPROPAGATION
  for $i = 1, \dots n_k$ **do**
    $b_{i,k} = \Phi'_k(a_{i,k}) \frac{\partial L}{\partial o_i}$
    for $j = 1, \dots n_{k-1}$ **do**
      $\frac{\partial L}{\partial w_{k,ij}} = b_{i,k} h_{i,k-1}$
    end for
  end for
  for $l = k-1, \dots, 1$ **do**
    $\vec{b}^*_l = \mathbb{W}^T_{l+1} \vec{b}_{.,l+1}$
    for $i = 1, \dots, n_l$ **do**
      $b_{il} = \Phi'_l(a_{il}) b^* il$
      for $j = 1, \dots, n_{l-1}$ **do**
        $\frac{\partial L}{\partial w_{l,ji}} = b_{jl} h_{i,l-1}$
      end for
    end for
  end for
end procedure

Implementace NN

Implementaci plně propojené NS lze najít zde (autor J. Kovář).

```
git clone TODO
```

TODO - doplnit

Mizející a explodující gradient

- předpokládejme neuronovou síť o $l + 1$ vrstvách, každá vrstva mé jeden neuron
- váhy označíme $w_1, \ldots w_l$
- na každé vrstvě používáme nelinearitu Φ_i

- potom

$$\frac{\partial L}{\partial h_t} = \phi'(h_t) w_{t+1} \frac{\partial L}{\partial h_{t+1}}$$

- nebo-li

$$\frac{\partial L}{\partial h_1} = \prod_{i=1}^{l} \phi'(h_i) w_{i+1} \Rightarrow \frac{\partial L}{\partial w_1} = x\Phi'(x) \frac{\partial L}{\partial h_1} = x\Phi'(x) \prod_{i=1}^{l} \phi'(h_i) w_{i+1}$$

Mizející a explodující gradient

- vidíme, že výpočet gradientu obsahuje produkt o délce rovné počtu vrstev
- tedy

$$\frac{\partial L}{\partial w_1} \approx \phi'(\cdot)^{l+1}$$

- bude-li $|\phi'(h_i)| \ll 1$, pak produkt exponenciálně rychle konverguje k nule = ***vanishing gradient***
- bude-li $|\phi'(h_i)| \gg 1$, pak produkt exponenciálně rychle diverguje = ***exploding gradient***

Mizející a explodující gradient

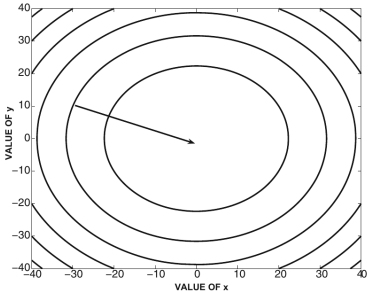
- nelinearity pro NN jsou navrženy tak, aby se tento problém omezily
  - *ReLU* a *hard tanh* jsou oblíbenější než *sigmoid* a *tanh*, jejichž derivace jsou malé
- je také důležité rozumě inicializovat váhy před učením, většinou pomocí gaussovského náhodného rozdělení s malý, rozptylem a středem v nule
- přesto tento efekt nelze úplně eliminovat a způsobuje to problém s konvergencí GD

- uvažujme ztrátovou funkci tvaru

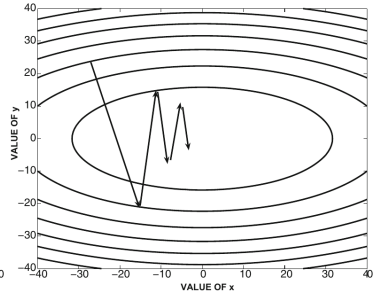
$$L(x, y) = \frac{1}{2}(\alpha x^2 + \beta y^2) \Rightarrow \nabla L = \begin{pmatrix} \alpha x \\ \beta y \end{pmatrix}$$

- pro $\alpha \ll \beta$ (nebo naopak) má GD tendenci sledovat "cik cak" stopu
- ve směrech, kde jsou složky gradientu velké, pozorujeme **oscilace**
- ve směrech, kde jsou složky gradientu malé, pozorujeme **velice pomalou konvergenci**

