Tomáš Oberhuber

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

Gradientní sestup

Gradientní sestup aplikovaný na úlohu

$$\min_{\vec{w}} f(\vec{w})$$

budeme psát ve tvaru

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \alpha_G \nabla f(\vec{w}^{(k)})$$

S. Sun, Z. Cao, H. Zhu and J. Zhao, A Survey of Optimization Methods From a Machine Learning Perspective, in IEEE Transactions on Cybernetics, vol. 50, no. 8, pp. 3668-3681, 2020.

Gradientní sestup - kvadratická úloha

GD si nyní rozebereme pro případ kvadratické úlohy (Why Momentum Really Works):

$$f(\vec{w}) = \frac{1}{2}\vec{w}^T \mathbb{A}\vec{w} - \vec{b}^T\vec{w},$$

pro $\vec{w} \in Rn$ a $\mathbb{A}$ symetrická, PD a regulární.

- Jde o konvexní úlohu.
- Řešením je

$$\vec{w}^* = \mathbb{A}^{-1}\vec{b}.$$

- Pro gradient platí

$$\nabla f(\vec{w}) = \mathbb{A}\vec{w} - \vec{b}.$$

Gradientní sestup - kvadratická úloha

Předpis pro gradient descent má tvar:

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \alpha_G \left(\mathbb{A}\vec{w}^{(k)} - \vec{b} \right)$$

- Jde tedy o **Richardsonovu metodu** (relaxovaná metoda postupných aproximací).
- Podle Schurovy věty lze $\mathbb{A}$ přepsat jako

$$\mathbb{A} = \mathbb{Q}\mathbb{D}\mathbb{Q}^T,$$

kde $\mathbb{Q}$ je unitární matice, $\mathbb{D} = diag(\lambda_1, \ldots, \lambda_n)$, λ_i jsou seřazená vlastní čísla matice $\mathbb{A}$, λ_1 nejmenší a λ_n největší.

- Vyjádříme s chybu v k-té iteraci v bázi dané maticí $\mathbb{Q}$

$$\vec{e}^{(k)} = \mathbb{Q}^T \left(\vec{w}^{(k)} - \vec{w}^* \right).$$

Gradientní sestup - kvadratická úloha

Dostáváme:

$$
\begin{aligned}
\vec{e}^{(k+1)} &= \mathbb{Q}^T\left(\vec{w}^{(k+1)} - \vec{w}^*\right) = \mathbb{Q}^T\left(\vec{w}^{(k)} - \alpha_G\left(\mathbb{A}\vec{w}^{(k)} - \vec{b}\right) - \vec{w}^*\right) \\
&= \mathbb{Q}^T\left(\vec{w}^{(k)} - \vec{w}^* - \alpha_G\left(\mathbb{A}\vec{w}^{(k)} - \vec{b}\right)\right) \\
&= \mathbb{Q}^T\left(\vec{w}^{(k)} - \vec{w}^* - \alpha_G\left(\mathbb{A}\vec{w}^{(k)} - \mathbb{A}\vec{w}^*\right)\right) \\
&= \mathbb{Q}^T\left(\vec{w}^{(k)} - \vec{w}^* - \alpha_G\mathbb{A}\left(\vec{w}^{(k)} - \vec{w}^*\right)\right) \\
&= \mathbb{Q}^T\left(\left(\mathbb{I} - \alpha_G\mathbb{A}\right)\left(\vec{w}^{(k)} - \vec{w}^*\right)\right) \\
&= \mathbb{Q}^T\left(\left(\mathbb{I} - \alpha_G\mathbb{Q}\mathbb{D}\mathbb{Q}^T\right)\left(\vec{w}^{(k)} - \vec{w}^*\right)\right) = \left(\mathbb{Q}^T - \alpha_G\mathbb{D}\mathbb{Q}^T\right)\left(\vec{w}^{(k)} - \vec{w}^*\right) \\
&= \left(\mathbb{I} - \alpha_G\mathbb{D}\right)\mathbb{Q}^T\left(\vec{w}^{(k)} - \vec{w}^*\right) = \left(\mathbb{I} - \alpha_G\mathbb{D}\right)\vec{e}^{(k)}.
\end{aligned}
$$

