

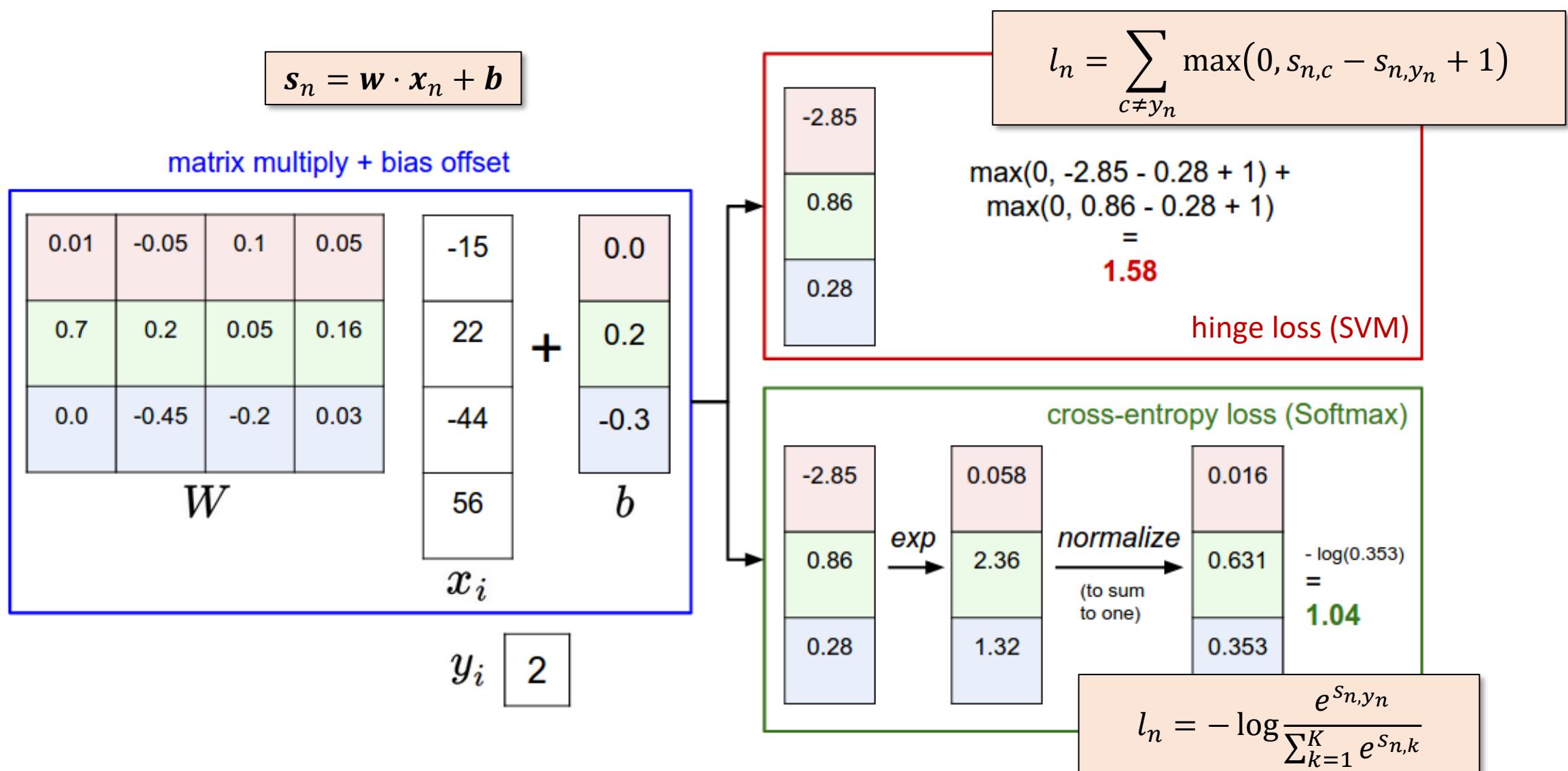
Aplikace neuronových sítí

Metoda největšího spádu a její varianty

Návrh a trénování lineárního klasifikátoru

1. Navrheme diskriminativní klasifikační funkci s upravitelnými parametry
2. Kvantifikujeme její úspěšnost klasifikace nějakým kritériem
3. **Budeme upravovat parametry a poznamenávat si výsledek (hodnotu kritéria)**

Lineární model a klasifikační loss



obrázek: <http://cs231n.github.io/linear-classify/>

Klasifikátor + chybovost jako složená funkce

- Výpočet skóre \hat{s}_n je funkce vstupu x_n a parametrů (w, b)

$$\hat{s}_n = f_{\text{model}}(x_n, w, b)$$

- Výpočet hodnoty lossu l_n je funkce skóre \hat{s}_n a targetu y_n

$$l_n = L_n(\hat{s}_n, y_n)$$

- Výpočet hodnoty lossu l na celém datasetu je průměr nebo součet dílčích lossů

$$l = f_{\text{reduce}}(l_1, \dots, l_N),$$

- Loss l je tedy v konečném důsledku nějaká funkce L všech zainteresovaných veličin

$$l = L(x, y, w, b)$$

Optimalizační kritérium

- Pro zkrácení a zobecnění zápisu označíme data jako \mathbf{X} a parametry jako $\boldsymbol{\theta}$
- Data \mathbf{X} považujeme za konstantní, měnit se mohou pouze parametry $\boldsymbol{\theta}$
- Loss l (vyhodnocený na celém datasetu) je tedy při neměnnosti dat \mathbf{X} efektivně funkcií pouze parametrů $\boldsymbol{\theta}$

$$l = L(\mathbf{X}, \boldsymbol{\theta})$$

- Trénování znamená nalézt optimální parametry $\boldsymbol{\theta}^*$, které celkový loss l na datasetu \mathbf{X} minimalizují

Strojové učení je optimalizace

$$\theta^* = \operatorname{argmin}_{\theta} L(\mathbf{X}, \theta)$$



Minimalizace funkce metodou největšího spádu

Minimalizace funkce

- Mějme jednorozměrnou funkci

$$f(x): \mathbb{R} \rightarrow \mathbb{R}$$

tj. vstup i výstup jsou reálná čísla (skaláry)

- Chceme najít bod x^* , kde funkce f nabývá minimální hodnoty, tj.

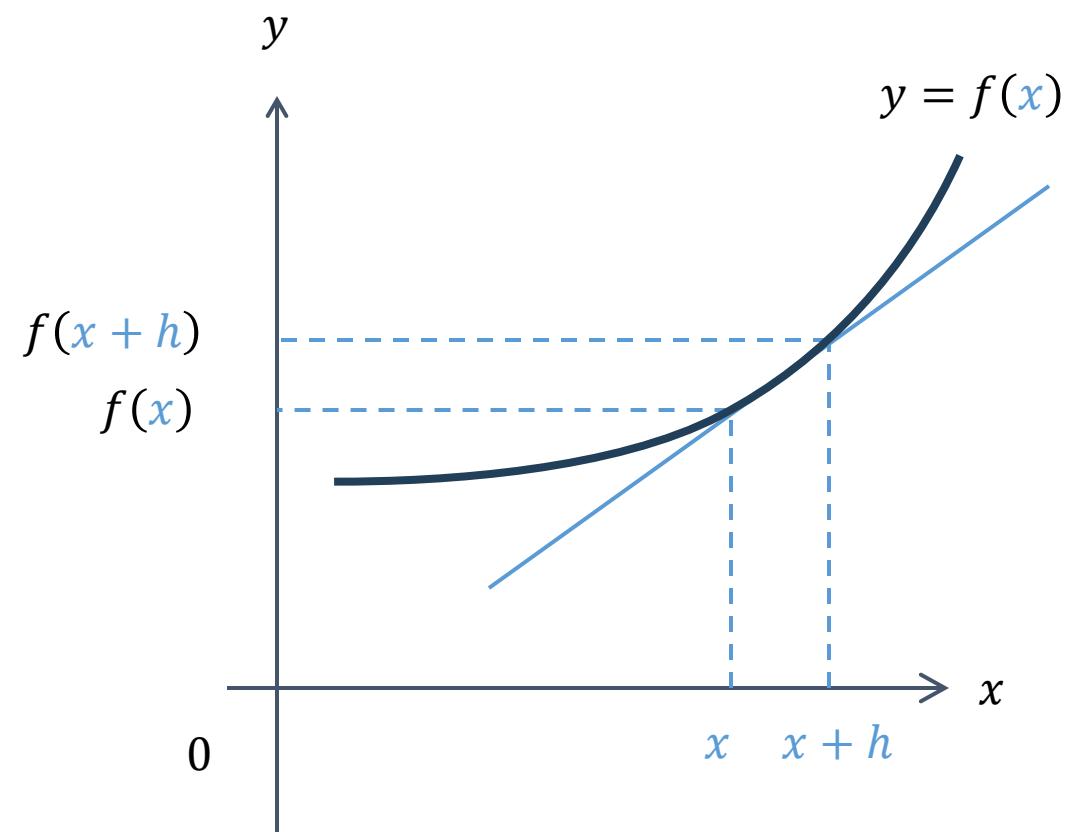
$$x^* = \operatorname{argmin}_x f(x)$$

Derivace funkce: 1D vstup, 1D výstup

- Derivace $\frac{df(x)}{dx}$ funkce $f(x)$ podle x je limita

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

- Derivace $\frac{df(x)}{dx}$ v bodě x nám říká, jak se změní hodnota funkce f , pokud "o trochu" zvětšíme vstup x



Derivace funkce: 1D vstup, 1D výstup

- Na $\frac{df(x)}{dx}$ můžeme nahlížet jako na funkci,

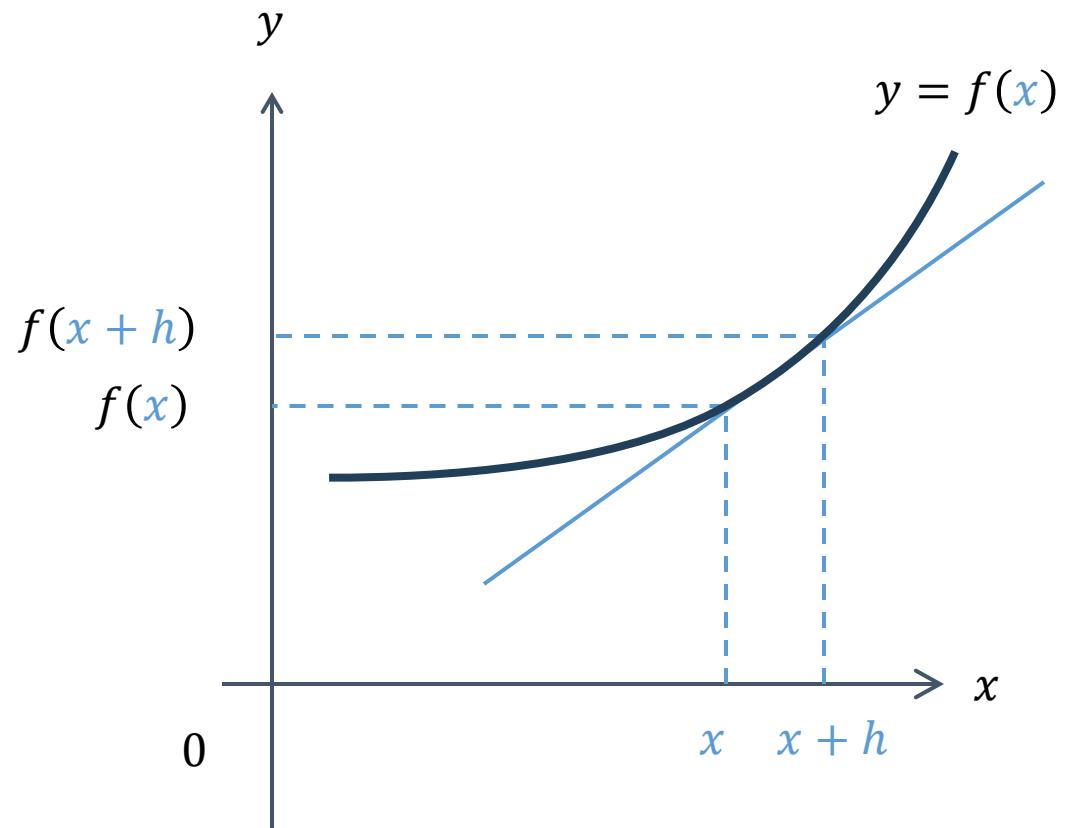
která v každém bodě x vrací sklon ρ

funkce $f(x)$

$$\rho(x) = \frac{df(x)}{dx}$$

- Pokud je $\frac{df(x)}{dx} > 0$, hodnota funkce f s rostoucím x roste; v opačném případě neroste nebo klesá

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$



Aproximace derivace: 1D vstup, 1D výstup

- Pokud neznáme či nechceme znát $\frac{df(x)}{dx}$ exaktně, můžeme ji approximovat diferencí

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h}$$

kde za krok h zvolíme nějaké malé číslo, např. $h = 0.001$

- Např.:

$$f(x) = x^2 \rightarrow \frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h} = 2 \cdot x$$