Gradient descent



(a) Loss function is circular bowl
$L = x^2 + y^2$



(b) Loss function is elliptical bowl
$L = x^2 + 4y^2$

Figure: Ch.C.Aggarwal, Neural networks and deep learning

Gradient descent - learning rate decay

- nejjednodušším řešením je zmenšování relaxačního parametru
- na začátku nastavíme větší relaxační parametr
  - urychlíme pohyb ve směrech, kde má gradient malé složky
  - ve směrech velkých gradientních složek se zvýrazní oscilace, ale to nemusí vadit
- v průběhu iterací se relaxační parametr zmenšuje
  - s tím, jak se blížíme lokálnímu minimu, začínají být oscilace problém, a proto je musíme omezit
- relaxační parametr γ_k se v k-té iteraci často volí jako

$$\gamma_k = \gamma_0 e^{(-\alpha k)} \text{ exponential decay },$$
$$\gamma_k = \gamma_0 \frac{1}{1+\alpha k} \text{ inverse decay}$$

- Ch.C.Aggarwal - *"In some cases, the analyst might even babysit the learning process, and change the rate manually."* :)

Momentové metody

- jiným řešením jsou momentové metody (1986)
- upravíme předpis pro GD

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} + \vec{v}^{(k+1)}, \text{ kde } \vec{v}^{(k+1)} = -\gamma \nabla L(\vec{w}^{(k)})$$

- základní momentová metoda má tvar

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} + \vec{v}^{(k)}, \text{ kde } \vec{v}^{(k+1)} = \beta v^{(k)} - \gamma \nabla L(\vec{w}^{(k)}),$$

pro $\beta \in (0, 1)$

Why Momentum Really Works

- snadno vidíme, že

$$\begin{aligned}
\vec{w}^{(1)} &= \vec{w}^{(0)} - \gamma \left(\nabla L(\vec{w}^{(0)}) \right), \\
\vec{w}^{(2)} &= \vec{w}^{(1)} - \gamma \left(\nabla L(\vec{w}^{(1)}) - \beta \nabla L(\vec{w}^{(0)}) \right), \\
\vec{w}^{(3)} &= \vec{w}^{(2)} - \gamma \left(\nabla L(\vec{w}^{(2)}) - \beta \nabla L(\vec{w}^{(1)}) - \beta^2 \nabla L(\vec{w}^{(0)}) \right), \\
&\vdots \\
\vec{w}^{(k+1)} &= \vec{w}^{(k)} - \gamma \sum_{i=0}^{k} \beta^i \nabla L(\vec{w}^{(k-i)})
\end{aligned}$$

- předpis si tedy pamatuje celou historii kroků a průměruje je s tím, že starší kroky stále více a více "zapomíná"