Gradientní sestup - kvadratická úloha

Po složkách pak platí:

$$e_i^{(k+1)} = e_i^{(k)} - \alpha_G \lambda_i e_i^{(k)} = (1 - \alpha_G \lambda_i) e_i^{(k)} = (1 - \alpha_G \lambda_i)^k e_i^{(0)}.$$

Celkem tedy platí:

$$\vec{w}^{(k)} - \vec{w}^* = \mathbb{Q}\vec{e}^{(k)} = \sum_{i=1}^{n} e_i^{(0)} (1 - \alpha_G \lambda_i)^k \vec{q}_i.$$

Celkovou chybu lze tedy rozložit do nezávislých směrů, ve kterých konverguje k nule.

Gradientní sestup - kvadratická úloha

- Aby metoda konvergovala, je potřeba volit α, tak aby

$$|1 - \alpha_G \lambda_i| < 1 \text{ pro všechna } \lambda_i \in \sigma(\mathbb{A}),$$

tj.

$$\alpha_G \lambda_i \in (0, 2).$$

- Celková rychlost konvergence je dána rychlostí konvergence nejpomalejší složky, tj.

$$|1 - \alpha_G \lambda_i| \approx 1,$$

což nastává pro nejmenší nebo největší vl. číslo. Tedy

$$\mathit{rate}(\alpha_G) = \max_i |1 - \alpha_G \lambda_i| = \max\{|1 - \alpha_G \lambda_1|, |1 - \alpha_G \lambda_n|\}.$$

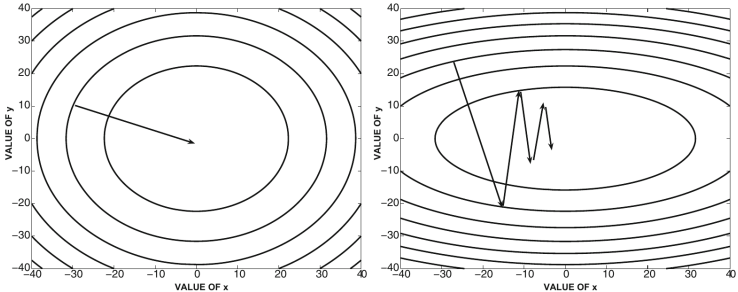
Gradientní sestup - kvadratická úloha

- Uvažujme nyní

$$f(\vec{w}) = \frac{1}{2}\vec{w}^T \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \vec{w} \Rightarrow \nabla f = \begin{pmatrix} \lambda_1 w_1 \\ \lambda_2 w_2 \end{pmatrix}.$$

- Pro $\lambda_1 \ll \lambda_2$ 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**.

Gradientní sestup - kvadratická úloha



(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

Gradientní sestup - kvadratická úloha

Optimální rychlost konvergence dostaneme pokud

$$|1 - \alpha_G \lambda_1| = |1 - \alpha_G \lambda_n|.$$

Netriviální řešení dostaneme volbou

$$1 - \alpha_G \lambda_1 = -(1 - \alpha_G \lambda_n) \Rightarrow \arg\min_{\alpha_G} \mathit{rate}(\alpha_G) = \frac{2}{\lambda_1 + \lambda_n},$$

pro které dostáváme (dosazením do $1 - \alpha_G \lambda_1$)

$$\min_{\alpha_G} \mathit{rate}(\alpha_G) = \frac{\lambda_n/\lambda_1 - 1}{\lambda_n/\lambda_1 + 1} = \frac{\kappa - 1}{\kappa + 1},$$

kde κ je číslo podmíněnosti matice $\mathbb{A}$.