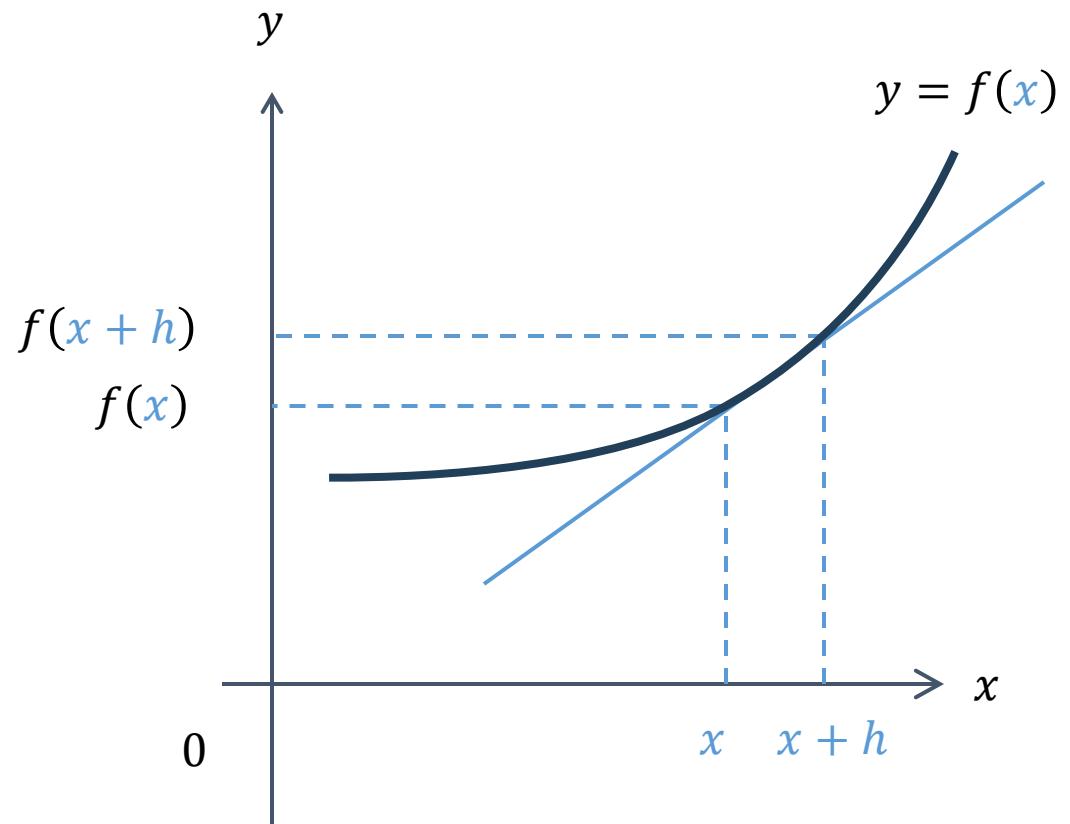
Chceme hodnotu derivace v bodě $x = 3$

$$\frac{df(3)}{dx} = 2 \cdot 3 = 6$$

Numerická approximace je

$$\frac{df(3)}{dx} \approx \frac{f(3.001) - f(3)}{0.001} = \frac{9.006001 - 9}{0.001} = 6.001$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$



Metoda největšího spádu (Gradient Descent, GD)

- Metoda největšího spádu využívá derivaci pro zjištění, kterým směrem funkce roste
 1. V aktuální pozici x spočítá derivaci $\frac{df(x)}{dx}$ (exaktně nebo numericky)
 2. Vylepší aktuální odhad x posunutím se ve směru opačném ke směru růstu, formálně

$$x' := x - \left[\gamma \cdot \frac{df(x)}{dx} \right]$$

Výsledná velikost **rozdílu** mezi původním odhadem a novým odhadem je ovlivněna

- a) velikostí kroku γ , kterou se výsledek derivace škáluje
- b) absolutní hodnotou derivace $\frac{df(x)}{dx}$, tj. strmostí $f(x)$ v aktuálním bodě x

3. S novým odhadem x' se uvedený postup opakuje
- Na začátku vyžaduje nějaký počáteční odhad x

Příklad GD: funkce $f(x) = x^4 + x + 2$

- Mějmě např. funkci

$$f(x) = x^4 + x + 2$$

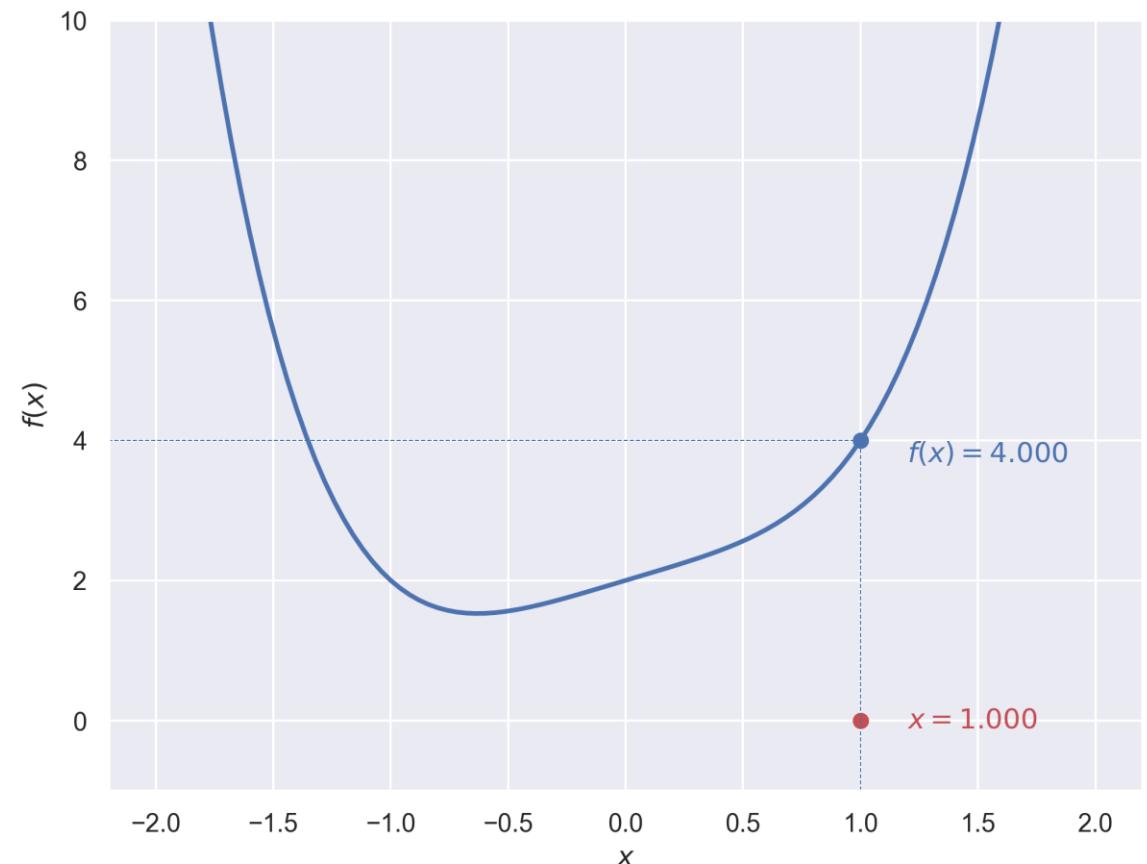
- Hledáme pozici x^* jejího minima, tj.

$$x^* = \underset{x}{\operatorname{argmin}} f(x)$$

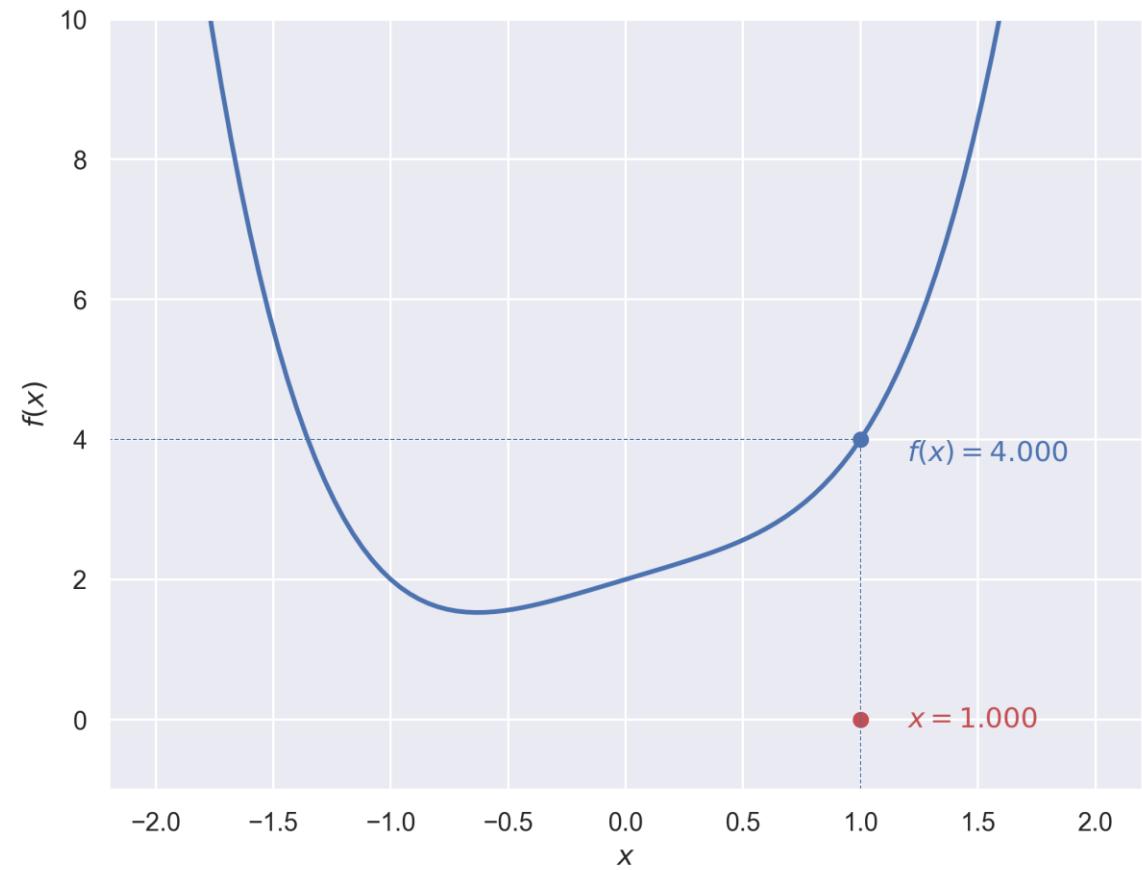
- Jako počáteční odhad minima zkusíme

$$x^{(0)} = 1$$

- Nastavíme $\gamma = 0.2$ a $h = 0.001$



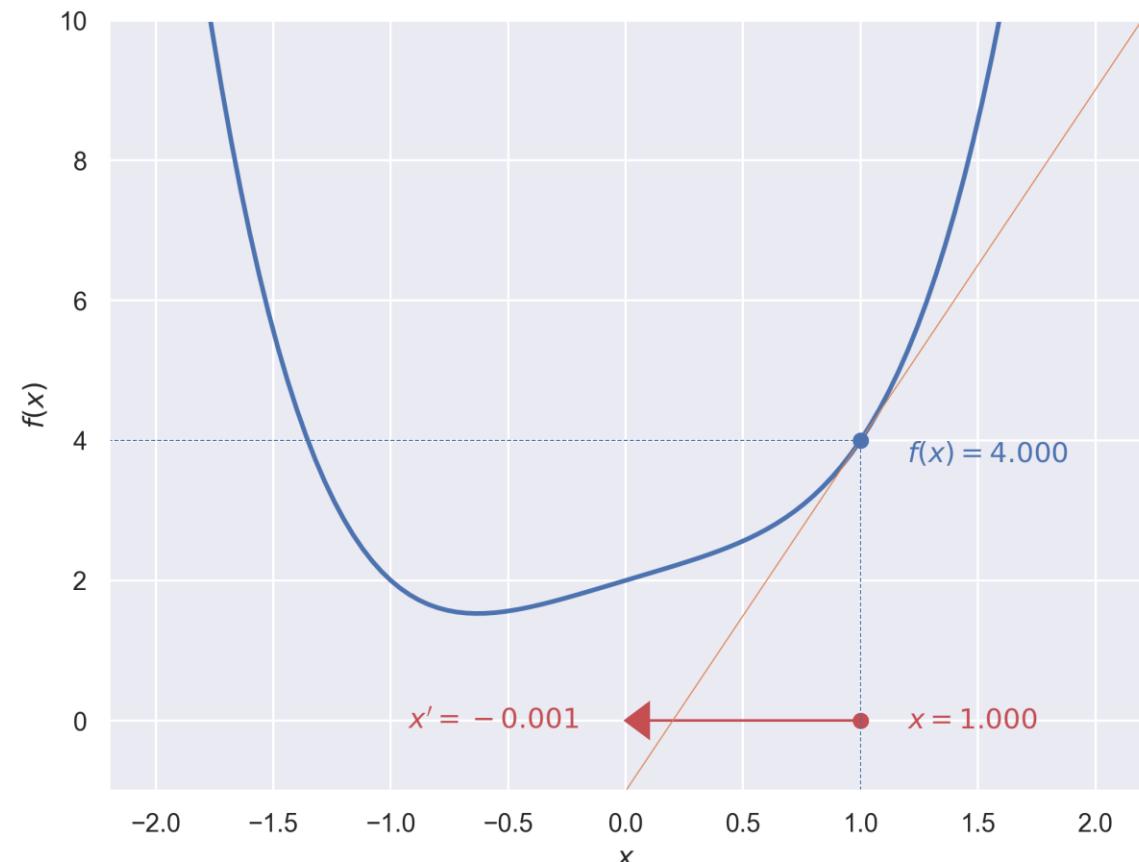
Příklad GD: funkce $f(x) = x^4 + x + 2$



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006

$$\begin{aligned}
 x &= 1.000 \\
 f(x) &= 1.000^4 + 1.000 + 2 = 4.000 \\
 \frac{df(x)}{dx} &= \frac{f(1.000 + 0.001) - f(1.000)}{0.001} \\
 &= \frac{4.005 - 4.000}{0.001} \\
 &= 5.006 \\
 x' &= 1.000 - 0.2 \cdot 5.006 \\
 &= -0.001
 \end{aligned}$$