Momentové metody

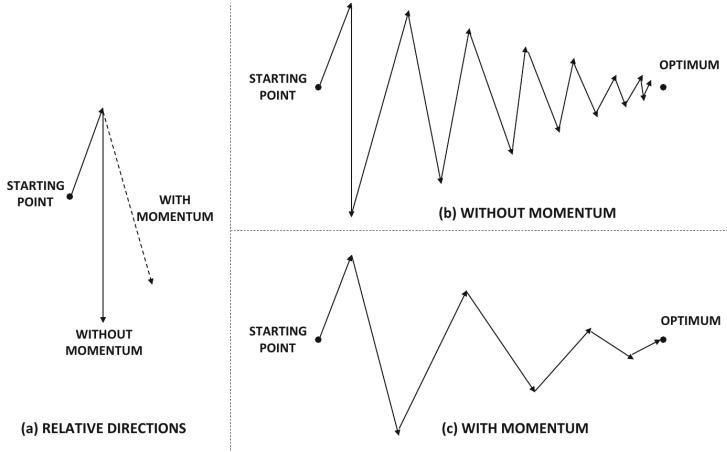


Figure: Ch.C.Aggarwal, Neural networks and deep learning

Momentové metody

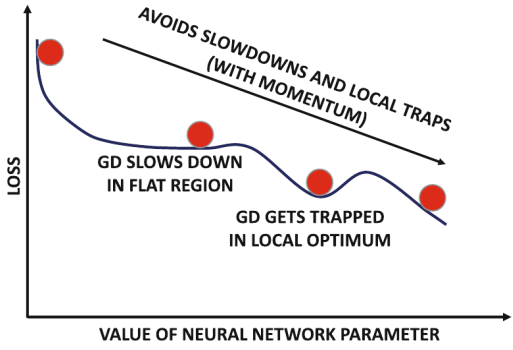


Figure: Ch.C.Aggarwal, Neural networks and deep learning

- metoda vykazuje setrvačnost, která umožňuje navíc překonat mělká lokální minima

Nesterovy momenty

- metoda Nesterových momentů je dána předpisem

$$\begin{aligned} \vec{w}^{(k+1)} &= \vec{w}^{(k)} + \vec{v}^{(k+1)}, \\ \vec{v}^{(k+1)} &= \beta \vec{v}^{(k)} - \gamma \nabla L(\vec{w}^{(k)} + \beta \vec{v}^{(k)}) \end{aligned}$$

- myšlenka je taková, že se "při řízení díváme kousek před sebe"
- nevýrazná lokální minima takto spíše přeskočíme
- u těch výraznějších začneme brzdit předem

Metody parametrické relaxace

- tyto metody sledují jednotlivé složky gradientu ∇L a vhodně je modifikují
- jedna z prvních metod byla *delta-bar-delta*
- sleduje, které složky gradientu mezi iteracemi mění znaménko
- takové složky signalizují oscilace v daném směru
- tyto složky se pak násobí menším relaxačním parametrem

Metody parametrické relaxace - AdaGrad, 2011

- AdaGrad = Adaptive Gradient
- tato metoda nasčítává druhé mocniny složek gradientu ∇L
- výsledek využívá k "normování" jednotlivých složek
- metoda je dána předpisem

$$a_i^{(k+1)} = a_i^{(k)} + \left(\frac{\partial L(\vec{w}^{(k)})}{\partial w_i} \right)^2, \text{ pro } i \in \hat{n}$$

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\gamma}{\sqrt{a_i^{(k+1)}}} \frac{\partial L(\vec{w}^{(k)})}{\partial w_i}, \text{ pro } i \in \hat{n}.$$

Metody parametrické relaxace - RMSProp

- RMSProp = Root Mean Square Propagation
- nepublikovaný algoritmus zmíněný G. Hintonem, 2018
- metoda RMSProp vylepšuje AdaGrad, u kterého jsou a_i monotonně rostoucí
- to způsobuje přílišné zpomalování konvergence v pozdějších iteracích
- RMSProp používá stejný trik jako momentové metody k tomu, aby postupně zapomínal velikosti starších složek
- metoda je dána předpisem

$$
\begin{aligned}
a_i^{(k+1)} &= \rho a_i^{(k)} + (1-\rho)\left(\frac{\partial L(\vec{w}^{(k)})}{\partial w_i}\right)^2, \text{ pro } i \in \hat{n} \\
w_i^{(k+1)} &= w_i^{(k)} - \frac{\gamma}{\sqrt{a_i^{(k+1)}}}\frac{\partial L(\vec{w}^{(k)})}{\partial w_i}, \text{ pro } i \in \hat{n}.
\end{aligned}
$$

Metody parametrické relaxace - RMSProp

- RMSProp lze kombinovat s metodou Nesterových momentů

$$
\begin{aligned}
a_i^{(k+1)} &= \rho a_i^{(k)} + (1-\rho)\left(\frac{\partial L(\vec{w}^{(k)})}{\partial w_i}\right)^2, \text{ pro } i \in \hat{n} \\
v_i^{(k+1)} &= \beta v_i^{(k)} - \frac{\gamma}{\sqrt{a_i^{(k+1)}}} \frac{\partial L(\vec{w}^{(k)} + \vec{v}^{(k)})}{\partial w_i}, \text{ pro } i \in \hat{n}, \\
\vec{w}^{(k+1)} &= \vec{w}^{(k)} + \vec{v}^{(k+1)}
\end{aligned}
$$