Gradientní sestup - polynomiální regrese

Vrátíme se k problému polynomiální regrese:

$$P(\vec{w}, x) = w_d x^d + \dots w_1 x + w_0,$$

$$\begin{aligned}
\min_{\vec{w}} \frac{1}{2} \sum_{i=1}^{N} \left(P(\vec{w}, x) - d_i\right)^2 &= \min_{\vec{w}} \frac{1}{2} \|\mathbb{V}^{(n)}_{x_0,\dots,x_n} \vec{w} - \vec{d}\|_2^2 = \min_{\vec{w}} \frac{1}{2} \|\mathbb{V}\vec{w} - \vec{d}\|_2^2 \\
&= \min_{\vec{w}} \frac{1}{2} \left(\vec{w}^T \mathbb{V}^T \mathbb{V} \vec{w} - 2\vec{y}^T \mathbb{V} \vec{w} + \vec{y}^T \vec{y}\right) \\
&= \min_{\vec{w}} \frac{1}{2} \left(\vec{w}^T \mathbb{V}^T \mathbb{V} \vec{w} - 2\vec{y}^T \mathbb{V} \vec{w}\right).
\end{aligned}$$

Opět použijeme transformaci $\mathbb{V}^T\mathbb{V} = \mathbb{Q}\mathbb{D}\mathbb{Q}^T$.

Gradientní sestup - polynomiální regrese

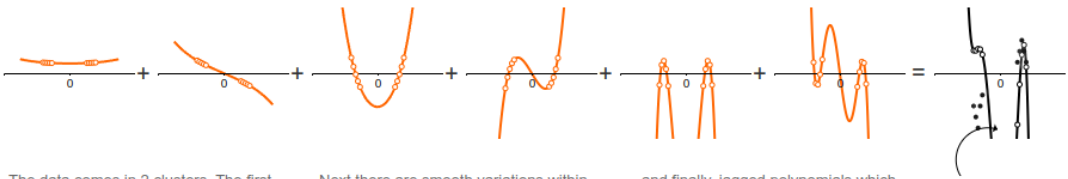
Od polynomů $p_i = x^{i-1}$ pro $i = 1, \ldots, d+1$ přejdeme k polynomům

$$\bar{p}_i = \sum_{j=1}^{d+1} q_{ij} p_j,$$

které přísluší jednotlivým vlastním číslům seřazeným sestupně podle velikosti. Tím převedeme optimalizační úlohu na $d+1$ nezávislých optimalizačních úloh, kde každý parametr může optimalizován nezávisle.

Gradientní sestup - polynomiální regrese

$$0.458\bar{p}_1 + 0.903\bar{p}_2 + 6.13\bar{p}_3 + 12.4\bar{p}_4 + 128\bar{p}_5 + 261\bar{p}_6 = \text{model}$$



The data comes in 2 clusters. The first 2 eigenfeatures capture variations between the clusters.

Next there are smooth variations within clusters, peaks within clusters,

and finally, jagged polynomials which differ wildly on neighboring points.

drag points to fit data

Why Momentum Really Works

Gradientní sestup - polynomiální regrese

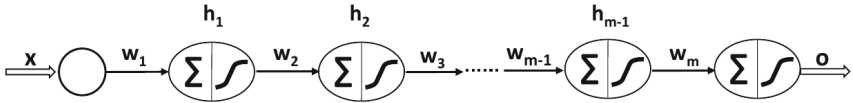
- Jako první vidíme polynomy odpovídající větším vlastním číslům a na konci jsou polynomy odpovídající mešním vlastním číslům.
- U těch pozorujeme větší koeficienty, protože ty jsou násobené malým vlastním číslem.
- U modelů nás dále zajímá citlivost na šum, tj. drobné změny ve vstupních datech.
- Pokud lehce pohneme jedním bodem:
  - U polynomů odpovídajících **velkým vlastním číslům** dojde k **malé změně** příslušného koeficientu.
  - U polynomů odpovídajících **malým vlastním číslům** dojde k **velké změně** příslušného koeficientu.
- Polynomy odpovídající malým vlastním číslům jsou tak jakési **patologické směry**.