Příklad GD: funkce $f(x) = x^4 + x + 2$

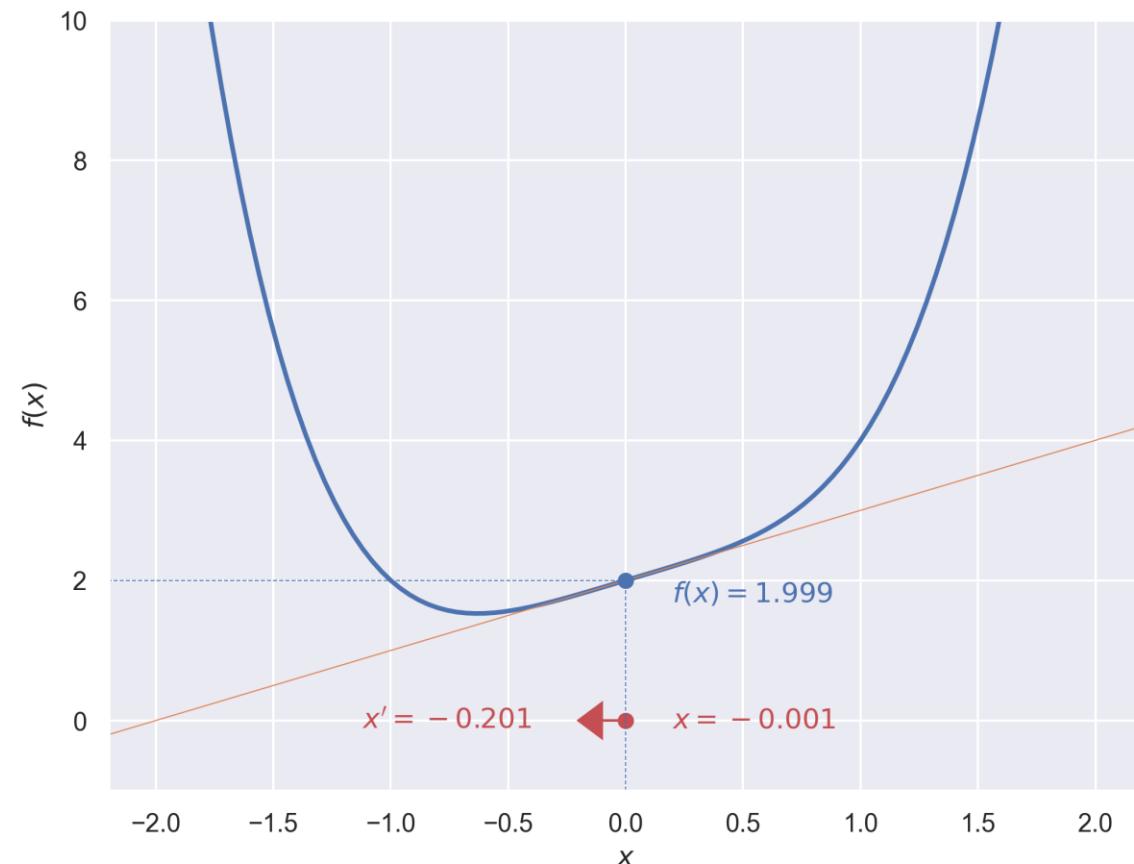
t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.000

$$x = -0.001$$

$$f(x) = (-0.001)^4 + (-0.001) + 2 = 1.999$$

$$\begin{aligned} \frac{df(x)}{dx} &= \frac{f(-0.001 + 0.001) - f(-0.001)}{0.001} \\ &= \frac{2.000 - 1.999}{0.001} \\ &= 1.000 \end{aligned}$$

$$\begin{aligned} x' &= -0.001 - 0.2 \cdot 1.000 \\ &= -0.201 \end{aligned}$$



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968

$$x = -0.201$$

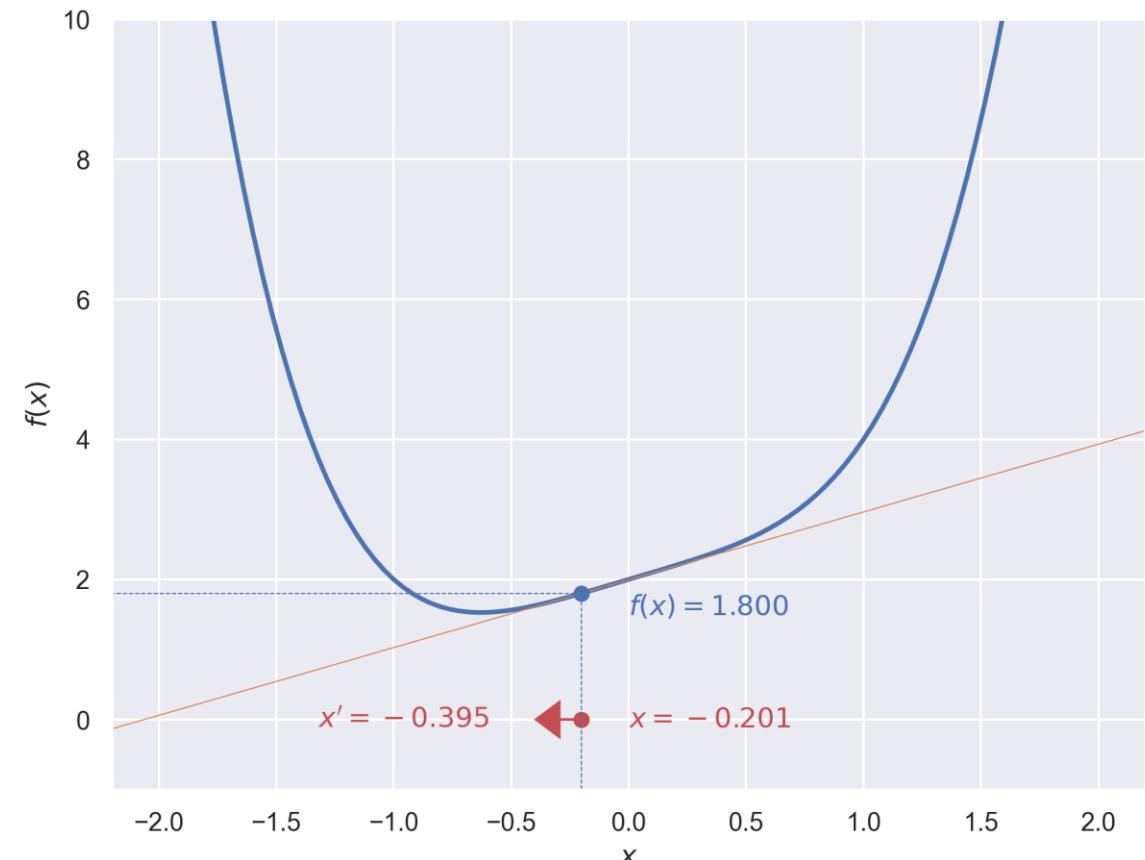
$$f(x) = (-0.201)^4 + (-0.201) + 2 = 1.800$$

$$\frac{df(x)}{dx} = \frac{f(-0.201 + 0.001) - f(-0.201)}{0.001}$$

$$= \frac{1.801 - 1.800}{0.001}$$

$$= 0.968$$

$$x' = -0.201 - 0.2 \cdot 0.968 \\ = -0.395$$



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968
3	-0.395	1.630	0.755

$$x = -0.395$$

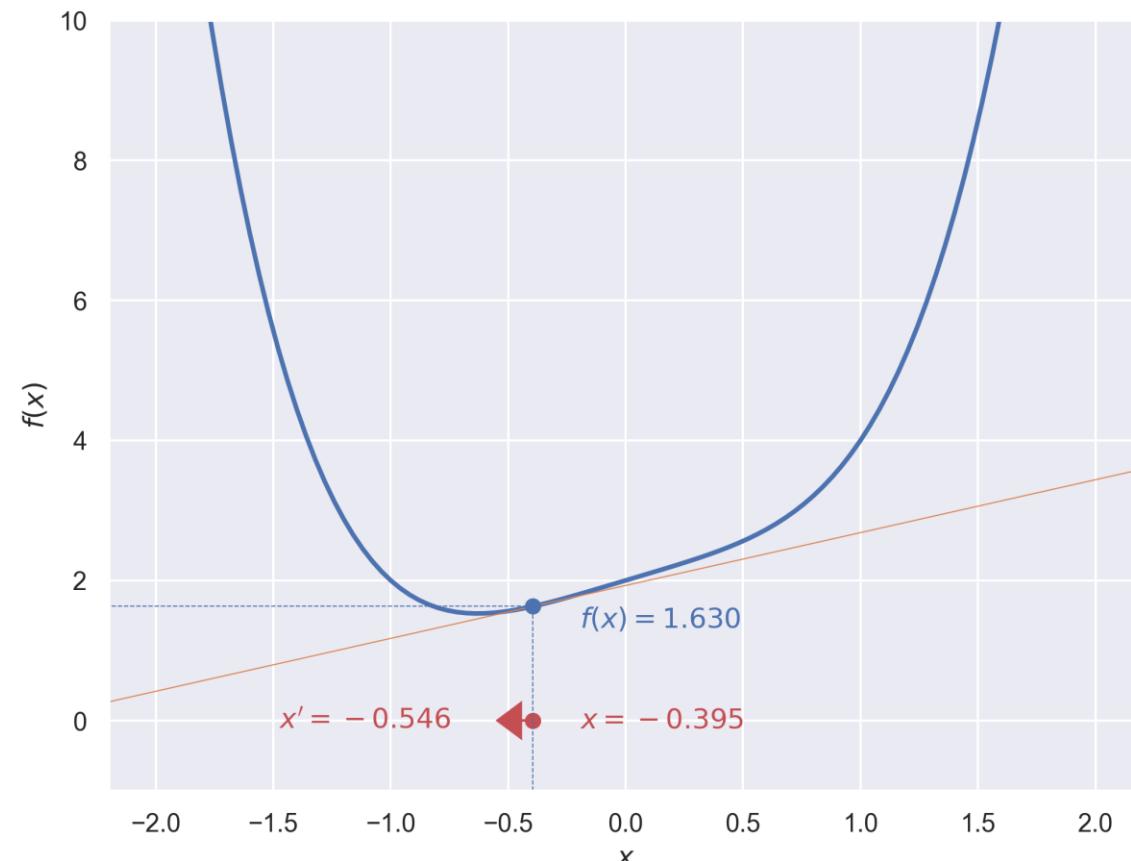
$$f(x) = (-0.395)^4 + (-0.395) + 2 = 1.630$$

$$\frac{df(x)}{dx} = \frac{f(-0.395 + 0.001) - f(-0.395)}{0.001}$$

$$= \frac{1.630 - 1.630}{0.001}$$

$$= 0.755$$

$$x' = -0.395 - 0.2 \cdot 0.755 \\ = -0.546$$



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968
3	-0.395	1.630	0.755
4	-0.546	1.543	0.352

$$x = -0.546$$

$$f(x) = (-0.546)^4 + (-0.546) + 2 = 1.543$$

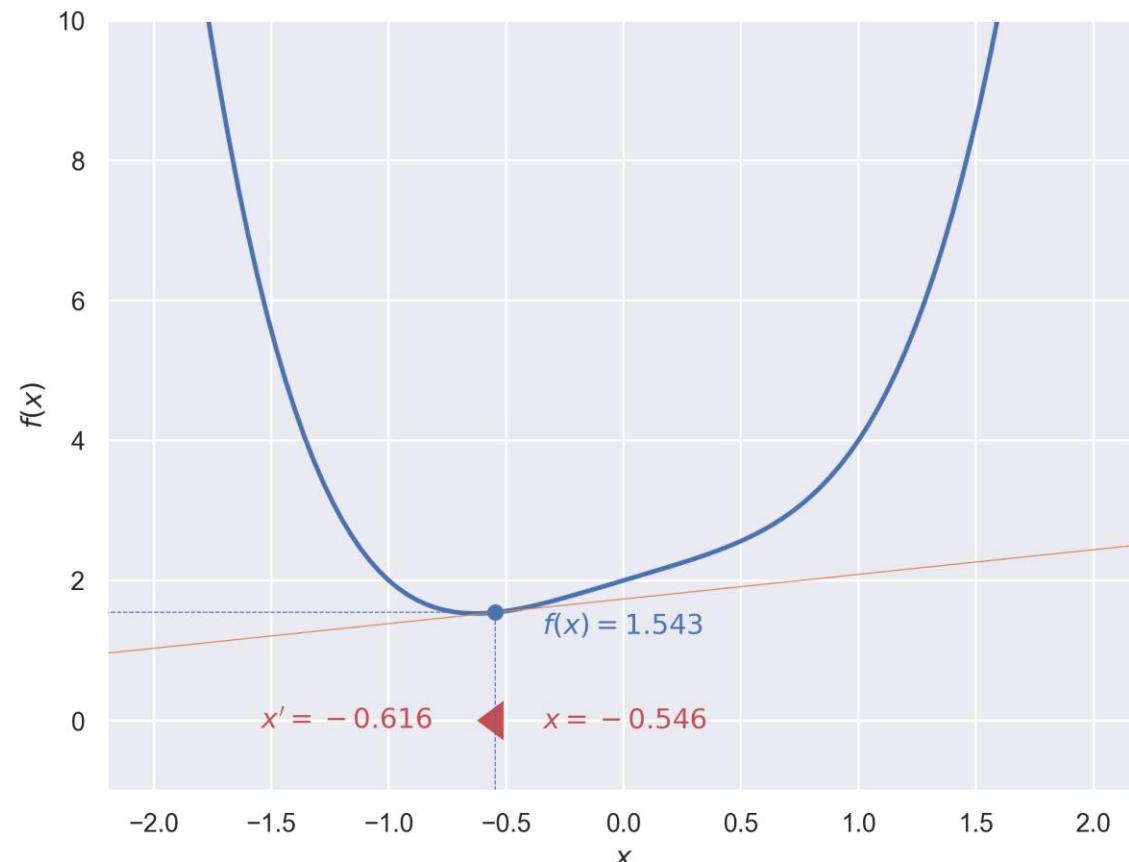
$$\frac{df(x)}{dx} = \frac{f(-0.546 + 0.001) - f(-0.546)}{0.001}$$

$$= \frac{1.543 - 1.543}{0.001}$$

$$= 0.352$$

$$x' = -0.546 - 0.2 \cdot 0.352$$

$$= -0.616$$



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968
3	-0.395	1.630	0.755
4	-0.546	1.543	0.352
5	-0.616	1.528	0.067

$$x = -0.616$$

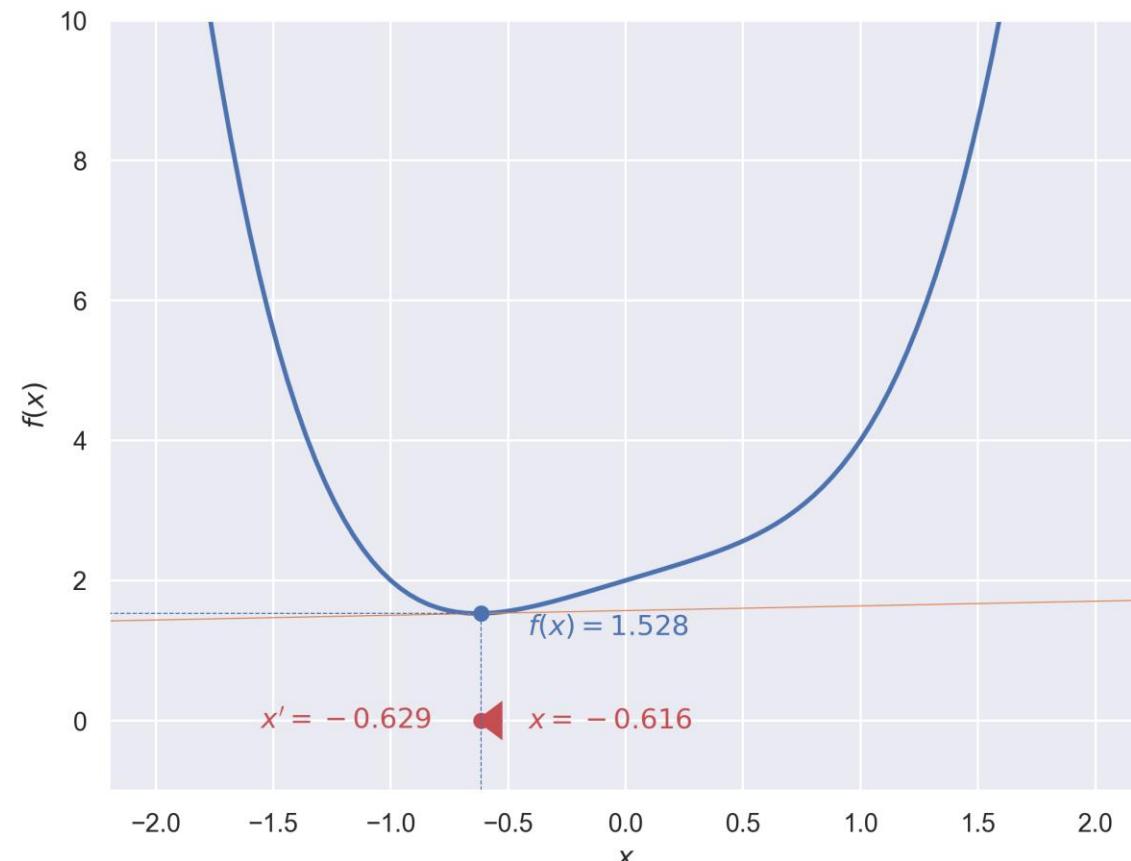
$$f(x) = (-0.616)^4 + (-0.616) + 2 = 1.528$$