Metody parametrické relaxace - Adadelta

- vychází z RMSProp a nabízí navíc automatickou volbu relaxačního parametru
- má tvar

$$
\begin{aligned}
a_i^{(k+1)} &= \rho a_i^{(k)} + (1-\rho)\left(\frac{\partial L(\vec{w}^{(k)})}{\partial w_i}\right)^2, \text{ pro } i \in \hat{n} \\
\gamma^{(k+1)} &= \rho \gamma^{(k)} + (1-\rho)\left\|\Delta \vec{w}^{(k)}\right\|_2^2 \\
w_i^{(k+1)} &= w_i^{(k)} - \underbrace{\sqrt{\frac{\gamma^{k+1}}{a_i^{(k+1)}}}\frac{\partial L(\vec{w}^{(k)})}{\partial w_i}}_{\Delta \vec{w}^{(k)}}, \text{ pro } i \in \hat{n},
\end{aligned}
$$

- výraz $\sqrt{\frac{\gamma^{(k+1)}}{a_i^{(k+1)}}}$ připomíná heuristický náhradu druhé derivace z metod druhého řádu

Metody parametrické relaxace - Adam, 2014

- Adam = Adaptive Moment Estimation - nejpopulárnější metoda

$$
\begin{aligned}
a_i^{(k+1)} &= \rho a_i^{(k)} + (1-\rho)\left(\frac{\partial L(\vec{w}^{(k)})}{\partial w_i}\right)^2, \text{ pro } i \in \hat{n} \\
f_i^{(k+1)} &= \rho_f f_i^{(k)} + (1-\rho_f)\left(\frac{\partial L}{\partial w_i}\right), \text{ pro } i \in \hat{n} \\
\gamma^{(k)} &= \gamma \frac{\sqrt{1-\rho^{(k)}}}{1-\rho_f^{(k)}}, \\
w_i^{(k+1)} &= w_i^{(k)} - \frac{\gamma^{(k+1)}}{a_i^{(k+1)}} f_i^{(k+1)}, \text{ pro } i \in \hat{n},
\end{aligned}
$$

- $\gamma^{(k)} \to \gamma$ poměrně rychle - má to význam v prvních iteracích, kdy špatný odhad $\vec{a}^{(k)}$ a $\vec{f}^{(k)}$ způsobuje výrazný bias
- $\vec{f}$ je momentově korigovaný gradient

Ořezávání gradientu - gradient clipping

- jde o metodu, kdy ořízneme hodně velké a hodně malé složky gradientu
- narozdíl od momentových metod se to ale neděje na základě celé minulé historie gradientů, ale pouze na základě aktuální hodnoty gradientu
- **value based clipping**
  - zvolíme minimální a maximální prah t_{min} a t_{max} a měníme jednotlivé složky gradientu

$$w_i := \min\left(t_{max}, \max\left(t_{min}, w_i\right)\right)$$

- **norm-based clipping**
  - celý gradient normujeme tak, aby v určité normě byl roven 1, např.

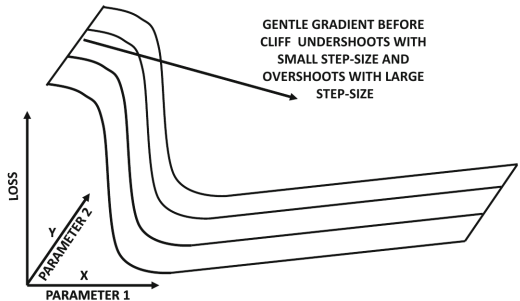
$$\vec{w} := \frac{\vec{w}}{\left\|\vec{w}\right\|_2}$$

- tyto metody jsou obzvlášť efektivní při trénování **rekurentního neuronových sítí**

R. Pascanu, T. Mikolov, Y. Bengio, *On the difficulty of training recurrent neural networks*, Proceedings of the 30th International Conference on Machine Learning, PMLR 28(3):1310-1318, 2013.