Gradientní sestup - polynomiální regrese

- Jak ale víme, tak u polynomů odpovídajících nejmenším vlastním číslům probíhá konvergence mnohem pomaleji.
- Iterace tedy můžeme zastavit dříve, než se ony patologické směry plně vyvinou.
- Jde o tzv. **early stopping**.
- To vlastně zabrání vzniku vysokých koeficientů v polynomy a jde tedy o stejný efekt, jako má Tichonova regularizace.

Mizející a explodující gradient

- předpokládejme neuronovou síť o $l + 1$ vrstvách, každá vrstva má 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 malým rozptylem a středem v nule
- přesto tento efekt nelze úplně eliminovat a způsobuje to problém s konvergencí GD

Gradientní sestup - 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

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

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

Momentové metody

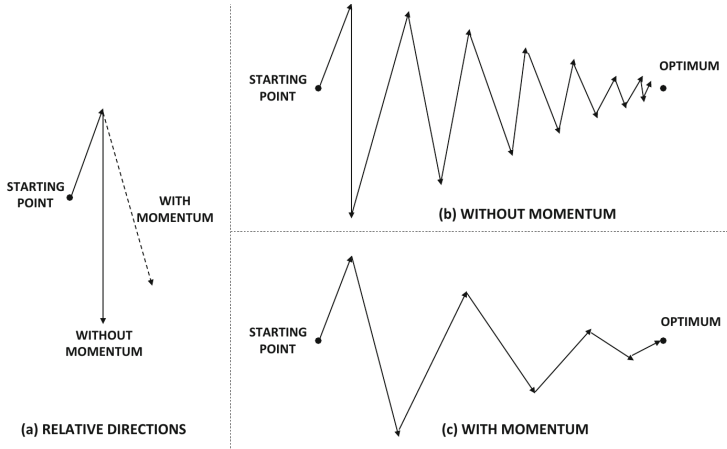


Figure: Ch.C.Aggarwal, Neural networks and deep learning

Momentové metody

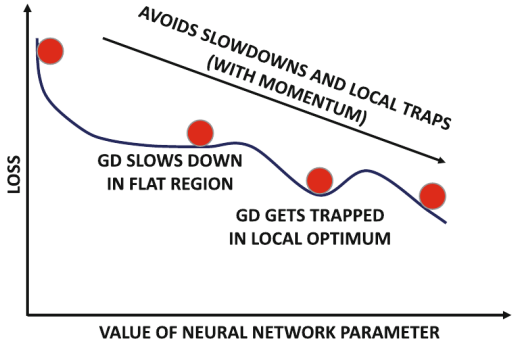


Figure: Ch.C.Aggarwal, Neural networks and deep learning

- 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

Momentové metody - kvadratická úloha

Why Momentum Really Works

Nyní budeme řešit kvadratickou úlohu pomocí momentové metody:

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

Nyní je

$$\nabla f(\vec{w}^{(k)}) = \mathbb{A}\vec{w}^{(k)} - \vec{b},$$

a tedy

$$\begin{aligned} \vec{v}^{(k+1)} &= \beta v^{(k)} + \left(\mathbb{A}\vec{w}^{(k)} - \vec{b}\right), \\ \vec{w}^{(k+1)} &= \vec{w}^{(k)} - \alpha_M \vec{v}^{(k+1)}. \end{aligned}$$

Momentové metody - kvadratická úloha

Provedeme stejnou transformaci

$$\vec{x}^{(k)} = \mathbb{Q}\left(\vec{w}^{(k)} - \vec{w}^{*}\right) \text{ a } \vec{y}^{(k)} = \mathbb{Q}\vec{v}^{(k)},$$

a iterace přepíšeme do tvaru

$$\begin{aligned} y_i^{(k+1)} &= \beta y_i^{(k)} + \lambda_i x_i^{(k)}, \\ x_i^{(k+1)} &= x_i^{(k)} - \alpha_M y_i^{(k+1)}. \end{aligned}$$