$$\frac{df(x)}{dx} = \frac{f(-0.616 + 0.001) - f(-0.616)}{0.001}$$

$$= \frac{1.528 - 1.528}{0.001}$$

$$= 0.067$$

$$x' = -0.616 - 0.2 \cdot 0.067 \\ = -0.629$$



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968
3	-0.395	1.630	0.755
4	-0.546	1.543	0.352
5	-0.616	1.528	0.067
6	-0.629	1.528	0.005

$$x = -0.629$$

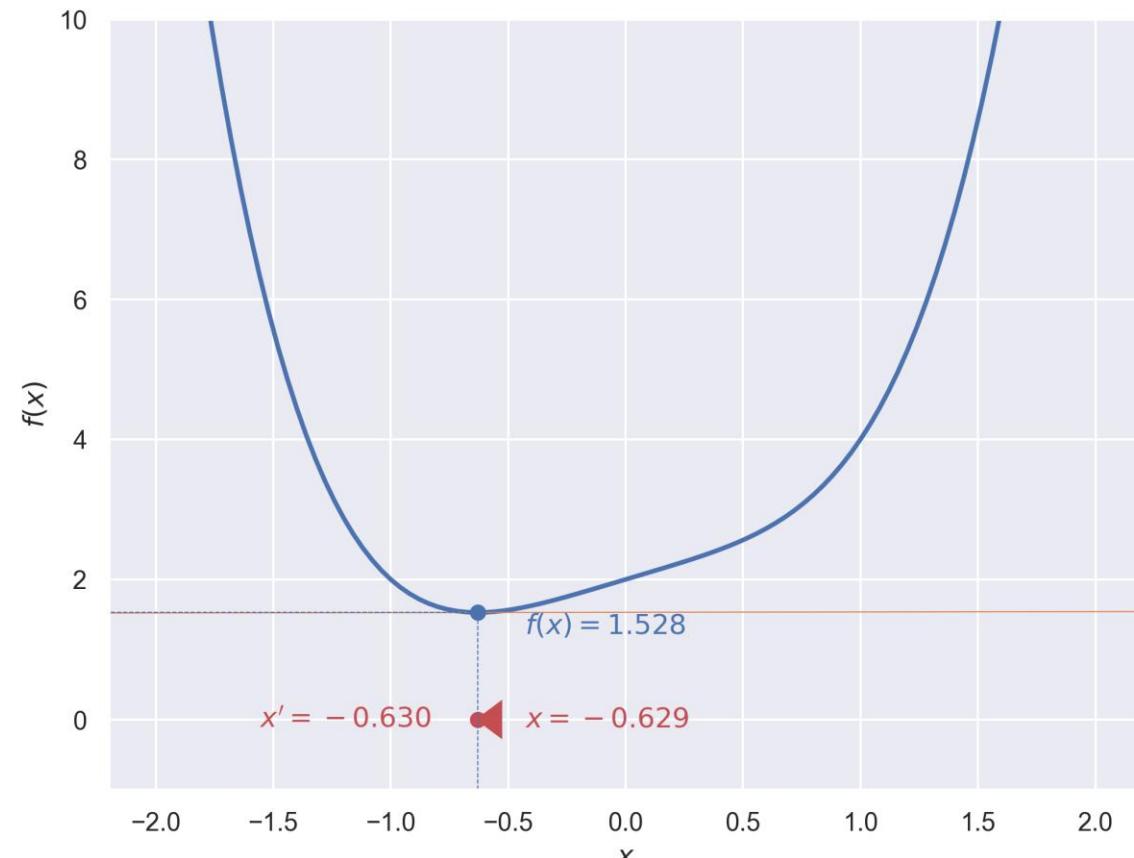
$$f(x) = (-0.629)^4 + (-0.629) + 2 = 1.528$$

$$\frac{df(x)}{dx} = \frac{f(-0.629 + 0.001) - f(-0.629)}{0.001}$$

$$= \frac{1.528 - 1.528}{0.001}$$

$$= 0.005$$

$$x' = -0.629 - 0.2 \cdot 0.005 \\ = -0.630$$



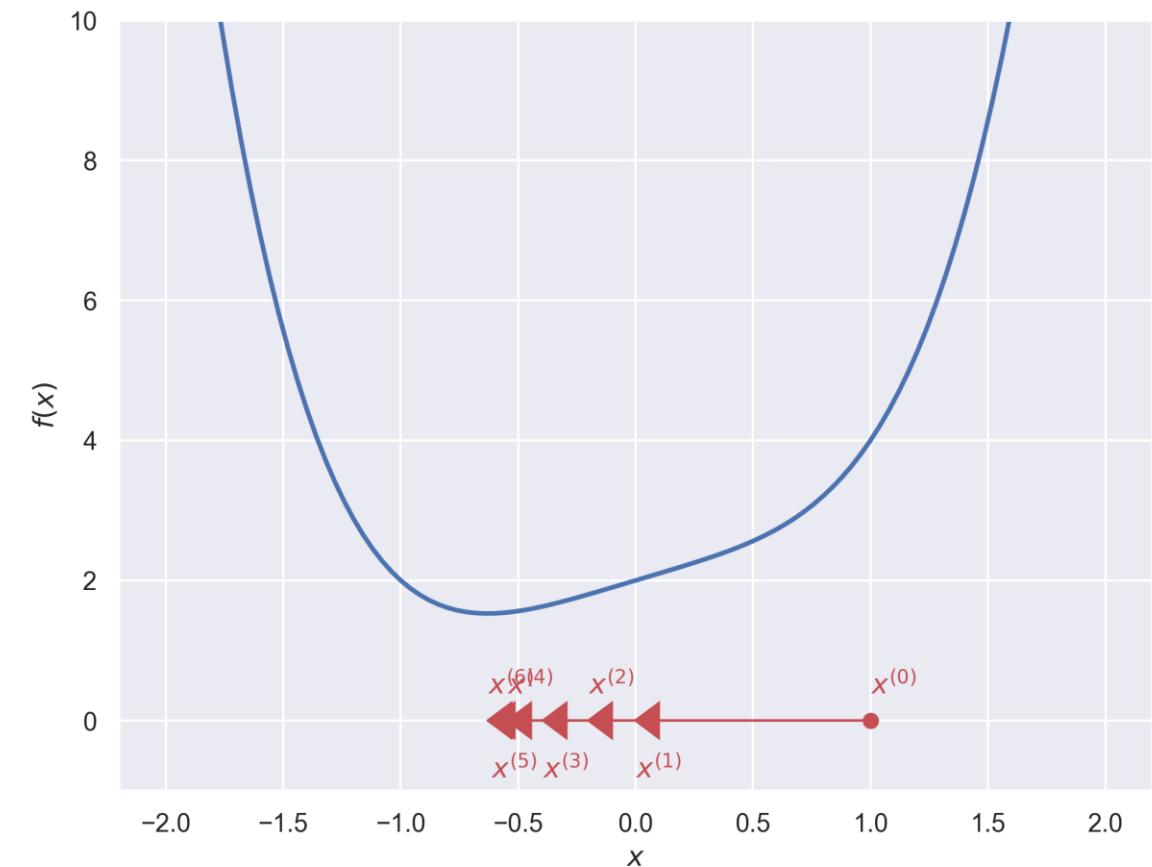
Optimální řešení dle
<https://www.wolframalpha.com/>

$$x^* = \frac{-1}{2^{2/3}} = 0.62996$$

Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968
3	-0.395	1.630	0.755
4	-0.546	1.543	0.352
5	-0.616	1.528	0.067
6	-0.629	1.528	0.005

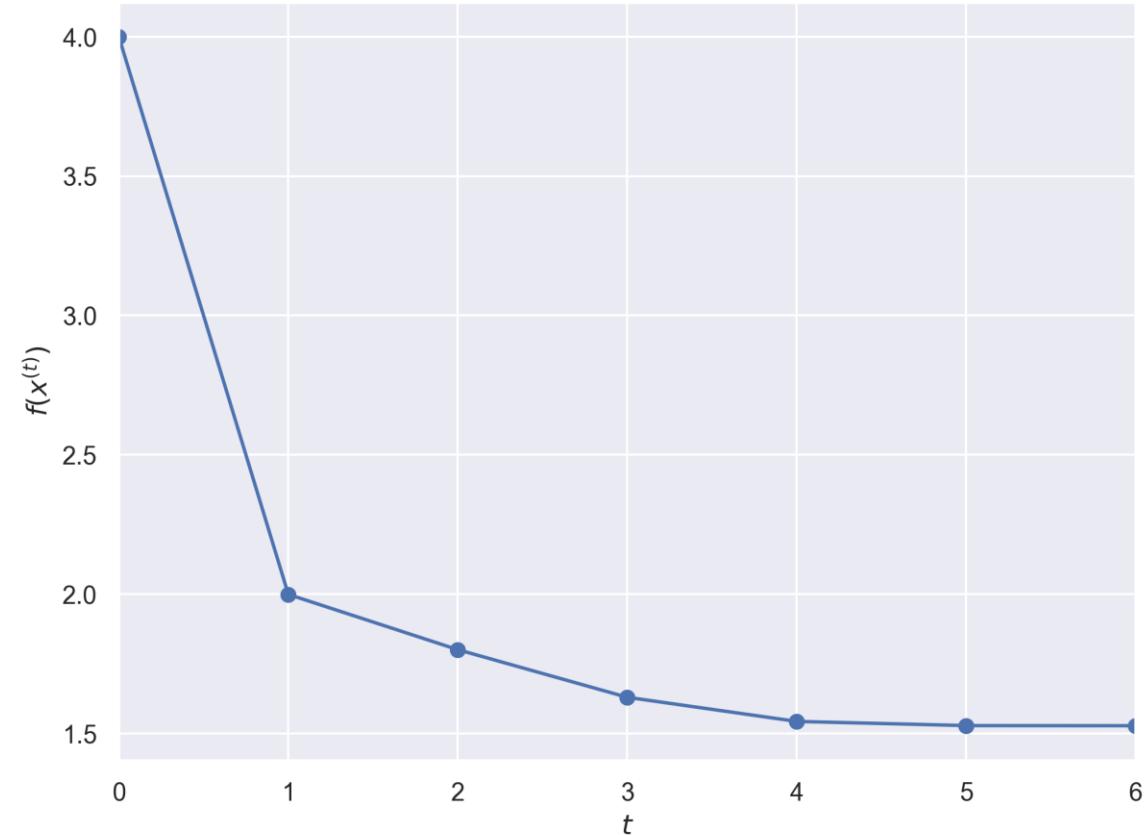
Vizualizace optimálního odhadu v čase



Příklad GD: funkce $f(x) = x^4 + x + 2$

t	x	$f(x)$	df/dx
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968
3	-0.395	1.630	0.755
4	-0.546	1.543	0.352
5	-0.616	1.528	0.067
6	-0.629	1.528	0.005

Průběh lossu (optimalizované funkce) v čase



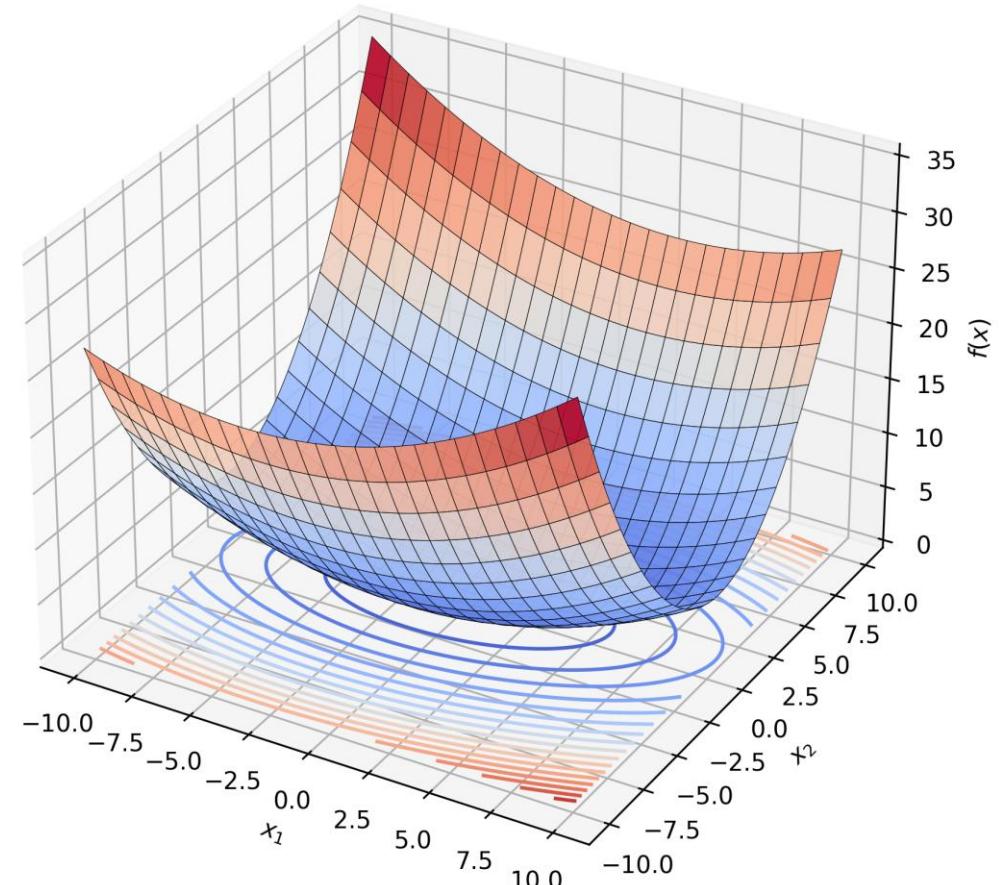
Funkce: n D vstup, 1D výstup

- Trénovaný model má obvykle více než jeden parametr
- Optimalizovaná loss funkce má tedy vstup \mathbf{x} jako vektor obecně s rozměrem D
- Vrací přitom skalární hodnotu (chybovost modelu na datasetu jako jediné číslo)
- Potřebujeme tedy minimalizovat