Nestability

- doposud prezentované metody prvního řádu mají potíže v situacích, kdy je graf¨ ztrátové funkce výrazně zakřiven
- jde například o tzv. útesy (*clifs*)
- na kraji útesu vede malý krok ve směru gradientu k pomalé konvergenci a velký krok k přeskočení celého údolí pod útesem
- řešením mohou být metody **druhého řádu**



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Metody druhého řádu

- metody druhého řádu jsou založené na výpočtu (nebo aproximaci) druhých parciálních derivací ztrátové funkce
- ty jsou reprezentovány pomocí Hessiánu

$$\mathbb{H}(\vec{w}) = \nabla^2 L(\vec{w})_{ij} = \frac{\partial^2 L(\vec{w})}{\partial w_i \partial w_j}$$

- hledání minima lze převést na řešení soustavy rovnic

$$\nabla L(\vec{w}) = \vec{0}$$

- aplikací Newtonovy metody dostáváme předpis

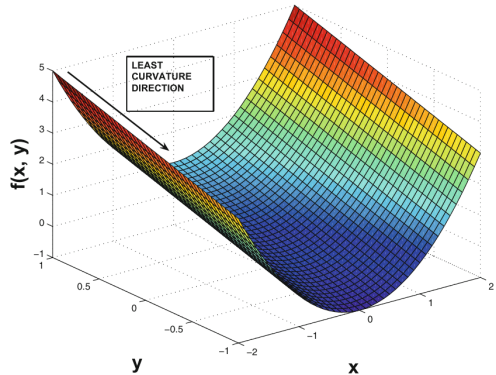
$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \mathbb{H}^{-1}(\vec{w}) \nabla L(\vec{w})$$

Metody druhého řádu

- pokud by L byla kvadratická funkce, pak $\nabla L(\vec{w}) = \vec{0}$ je lineární soustava a řešení najdeme po první iteraci
- Newtonova metoda aproximuje ztrátovou funkci pomocí kvadratické funkce
- tím, že bere v potaz i změnu gradientu (trefuje se do ďolíku kvadratické aproximace) dokáže lépe odhadnout i velikost kroku
- tím v sobě jakoby obsahuje i volbu relaxačního/učícího parametru

Metody druhého řádu

- například v dlouhých úzkých údolích má GD tendenci houpat se z jedné strany na druhou a jen velmi pomalu se pohybuje kupředu - Newtonova metoda dokáže korigovat svůj pohyb i ve směru malé změny gradientu, tj. v podélném směru údolí



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Metody druhého řádu

- velkou nevýhodou této metody je nutnost počítat Hessian ztrátové funkce
- tato matice může být neúnosně veliká pro NN s velkým počtem parametrů - 10^6 např.
- výpočet inverzní matice je pak téměř nemožný
- proto se odvozují metody, které inverzi Hessianu aproximují

J. Martens, I. Sutskever, K. Swersky, *Estimating the Hessian by Back-propagating Curvature*, Proceedings of the 29th International Conference on Machine Learning (ICML 2012).

Metoda konjugovanych gradientu

- jde o metodu prvního řádu, která se ale snaží v každém kroku pohybovat se ve směru ortogonálním k předchozím krokům
- tím tedy také řeší problém GD a navíc nevyžaduje výpočet Hessianu
- metoda má předpis

$$\begin{aligned} \vec{w}^{(k+1)} &= \vec{w}^{(k)} + \alpha^{(k)} \vec{q}^{(k)}, \\ \vec{q}^{(k+1)} &= -\nabla L(\vec{w}^{(k+1)}) + \left(\frac{\vec{q}^{(k)T} \mathbb{H} \nabla L(\vec{w}^{(k+1)})}{\vec{q}^{(k)T} H \vec{q}^{(k)}} \right), \end{aligned}$$

kde $\vec{q}^{(0)} = -\nabla L(\vec{w}^{(k)})$

Metody druhého řádu

- v předpisu se sice Hessian vyskytuje, ale aplikuje se na vektor a nepočítá se jeho inverze
- lze tedy použít aproximaci pomocí konečných diferencí

$$\mathbb{H}\vec{v} \approx \frac{\nabla L(\vec{w} + \delta\vec{v}) - \nabla L(\vec{w})}{\delta}$$