Což lze zapsat i jako

$$\begin{pmatrix} 1 & 0 \\ \alpha_M & 1 \end{pmatrix} \begin{pmatrix} y_i^{(k+1)} \\ x_i^{(k+1)} \end{pmatrix} = \begin{pmatrix} \beta & \lambda_i \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y_i^{(k)} \\ x_i^{(k)} \end{pmatrix} \dots$$

Momentové metody - kvadratická úloha

... a následně převést na tvar

$$\begin{pmatrix} y_i^{(k)} \\ x_i^{(k)} \end{pmatrix} = \mathbb{R}^k \begin{pmatrix} y_i^{(0)} \\ x_i^{(0)} \end{pmatrix} \text{ kde } \mathbb{R} = \begin{pmatrix} \beta & \lambda_i \\ -\alpha_M \beta & 1 - \alpha_M \lambda_i \end{pmatrix}.$$

Lze ukázat (Why Momentum Really Works), že podmínky konvergence jsou:

$$\begin{aligned} 0 < \alpha\lambda_i < 2 + 2\beta, \\ 0 \leq \beta < 1. \end{aligned}$$

- Pro $\beta = 0$, dostáváme stejnou podmínku jako pro GD tj. $0 < \alpha_M \lambda_i < 2$.
- Pro $\beta > 1$ vzniká větší prostor i pro $\alpha_M \lambda_i$.

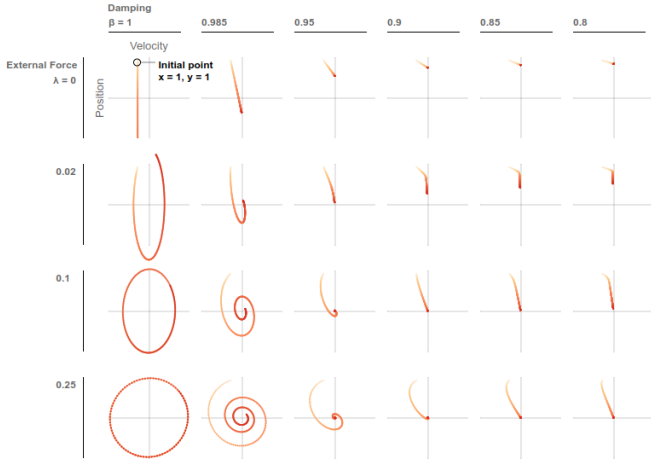
Momentové metody - kvadratická úloha

Předpis momentové metody lze chápat jako pohyb hmotného bodu:

$$\begin{aligned} \underbrace{y_i^{(k+1)}}_{\text{rychlost}} &= \underbrace{\beta y_i^{(k)}}_{\text{tření}} + \underbrace{\lambda_i}_{\text{externí síla}} x_i^{(k)}, \\ \underbrace{x_i^{(k+1)}}_{\text{pozice}} &= x_i^{(k)} - \alpha_M y_i^{(k+1)}. \end{aligned}$$

Ukážeme si závislost konvergence na změně parametrů β a λ.

Momentové metody - kvadratická úloha



Why Momentum Really Works

Momentové metody - kvadratická úloha

Lze ukázat, že optimální volby jsou

$$\alpha_M = \left(\frac{2}{\sqrt{\lambda_1} + \sqrt{\lambda_n}}\right)^2 \quad \beta = \left(\frac{\sqrt{\lambda_n} - \sqrt{\lambda_1}}{\sqrt{\lambda_n} + \sqrt{\lambda_1}}\right),$$

a rychlost konvergence pak vychází jako

$$\mathit{rate}(\alpha_M, \beta) = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}.$$

Pro porovnání pro GD jsme dostali

$$\mathit{rate}(\alpha_G) = \frac{\kappa - 1}{\kappa + 1} \text{ pro } \alpha_G = \frac{2}{\lambda_1 + \lambda_n}.$$

Momentové metody - kvadratická úloha

Pokud narazíme na špatně podmíněnou úlohu, pak $\lambda_1 \ll \lambda_n$ a dostáváme

$$\alpha_G \approx \frac{2}{\lambda_n} \text{ pro GD,}$$

a

$$\alpha_M \approx \frac{4}{\lambda_n} \quad \beta \approx \frac{\sqrt{\lambda_n}}{\sqrt{\lambda_n}} \approx 1 \text{ pro momentovou metodu.}$$