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$$

kde

$$f(\mathbf{x}): \mathbb{R}^D \rightarrow \mathbb{R}$$
$$\mathbf{x} \in \mathbb{R}^D$$



Derivace funkce: n D vstup, 1D výstup

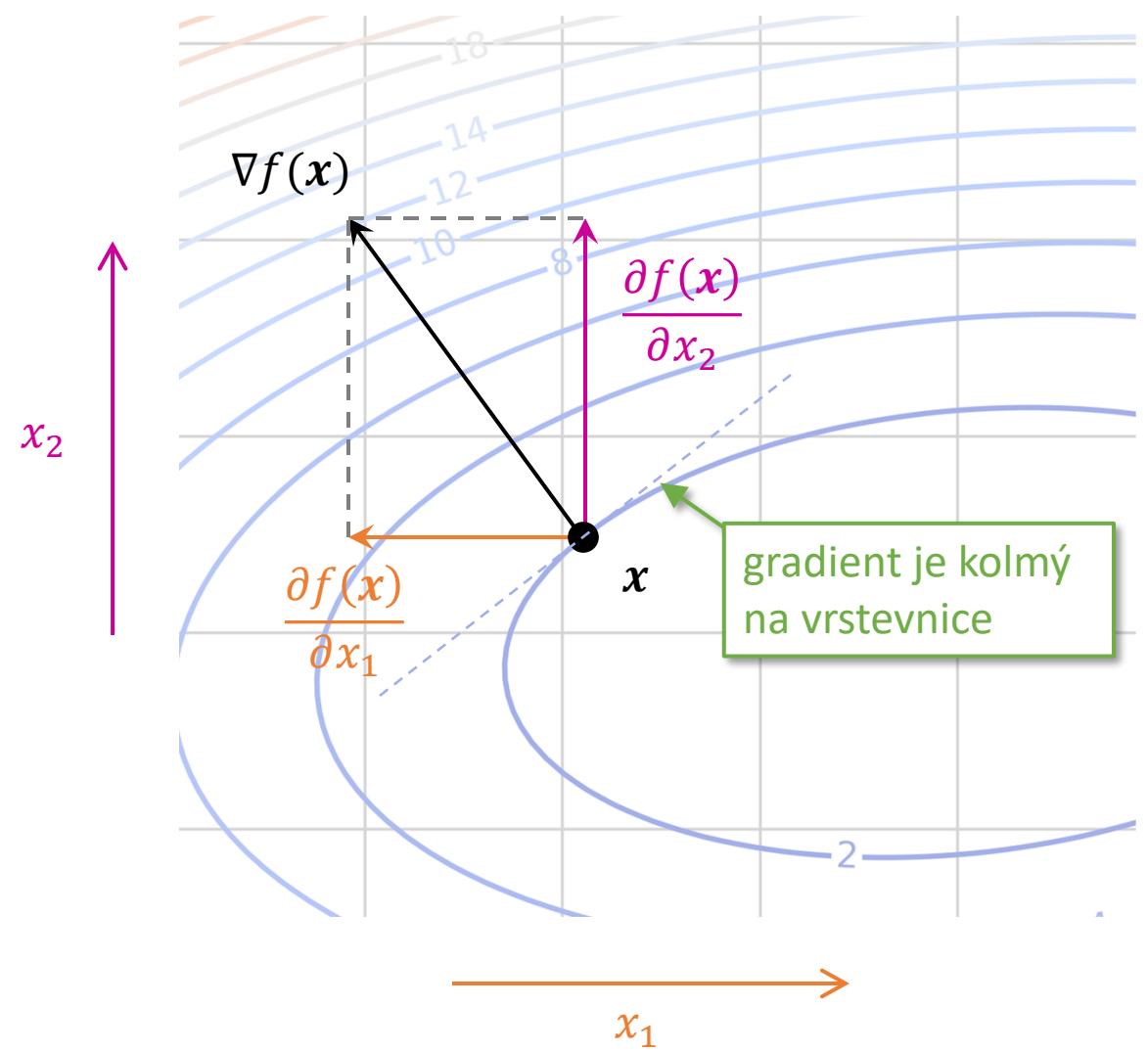
- U funkcí $f(\mathbf{x}): \mathbb{R}^D \rightarrow \mathbb{R}$, tj.

$$f(\mathbf{x}) = f(x_1, \dots, x_D)$$

tedy funkcí s D vstupy a 1 výstupem
můžeme derivovat vzhledem ke každému
z jednotlivých $x_d \rightarrow \text{parciální derivace}$

- Uspořádání všech D parciálních derivací
do vektoru se nazývá gradient:

$$\nabla f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_D} \right]^T$$



Parciální derivace

- U parciální derivace funkce více proměnných vůči jedné z těchto vstupních nezávislých proměnných považujeme ostatní vstupy za konstanty
- Parciální derivaci tedy definujeme jako

$$\frac{\partial f(x_1, \dots, x_D)}{\partial x_d} = \lim_{h \rightarrow 0} \frac{f(x_1, \dots, \cancel{x_d + h}, \dots, x_D) - f(x_1, \dots, x_D)}{h}$$

- Derivace $\partial f / \partial x_d$ nám říká, jak se změní hodnota funkce f , pokud “o trochu” zvětšíme d -tou složku vektoru vstupu x
- Abychom dostali kompletní gradient, výpočet musíme provést pro každou vstupní proměnnou

Metoda největšího spádu (Gradient Descent, GD)

- Metoda největšího spádu se pro vícerozměrné funkce principiálně nemění
- Jediný rozdíl je, že derivaci nahrazuje gradient a pravidlo je vektorové:

$$\textcolor{violet}{x}' := \textcolor{blue}{x} - \textcolor{magenta}{\gamma} \cdot \nabla f(\textcolor{red}{x})$$

kde

původní odhad $\textcolor{blue}{x} \in \mathbb{R}^D$ je vektor s rozměrem D

nový odhad $\textcolor{violet}{x}' \in \mathbb{R}^D$ je vektor s rozměrem D

gradient $\nabla f(\textcolor{red}{x}) \in \mathbb{R}^D$ je vektor s rozměrem D

krok učení (learning rate) $\textcolor{magenta}{\gamma} \in \mathbb{R}$ je skalár

Metoda největšího spádu (Gradient Descent, GD)

- Metoda největšího spádu se pro vícerozměrné funkce principiálně nemění
- Jediný rozdíl je, že derivaci nahrazuje gradient a pravidlo je vektorové:

$$\begin{bmatrix} x'_1 \\ \vdots \\ x'_D \end{bmatrix} := \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} - \gamma \cdot \begin{bmatrix} \frac{\partial f(\mathbf{x})}{x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{x_D} \end{bmatrix}$$

kde

původní odhad $\mathbf{x} \in \mathbb{R}^D$ je vektor s rozměrem D

nový odhad $\mathbf{x}' \in \mathbb{R}^D$ je vektor s rozměrem D

gradient $\nabla f(\mathbf{x}) \in \mathbb{R}^D$ je vektor s rozměrem D

krok učení (learning rate) $\gamma \in \mathbb{R}$ je skalár

Příklad GD: 2D funkce $f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

- Mějme např. funkci

$$f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$$

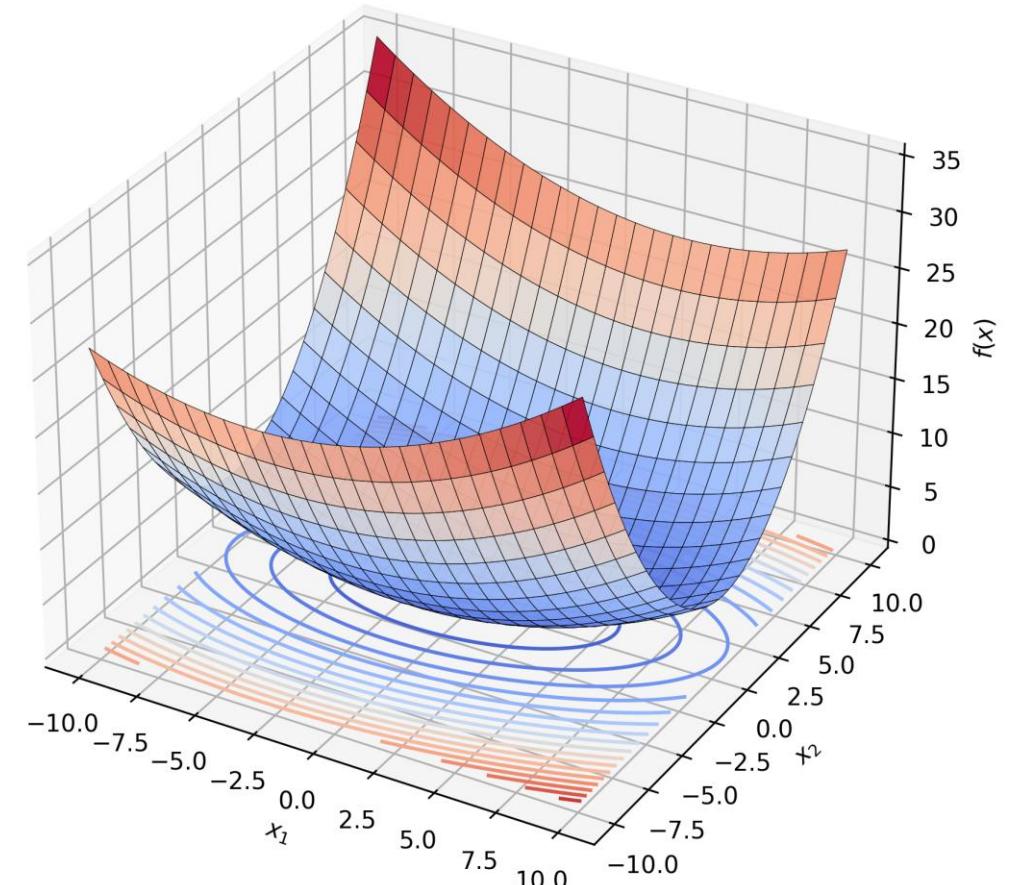
- Chceme nalézt bod $\mathbf{x}^* = [x_1^*, x_2^*]^\top$ takový, ve kterém $f(\mathbf{x})$ nabývá minimální hodnoty, tj.

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$$

- Jako počáteční odhad minima zkusíme

$$\mathbf{x}^{(0)} = \begin{bmatrix} -5.0 \\ -7.5 \end{bmatrix}$$

- Nastavíme $\gamma = 0.5$ a $h = 0.001$



Příklad GD: 2D funkce $f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

- Mějme např. funkci

$$f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$$

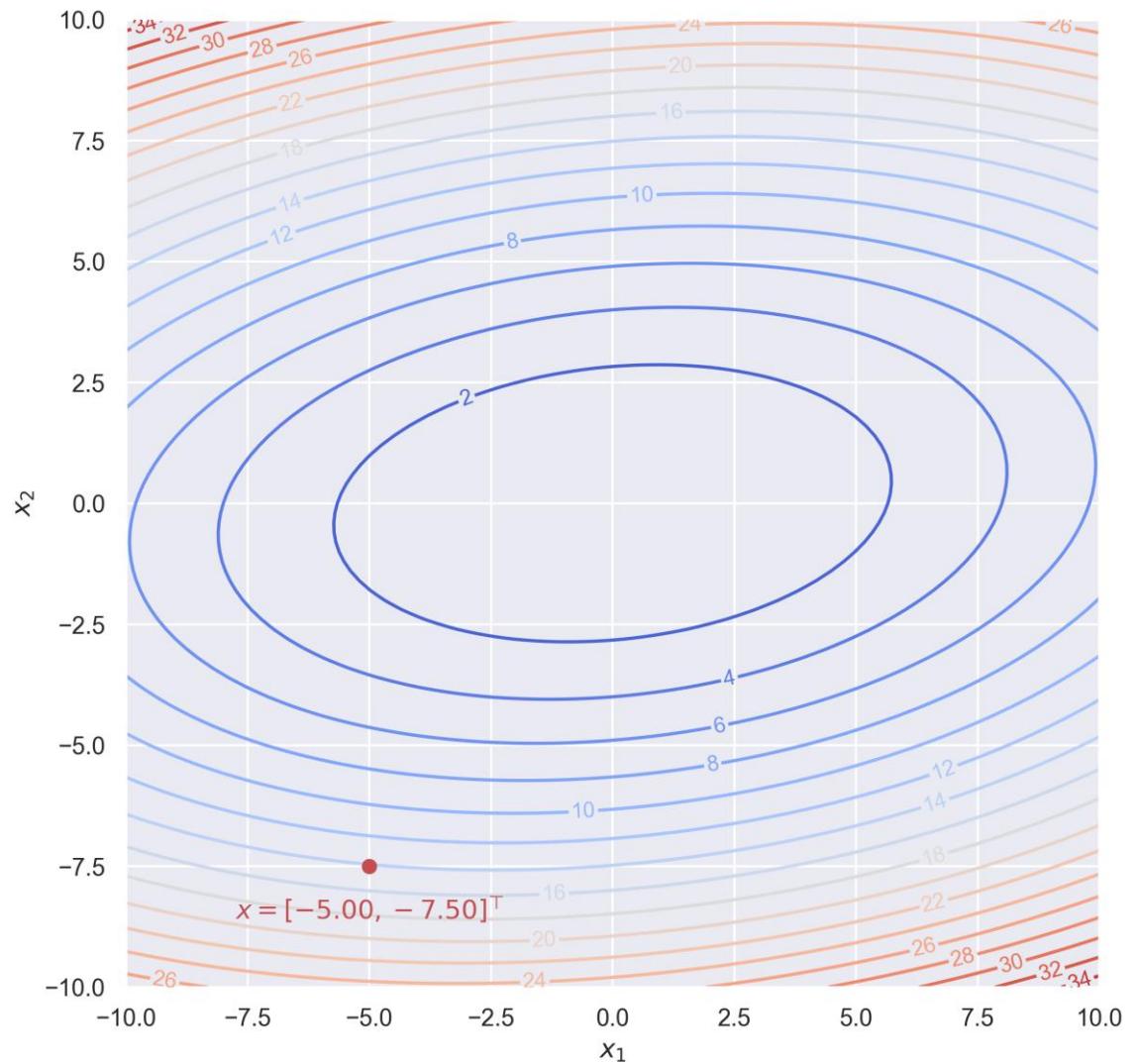
- Chceme nalézt bod $\mathbf{x}^* = [x_1^*, x_2^*]^\top$ takový, ve kterém $f(\mathbf{x})$ nabývá minimální hodnoty, tj.