J. Martens, Deep learning via hessian-free optimization, ICML conference, 2010.

BFGS = Broyden-Fletcher-Goldfarb-Shanno

- tato metoda se pokouší aproximovat $\mathbb{H}^{-1}$ maticí $\mathbb{G}^{(k)}$, tj.

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \mathbb{G}^{(k)}(\vec{w})\nabla L(\vec{w})$$

- využívá se vztahu

$$\nabla L(\vec{w}^{(k+1)}) - \nabla_L(\vec{w}^{(k)}) = \nabla^2 L(\vec{\xi})(\vec{w}^{(k+1)} - \vec{w}^{(k)})$$

- a tedy pro $\mathbb{G}^{(k+1)} = \nabla^2 L(\vec{\xi})^{-1}$ platí

$$\vec{w}^{(k+1)} - \vec{w}^{(k)} = \mathbb{G}^{(k+1)} \left(\nabla L(\vec{w}^{(k+1)}) - \nabla_L(\vec{w}^{(k)}) \right)$$

- podstata BFGS metody je v tom, že se snaží napočítat $\mathbb{G}^{(k+1)}$ z předchozího vztahu
- z něj ale matice není určena jednoznačně, proto se doplňuje podmínka na minimalizaci

$$\min \left\| \mathbb{G}^{(k+1)} - \mathbb{G}^{(k)} \right\|_F$$

- to vede na vztah (bez odvození)

$$\mathbb{G}^{(k+1)} = \mathbb{G}^{(k)} + \frac{\vec{v}^{(k)} \vec{v}^{(k)T}}{\vec{v}^{(k)T} \vec{q}^{(k)}} - \frac{\mathbb{G}^{(k)} \vec{q}^{(k)} \vec{q}^{(k)T} \mathbb{G}^{(k)T}}{\vec{q}^{(k)T} \mathbb{G}^{(k)} \vec{q}^{(k)}},$$

kde

$$\vec{q}^{(k)} = \vec{w}^{(k+1)} - \vec{w}^{(k)}, \vec{v}^{(k)} = \nabla L(\vec{w}^{(k+1)}) - \nabla_L(\vec{w}^{(k)})$$

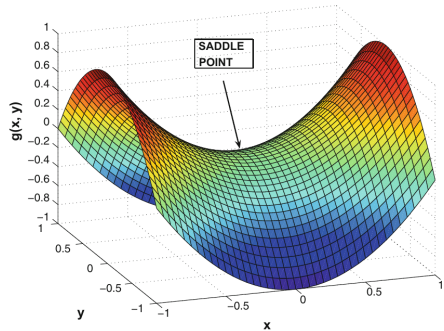
Metody druhého řádu

- BFGS se zbavuje nutnosti výpočtu Hessianu, ale stále pracuje s maticemi stejné velikosti
- nesnižuje tedy paměťové nároky
- za tím účelem existuje *limited memory BFGS* - L-BFGS
- ta snižuje paměťové nároky z $O(n^2)$ na $O(n)$, kde n je počet hledaných parametrů

A. Buckley, A. Lenir, *QN-like variable storage conjugate gradients*, Mathematical Programming vol. 27, pp. 155–175 (1983).
D. C. Liu, J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Mathematical Programming vol. 45, pp. 503–528 (1989).