Gradientní sestup lze rozepsat takto:

$$
\begin{aligned}
\vec{w}^{(1)} &= \vec{w}^{(0)} - \alpha_G \nabla f(\vec{w}^{(0)}), \\
\vec{w}^{(2)} &= \vec{w}^{(1)} - \alpha_G \nabla f(\vec{w}^{(0)}) - \alpha_G \nabla f(\vec{w}^{(1)}), \\
\vec{w}^{(3)} &= \vec{w}^{(2)} - \alpha_G \nabla f(\vec{w}^{(0)}) - \alpha_G \nabla f(\vec{w}^{(1)}) - \alpha_G \nabla f(\vec{w}^{(2)}), \\
&\vdots \\
\vec{w}^{(k+1)} &= \vec{w}^{(k)} - \alpha_G \sum_{i=0}^{k} \nabla f(\vec{w}^{(k-i)})
\end{aligned}
$$

Podobně rozepíšeme i momentovou metodu:

$$
\begin{aligned}
\vec{v}^{(1)} &= \nabla f(\vec{w}^{(0)}), \\
\vec{v}^{(2)} &= \beta v^{(1)} + \nabla f(\vec{w}^{(1)}) = \beta \nabla f(\vec{w}^{(0)}) + \nabla f(\vec{w}^{(1)}), \\
\vec{v}^{(3)} &= \beta v^{(2)} + \nabla f(\vec{w}^{(2)}) = \beta^2 \nabla f(\vec{w}^{(0)}) + \beta \nabla f(\vec{w}^{(1)}) \\
&\quad + \nabla f(\vec{w}^{(2)}), \\
&\vdots \\
\vec{v}^{(k)} &= \sum_{i=1}^{k} \beta^{k-i} \nabla f(\vec{w}^{(i-1)})
\end{aligned}
$$

$$
\begin{aligned}
\vec{w}^{(1)} &= \vec{w}^{(0)} - \alpha_M \vec{v}^{(1)}, \\
\vec{w}^{(2)} &= \vec{w}^{(1)} - \alpha_M \vec{v}^{(2)} = \vec{w}^{(0)} - \alpha_M \vec{v}^{(1)} - \alpha_M \vec{v}^{(2)} \\
&\vdots \\
\vec{w}^{(k)} &= \vec{w}^{(0)} - \alpha_M \sum_{j=1}^{k} \vec{v}^{(j)} \\
&= \vec{w}^{(0)} - \alpha_M \sum_{j=1}^{k} \sum_{i=1}^{j} \beta^{j-i} \nabla f(\vec{w}^{(i-1)}) \\
&= \vec{w}^{(0)} - \alpha_M \sum_{j=1}^{k} \sum_{i=1}^{j} \beta^{j-i} \nabla f(\vec{w}^{(i-1)})
\end{aligned}
$$

$$
\begin{aligned}
\vec{w}^{(k)} &= \vec{w}^{(0)} - \alpha_M \sum_{j=1}^{k} \sum_{i=1}^{j} \beta^{j-i} \nabla f(\vec{w}^{(i-1)}) = \vec{w}^{(0)} - \alpha_M \Big(\\
&\quad \nabla f(\vec{w}^{(0)}) + \\
&\quad \beta \nabla f(\vec{w}^{(0)}) + \nabla f(\vec{w}^{(1)}) + \\
&\quad \beta^2 \nabla f(\vec{w}^{(0)}) + \beta \nabla f(\vec{w}^{(1)}) + \nabla f(\vec{w}^{(2)}) + \ldots + \\
&\quad \beta^{k-1} \nabla f(\vec{w}^{(0)}) + \beta^{k-2} \nabla f(\vec{w}^{(1)}) + \ldots + \nabla f(\vec{w}^{(k-1)}) \Big) \\
&= \vec{w}^{(0)} - \alpha_M \left(\sum_{i=1}^{k} \sum_{j=1}^{i} \beta^{j-1} \nabla f(\vec{w}^{(k-i)}) \right)
\end{aligned}
$$

Předpis tedy v každé iteraci využívá celou historii kroků.