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$$

- Jako počáteční odhad minima zkusíme

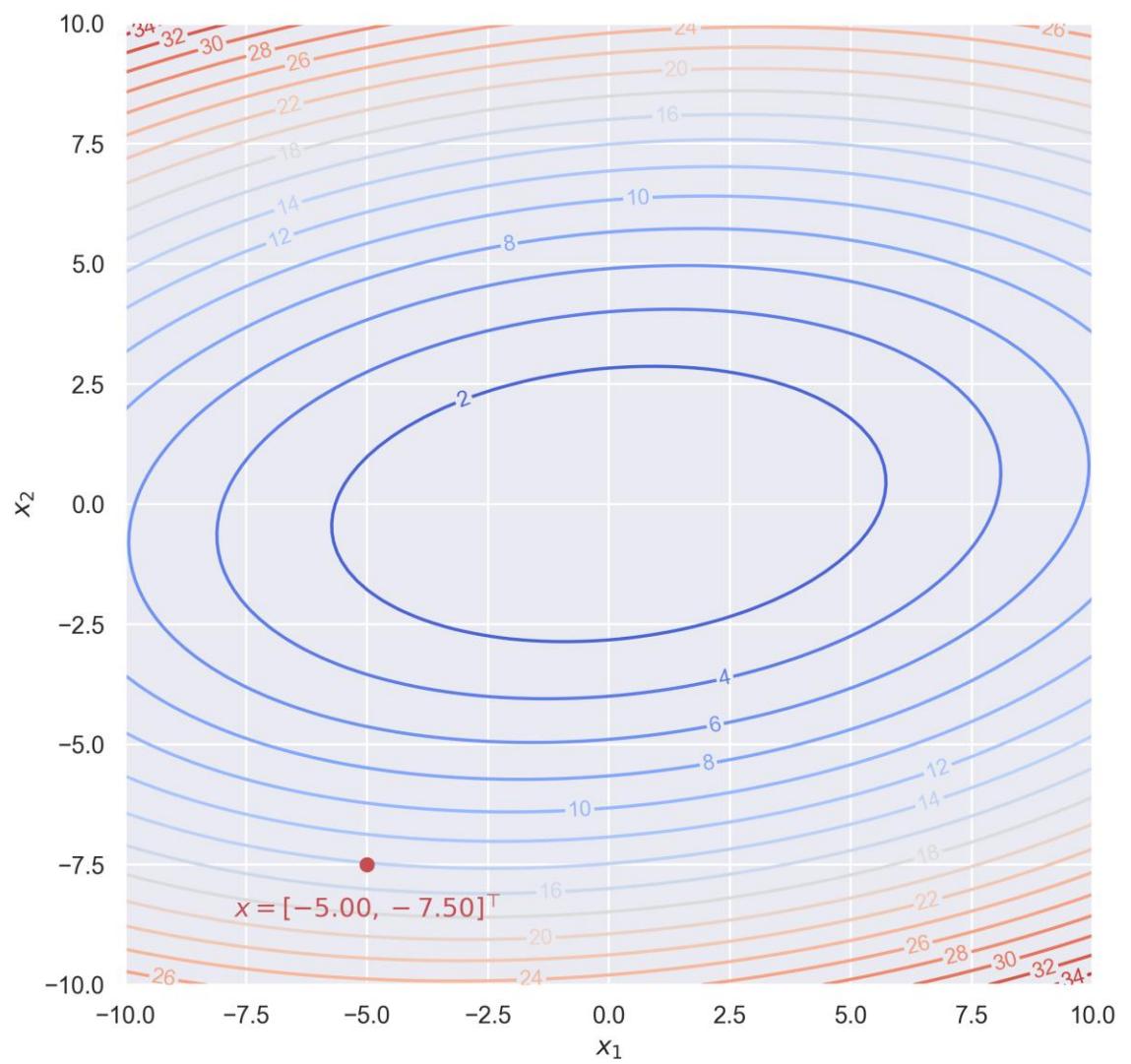
$$\mathbf{x}^{(0)} = \begin{bmatrix} -5.0 \\ -7.5 \end{bmatrix}$$

- Nastavíme $\gamma = 0.5$ a $h = 0.001$



Příklad GD: 2D funkce $f(x) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

t	x_1	x_2	$f(x)$	$\partial f / \partial x_1$	$\partial f / \partial x_2$



Příklad GD: 2D funkce $f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

t	x_1	x_2	$f(\mathbf{x})$	$\partial f / \partial x_1$	$\partial f / \partial x_2$
0	-5.00	-7.50	14.12	-0.32	-3.54

$$\mathbf{x} = [-5.00, -7.50]^\top$$

$$f(\mathbf{x}) = 0.06 \cdot (-5.00)^2 + 0.25 \cdot (-7.50)^2 - 0.04 \cdot (-5.00) \cdot (-7.50) = 14.12$$

$$\frac{\partial f(\mathbf{x})}{\partial x_1} = \frac{f(-5.00 + 0.001, -7.50) - f(-5.00, -7.50)}{0.001}$$

$$= \frac{14.125675 - 14.125}{0.001}$$

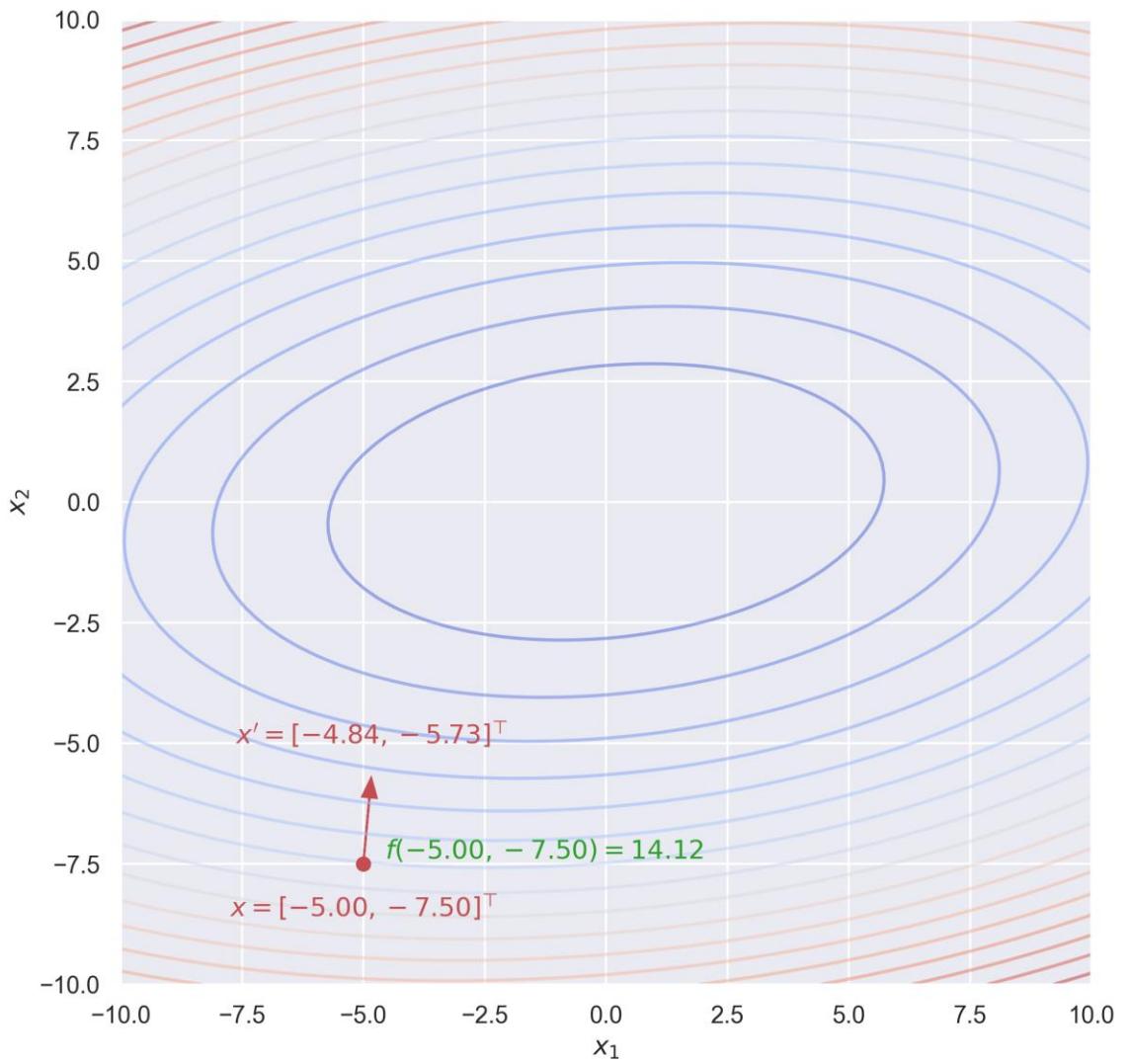
$$= -0.32$$

$$\frac{\partial f(\mathbf{x})}{\partial x_2} = \frac{f(-5.00, -7.50 + 0.001) - f(-5.00, -7.50)}{0.001}$$

$$= \frac{14.121 - 14.125}{0.001}$$

$$= -3.54$$

$$\begin{aligned} \mathbf{x}' &= [-5.00, -7.50]^\top - 0.5 \cdot [-0.32, -3.54]^\top \\ &= [-4.84, -5.73]^\top \end{aligned}$$



Příklad GD: 2D funkce $f(x) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

t	x_1	x_2	$f(x)$	$\partial f / \partial x_1$	$\partial f / \partial x_2$
0	-5.00	-7.50	14.12	-0.32	-3.54
1	-4.84	-5.73	8.55	-0.38	-2.67

$$x = [-4.84, -5.73]^\top$$

$$f(x) = 0.06 \cdot (-4.84)^2 + 0.25 \cdot (-5.73)^2 - 0.04 \cdot (-4.84) \cdot (-5.73) = 8.55$$

$$\frac{\partial f(x)}{\partial x_1} = \frac{f(-4.84 + 0.001, -5.73) - f(-4.84, -5.73)}{0.001}$$

$$= \frac{8.5487 - 8.549}{0.001}$$

$$= -0.38$$

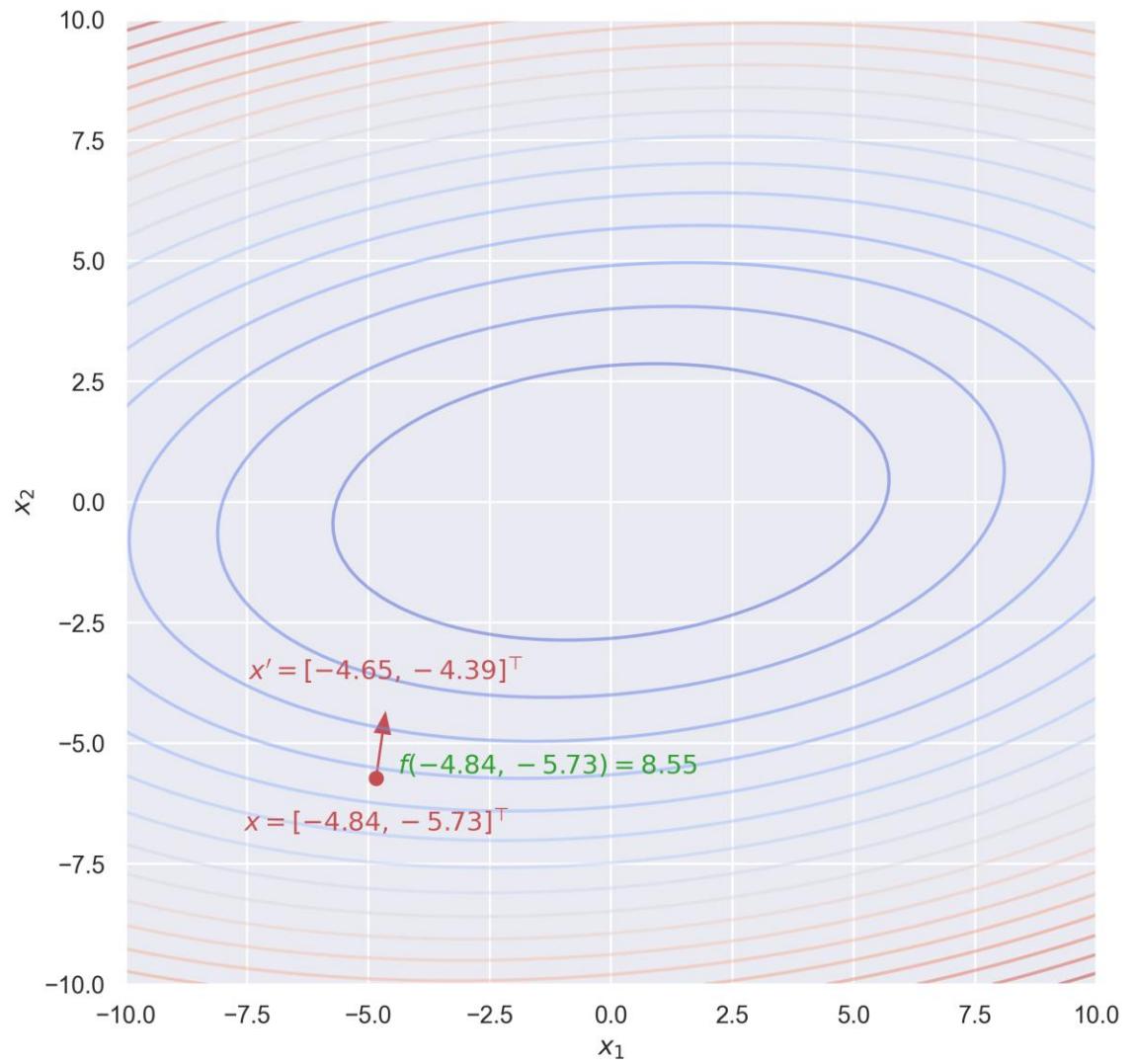
$$\frac{\partial f(x)}{\partial x_2} = \frac{f(-4.84, -5.73 + 0.001) - f(-4.84, -5.73)}{0.001}$$