Sedlové body

- v tomto bodě je i druhá derivace nulová a Newtonova metoda vede na podíl $0/0$
- v těchto bodech metody druhého řádu nefungují lépe než metody prvního řádu
- přitom metody druhého řádu jsou výpočetně náročnější
- zejména momentové metody mohou pomoci sedlový bod překonat
- ukazuje se, že ztrátové funkce pro NN sedlové body často obsahují, proto jsou pro učení NN stále spíše preferovány metody prvního řádu



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, Advances in Neural Information Processing Systems 27 (NIPS 2014).

Polyakovo průměrování

- Polyakovo průměrování je další způsob, jak stabilizovat gradientní metody
- lze ho kombinovat s libovolnou gradientní metodou
- pokud některá gradientní metoda vygenerovala posloupnost parametrů

$$\vec{w}^{(1)}, \ldots, \vec{w}^{(k)}$$

pak lze použít některou z následujících formulí

$$\begin{aligned}
\vec{w}_{av}^{(k)} &= \frac{1}{k}\sum_{i=1}^{k}\vec{w}^{(i)}, \\
\vec{w}_{av}^{(k)} &= \frac{\sum_{i=1}^{k}\beta^{k-i}\vec{w}^{(i)}}{\sum_{i=1}^{k}\beta^{k-i}}, \\
\vec{w}_{av}^{(k)} &= (1-\beta)\vec{w}^{(k)} + \beta\vec{w}^{(k-1)}
\end{aligned}$$

Lokální extrémy

- ztrátové funkce, se kterými pracujeme při učení NN jsou velmi komplexní
- obsahují mnoho lokálních minim a je prakticky nemožné najít globální minimum
- ukazuje se ale, že lokální minima u reálných NN mají hodnotu ztrátové funkce velmi blízko globálnímu minimu
- lokální minima jsou problémem spíše z pohledu generalizace
  - díky tomu, že NN trénujeme jen na podmnožině všech možných dat, pracujeme jen s určitou aproximací ztrátové funkce
  - pokud ta špatně aproximuje teoretickou skutečnou ztrátovou funkci, pak lokální minima nalezená při učení nemusí odpovídat lokálním minimum plnohodnotné ztrátové funkce¨

A. M. Saxe, J. L. McClelland, S. Ganguli, *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*, arXiv:1312.6120, 2013.

Generalizace

"Všechny generalizace jsou nebezpečné, včetně této." - A. Dumas, Ch.C.Aggarwal

- generalizace je schopnost neuronové sítě správně fungovat na datech, na kterých nebyla učena
- pokud síť správně funguje na učících datech a jinak ne, došlo k přetrénování
- používá se několik postupů, jak tomuto zabránit

L_1/L_2-regularizace

- možná jeden z nejoblíbenějších postupů
- v optimalizacích se této regularizaci říká Tichonovova regularizace
- ke ztrátové funkci se přidá L_1/L_2 norma vektoru parametrů $\vec{w}$
- to bude vyžadovat nižší hodnoty těchto parametrů
- parametr s menší hodnotou není pro model tak podstatný a tudíž by mohl být i zanedbán
- tím se model stává jednodušší
- L_2 regularizace je zřejmě častěji používaná
- L_1 regularizace je vhodnější, pokud chceme některé parametry ještě více přitlačit k nule
- to je vhodné pro učení řídkých modelů, tj. modelů, kde je většina parametrů nulová

Přidání gaussovského šumu

- přidání gaussovského šumu ke vstupním datům $\vec{x}_i$ má podobný efekt jako L_2 regularizace

Dropout

- spočívá v náhodném odstranění některých vnitřních uzlů

Výpočetní příklady

Reálná data pro klasifikaci lze najít např. zde:

- UC Irvine Machine Learning Repository,
- Kaggle Datasets,
- Scikit-Learn Datasets,
- OpenML.

Sdílení parametrů

- některé vnitřní váhy mohou sdílet jeden stejný parametr
- tím se snižuje počet parametrů, které je potřeba se učit
- tak lze neuronovou síť modifikovat pro určitou specifickou doménu
- příkladem mohou být
  - rekurentní neuronové sítě - *recurrent neural networks, RNN*
  - konvoluční neuronové sítě - *convolutional neural networks, CNN*