Dále dostáváme:

$$\begin{aligned}
\vec{w}^{(k)} &= \vec{w}^{(0)} - \alpha_M \left(\sum_{i=1}^{k} \sum_{j=1}^{i} \beta^{j-1} \nabla f(\vec{w}^{(k-i)}) \right) \\
&= \vec{w}^{(0)} - \alpha_M \left(\sum_{i=1}^{k} \frac{1-\beta^i}{1-\beta} \nabla f(\vec{w}^{(k-i)}) \right) \\
&= \vec{w}^{(0)} - \alpha_M \left(\sum_{i=1}^{k} \frac{1-\beta^{k-i}}{1-\beta} \nabla f(\vec{w}^{(i)}) \right)
\end{aligned}$$

Celkem tedy dostáváme:

$$\vec{w}^{(k)} = \vec{w}^{(0)} + \sum_{i=1}^{k} \gamma_i^k \nabla f(\vec{w}^{(i-1)}), \text{ kde } \gamma_i^k = \alpha_M \frac{1-\beta^{k-i}}{1-\beta}.$$

Předpis lze zobecnit na tvar

$$\vec{w}^{(k)} = \vec{w}^{(0)} + \sum_{i=1}^{k} \mathbb{D}_i^k \nabla f(\vec{w}^{(i-1)}),$$

kde $\mathbb{D}_i^k$ je diagonální matice.

Tím získáváme pro každou složku gradientu vlastní relaxační parametr.

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

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

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

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

- $\gamma^{(k)} \to \gamma$ 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
- na rozdíl 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 práh t_{min} a t_{max} a měníme jednotlivé složky gradientu

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

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

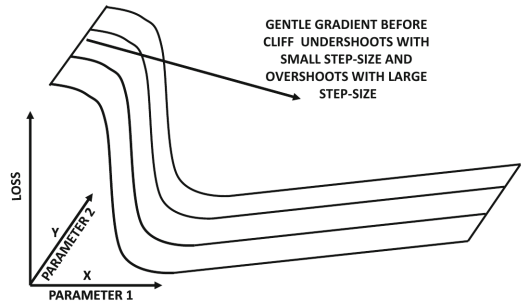
$$\nabla_{\vec{w}} L := \frac{\nabla_{\vec{w}} L}{\|\nabla_{\vec{w}} L\|_2}$$

- 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

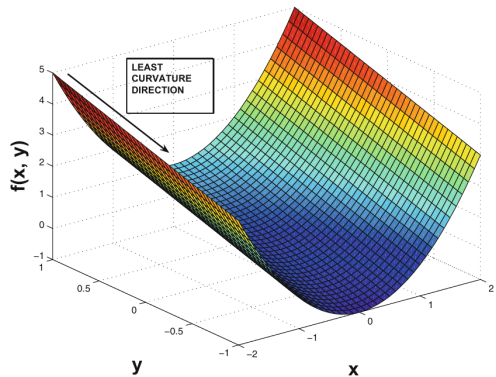
$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \mathbb{H}^{-1}(\vec{w}) \nabla L(\vec{w})$$

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 konjugovaných gradientů

- 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

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

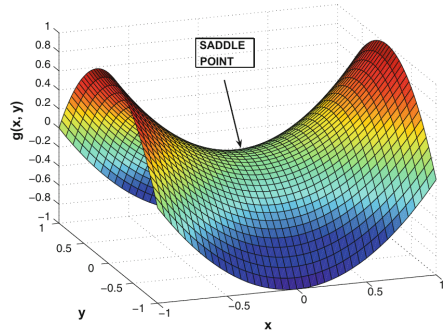
Metody druhého řádu

- 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 druhá derivace semidefinitní
- 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*