$$= \frac{8.546 - 8.549}{0.001}$$

$$= -2.67$$

$$x' = [-4.84, -5.73]^\top - 0.5 \cdot [-0.38, -2.67]^\top$$

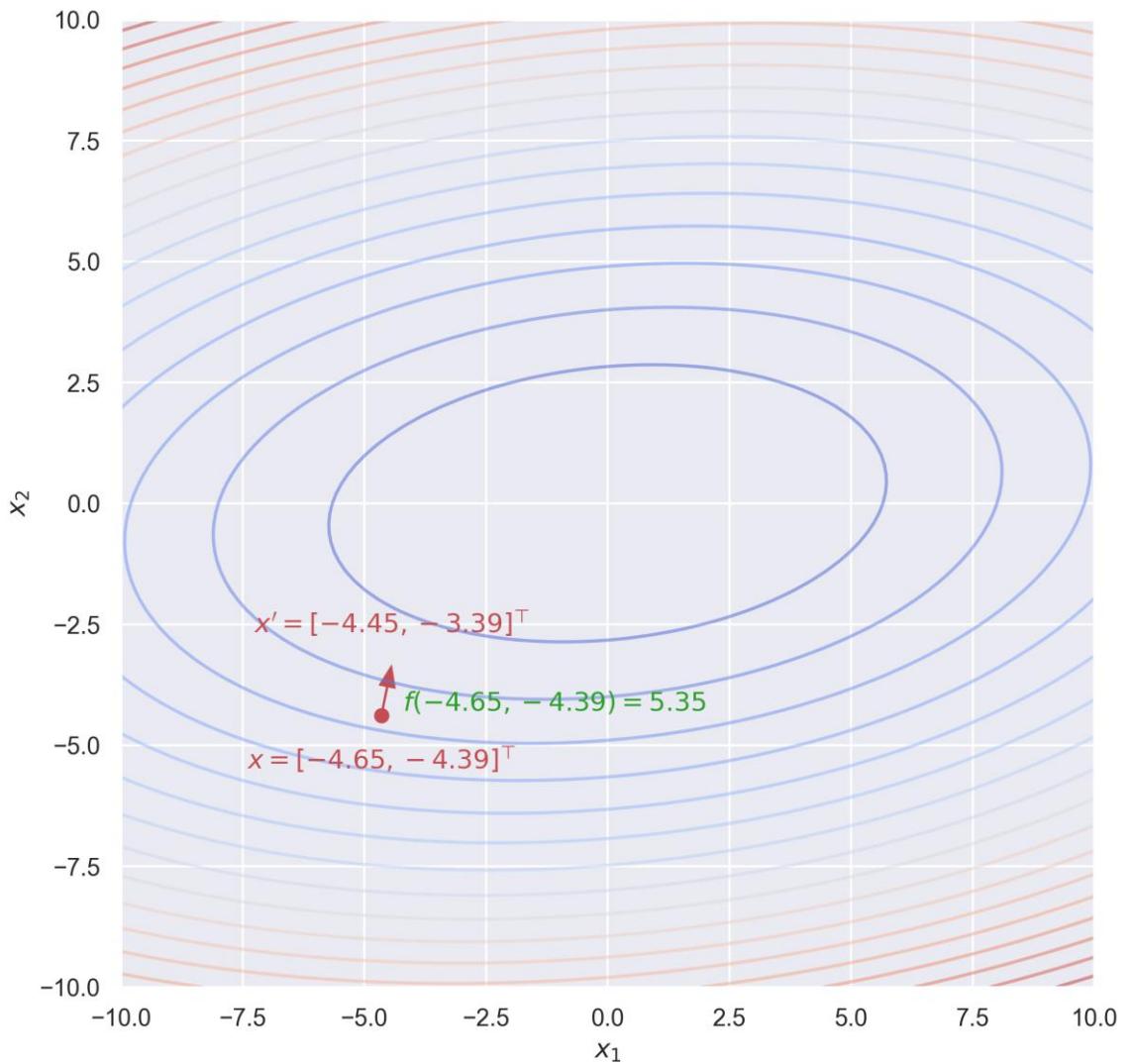
$$= [-4.65, -4.39]^\top$$



Příklad GD: 2D funkce $f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

t	x_1	x_2	$f(\mathbf{x})$	$\partial f / \partial x_1$	$\partial f / \partial x_2$
0	-5.00	-7.50	14.12	-0.32	-3.54
1	-4.84	-5.73	8.55	-0.38	-2.67
...

$$\begin{aligned} \mathbf{x} &= [-4.65, -4.39]^\top \\ f(\mathbf{x}) &= \dots = 8.55 \\ \frac{\partial f(\mathbf{x})}{\partial x_1} &= \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h} \\ &= \dots \\ &= \frac{0.001}{0.001} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} &= \frac{f(x_1, x_2 + h) - f(x_1, x_2)}{h} \\ &= \dots \\ &= \frac{0.001}{0.001} \\ \mathbf{x}' &= \mathbf{x} - 0.5 \cdot [\partial f / \partial x_1, \partial f / \partial x_2]^\top \\ &= \dots \end{aligned}$$



Příklad GD: 2D funkce $f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

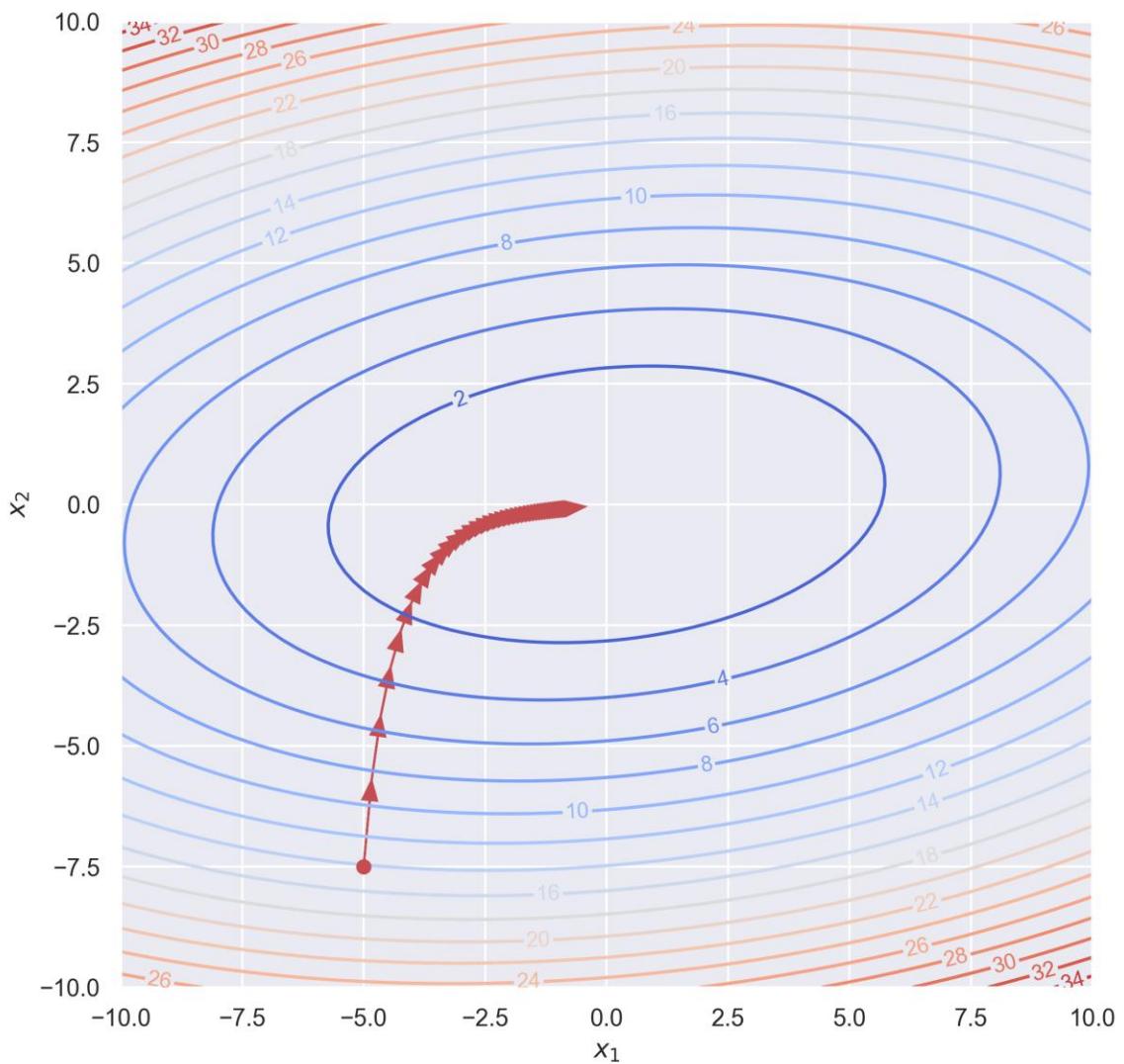
t	x_1	x_2	$f(x)$	$\partial f / \partial x_1$	$\partial f / \partial x_2$
0	-5.00	-7.50	14.12	-0.32	-3.54
1	-4.84	-5.73	8.55	-0.38	-2.67
...
40	-0.45	-0.05	0.01	-0.06	-0.01

- Odhad minima po 40 iteracích:

$$\mathbf{x} = [-0.45, -0.05]^T$$

- Skutečné minimum je

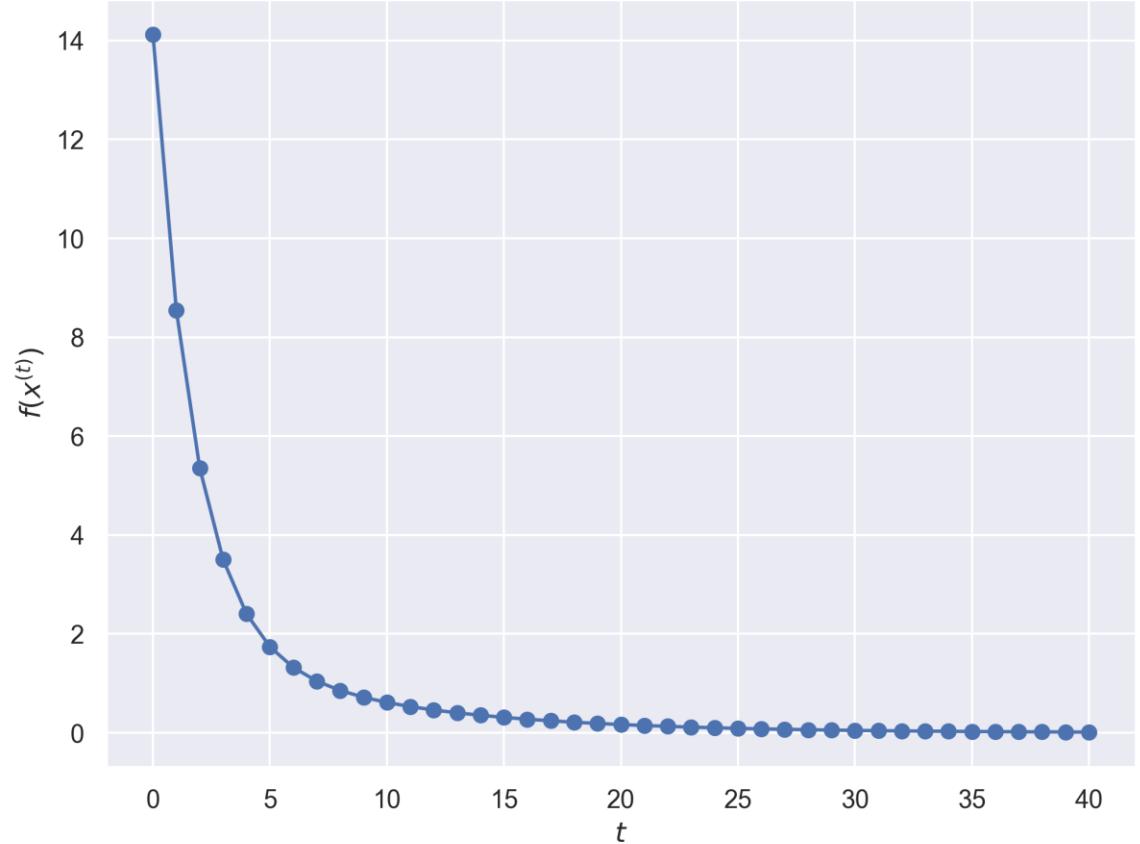
$$\mathbf{x}^* = [0, 0]^T$$



Příklad GD: 2D funkce $f(\mathbf{x}) = 0.06x_1^2 + 0.25x_2^2 - 0.04x_1x_2$

t	x_1	x_2	$f(\mathbf{x})$	$\partial f / \partial x_1$	$\partial f / \partial x_2$
0	-5.00	-7.50	14.12	-0.32	-3.54
1	-4.84	-5.73	8.55	-0.38	-2.67
...
40	-0.45	-0.05	0.01	-0.06	-0.01

Křivka lossu v čase



Příklad GD: lineární softmax cross entropy na CIFAR-10

- Optimalizovaná funkce má pro lineární klasifikátor se softmaxem a křížovou entropií formu

$$L(w_{1,1}, \dots, w_{K,D}, b_1, \dots, b_K) = -\frac{1}{N} \sum_{n=1}^N \log \frac{\exp(w_{y_n,:} \cdot x_n + b_{y_n})}{\sum_{k=1}^K \exp(w_{k,:} \cdot x_n + b_k)}$$

- Chceme nalézt bod $\theta^* = [w_{1,1}^*, \dots, w_{K,D}^*, b_1^*, \dots, b_K^*]^\top$ takový, ve kterém $L(\theta)$ nabývá minimální hodnoty, tj.

$$\theta^* = \operatorname{argmin}_{\theta} L(\theta)$$

- Data tedy považujeme za konstantu
- Jako počáteční odhad minima zkusíme
- Nastavíme $\gamma = 0.2$ (learning rate) a $h = 0.001$

$$\theta^{(0)} \sim \mathcal{N}(0, 0.001)$$

inicializujeme na náhodné hodnoty z normálního (gaussovského) rozdělení s nulovým průměrem a std. odch. 0.001

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix}$$

máme současný odhad vah

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} ? & ? & \dots & ? \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

chceme gradient na váhy, abychom mohli provést update

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} ? & ? & \dots & ? \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix} \longrightarrow L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$\mathbf{h} = \begin{bmatrix} 0.001 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

k $w_{1,1}$ přičteme $h_{1,1}$, tj. zkusíme váhy,
kde $w_{1,1}$ má hodnotu $0.001764 + 0.001$
a zjistíme loss

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} ? & ? & \dots & ? \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$\mathbf{h} = \begin{bmatrix} 0.001 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w} + \mathbf{h}, \mathbf{b}) = 2.3059561$$

k $w_{1,1}$ přičteme $h_{1,1}$, tj. zkusíme váhy,
kde $w_{1,1}$ má hodnotu $0.001764 + 0.001$
a zjistíme loss

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} ? & ? & \dots & ? \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix} \longrightarrow L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$\mathbf{h} = \begin{bmatrix} 0.001 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \longrightarrow L(\mathbf{w} + \mathbf{h}, \mathbf{b}) = 2.3059561$$

vypočteme paricální derivaci vůči $w_{1,1}$

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} -0.0057 & ? & \dots & ? \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

$$\begin{aligned} \frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial w_{1,1}} &\approx \frac{L(\mathbf{w} + \mathbf{h}, \mathbf{b}) - L(\mathbf{w})}{h} \\ &\approx \frac{2.3059561 - 2.3059618}{0.001} \\ &\approx -0.0057 \end{aligned}$$

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & \textcolor{red}{4.00} & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$\mathbf{h} = \begin{bmatrix} 0 & \textcolor{red}{0.001} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w} + \mathbf{h}, \mathbf{b}) = 2.3059519$$

opakujeme pro $w_{k,d} \dots$

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} -0.0057 & \textcolor{red}{-0.01} & \dots & ? \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

$$\begin{aligned} \frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial w_{1,2}} &\approx \frac{L(\mathbf{w} + \mathbf{h}, \mathbf{b}) - L(\mathbf{w})}{h} \\ &\approx \frac{2.3059519 - 2.3059618}{0.001} \\ &\approx -0.01 \end{aligned}$$

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$\mathbf{w} = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & \color{red}{9.79} \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.06 & -1.81 & \dots & -15.21 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$\mathbf{h} = \begin{bmatrix} 0 & 0 & \dots & \color{red}{0.001} \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w} + \mathbf{h}, \mathbf{b}) = 2.3059526$$

opakujeme pro $w_{k,d} \dots$

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} -0.0057 & -0.01 & \dots & \color{red}{-0.0093} \\ ? & ? & \dots & ? \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

$$\begin{aligned} \frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial w_{1,3072}} &\approx \frac{L(\mathbf{w} + \mathbf{h}, \mathbf{b}) - L(\mathbf{w})}{h} \\ &\approx \frac{2.3059526 - 2.3059618}{0.001} \\ &\approx -0.0093 \end{aligned}$$

Příklad GD: lineární softmax cross entropy na CIFAR-10

$$w = 10^{-4} \cdot \begin{bmatrix} 17.64 & 4.00 & \dots & 9.79 \\ -18.24 & -5.79 & \dots & -2.71 \\ \vdots & \vdots & \ddots & \vdots \\ 12.0 & & & \end{bmatrix} \quad \longrightarrow \quad L(\mathbf{w}, \mathbf{b}) = 2.3059618$$

$$h = \begin{bmatrix} 0 & 0 & \dots \\ 0 & 0 & \dots \\ \vdots & \vdots & \ddots \\ 0 & 0 & \dots \end{bmatrix}$$
$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \begin{bmatrix} -0.0001 & 0 & \dots & 0 \\ 0 & -0.0001 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ ? & ? & \dots & ? \end{bmatrix}$$

• Postup musíme provést celkem $(K \cdot D + K)$ -krát (všechny váhy + všechny biasy)
• Pro každou parciální derivaci přitom musíme znova vyhodnotit loss na celém datasetu
• U vícevrstvých sítí by počet parametrů byl ještě mnohem vyšší než $K \cdot (D + 1)$
• V praxi se proto numerická derivace nepoužívá
• Potřebujeme analytický vztah pro přímý výpočet celého gradientu ∇L najednou

$\frac{L(\mathbf{w}, \mathbf{b}) - L(\mathbf{w})}{h} \approx -0.0093$

Gradient lineárního softmaxu a křížové entropie analyticky

Optimalizovaná funkce (loss) je

$$L(\mathbf{w}, \mathbf{b}) = -\frac{1}{N} \sum_{n=1}^N \log \left[\frac{\exp(\mathbf{w}_{y_n,:} \cdot \mathbf{x}_n + b_{y_n})}{\sum_{k=1}^K \exp(\mathbf{w}_{k,:} \cdot \mathbf{x}_n + b_k)} \right] \hat{p}_{n,y_n}$$

Potřebujeme

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = ? \quad \longrightarrow$$



$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{b}} = ? \quad \longrightarrow$$



$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \frac{1}{N} \sum_{n=1}^N (K \times D) (\hat{\mathbf{p}}_n - \mathbf{p}_n) \cdot \mathbf{x}_n^\top$$



$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{b}} = \frac{1}{N} \sum_{n=1}^N (K \times 1) (\hat{\mathbf{p}}_n - \mathbf{p}_n)$$

$K \times 1$

Gradient descent (GD) pro multiclass logistickou regresi

Incializujeme:

$$\begin{aligned} w_{kd} &\sim \mathcal{N}(0, 0.001) \\ b_k &\sim \mathcal{N}(0, 0.001) \end{aligned}$$

Opakujeme:

$$\hat{p}_n := \frac{\exp(w_{y_n,:} \cdot x_n + b_{y_n})}{\sum_{k=1}^K \exp(w_{k,:} \cdot x_n + b_k)}$$

$$L(\mathbf{w}, \mathbf{b}) := -\frac{1}{N} \sum_{n=1}^N \log p_{n,y_n}$$

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} := \frac{1}{N} \sum_{n=1}^N (\hat{p}_n - p_n) \cdot x_n^\top$$

$$\frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{b}} := \frac{1}{N} \sum_{n=1}^N (\hat{p}_n - p_n)$$

$$\mathbf{w} := \mathbf{w} - \gamma \cdot \frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}}$$

$$\mathbf{b} := \mathbf{b} - \gamma \cdot \frac{\partial L(\mathbf{w}, \mathbf{b})}{\partial \mathbf{b}}$$

```
1. w = 1e-3 * np.random.randn(10, 3072)
2. b = 1e-3 * np.random.randn(10)
3. for t in range(200):
4.     l = 0.
5.     dw = np.zeros_like(w) # (K, D)
6.     db = np.zeros_like(b) # (K, )
7.     ids = np.random.permutation(len(X))[:bs]
8.     for n, (xn, yn) in enumerate(zip(X[ids], Y[ids])):
9.         # Loss
10.        sn = np.dot(w, xn) + b # (K, )
11.        pn = np.exp(sn) / np.sum(np.exp(sn)) # (K, )
12.        ln = -np.log(pn[yn])
```

```
13.
14.    # gradients
15.    dbn = pn.copy() # (K, )
16.    dbn[yn] -= 1
17.    dwn = np.dot(dbn.reshape(-1, 1), xn.reshape(1, -1)) # (K, D)
```

```
18.
19.    # accumulate
20.    l += ln
21.    dw += dwn
22.    db += dbn
23.
24.    # average
25.    l /= n + 1
26.    dw /= n + 1
27.    db /= n + 1
28.
29.    # update
30.    w -= lr * dw
31.    b -= lr * db
```

Gradient descent (GD) pro multiclass logistickou regresi

Incializujeme:

- $\theta = \{W, b\}$ na náhodné hodnoty

Opakujeme:

1. Pro každý vzorek x_n v trénovací sadě x_1, \dots, x_N
 - a. predikujeme pravděpodobnosti \hat{p}_n
 - b. vypočteme dílčí kritérium l_n a akumulujeme k celkovému l
 - c. vypočteme dílčí gradient ∇L_n a akumulujeme k celkovému ∇L
2. updatujeme parametry θ akumulovaným gradientem ∇L s krokem γ

Zastavíme:

- po fixním počtu iterací
- parametry θ se ustálí
- hodnota kritéria $L(\theta)$ již delší dobu neklesá

```
1. w = 1e-3 * np.random.randn(10, 3072)
2. b = 1e-3 * np.random.randn(10)
3. for t in range(200):
4.     l = 0.
5.     dw = np.zeros_like(w) # (K, D)
6.     db = np.zeros_like(b) # (K, )
7.     ids = np.random.permutation(len(X))[:bs]
8.     for n, (xn, yn) in enumerate(zip(X[ids], Y[ids])):
9.         # Loss
10.        sn = np.dot(w, xn) + b # (K, )
11.        pn = np.exp(sn) / np.sum(np.exp(sn)) # (K, )
12.        ln = -np.log(pn[yn])
13.
14.        # gradients
15.        dbn = pn.copy() # (K, )
16.        dbn[yn] -= 1
17.        dwn = np.dot(dbn.reshape(-1, 1), xn.reshape(1, -1)) # (K, D)
18.
19.        # accumulate
20.        l += ln
21.        dw += dwn
22.        db += dbn
23.
24.        # average
25.        l /= n + 1
26.        dw /= n + 1
27.        db /= n + 1
28.
29.        # update
30.        w -= lr * dw
31.        b -= lr * db
```

Gradient descent (GD) pro multiclass logistickou regresi

Incializujeme:

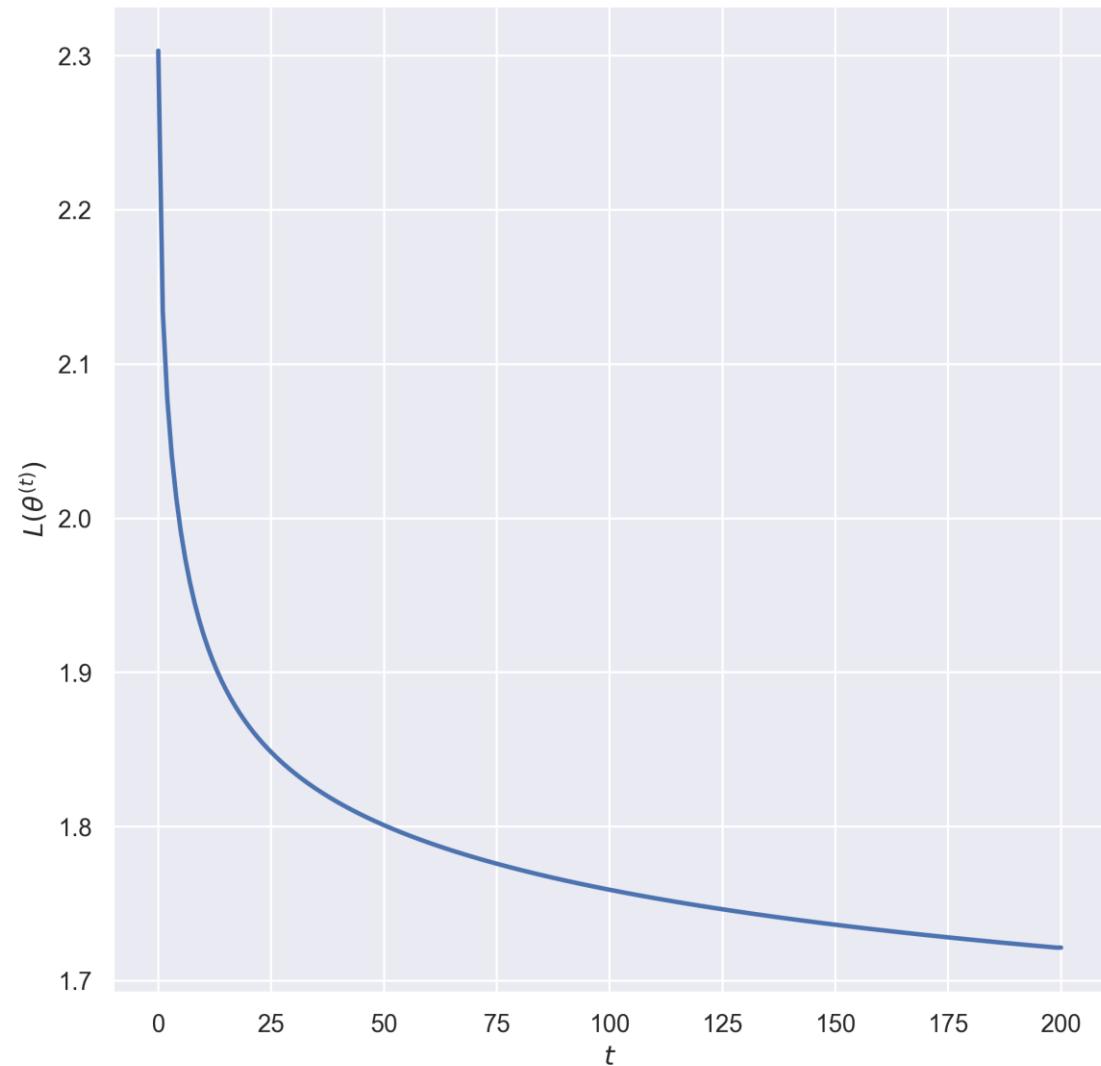
- $\theta = \{W, b\}$ na náhodné hodnoty

Opakujeme:

1. Pro každý vzorek x_n v trénovací sadě x_1, \dots, x_N
 - a. predikujeme pravděpodobnosti \hat{p}_n
 - b. vypočteme dílčí kritérium l_n a akumulujeme k celkovému l
 - c. vypočteme dílčí gradient ∇L_n a akumulujeme k celkovému ∇L
2. updatujeme parametry θ akumulovaným gradientem ∇L s krokem γ

Zastavíme:

- po fixním počtu iterací
- parametry θ se ustálí
- hodnota kritéria $L(\theta)$ již delší dobu neklesá



Gradient descent (GD) pro multiclass logistickou regresi

Inicializujeme:

- $\theta = \{W, b\}$ na náhodné hodnoty

jak budeme inicializovat?

Opakujeme:

1. Pro každý vzorek x_n v trénovací sadě x_1, \dots, x_N
 - a. predikujeme pravděpodobnosti \hat{p}_n
 - b. vypočteme dílčí kritérium l_n a akumulujeme k celkovému l
 - c. vypočteme dílčí gradient ∇L_n a akumulujeme k celkovému ∇L
2. updatujeme parametry θ akumulovaným gradientem ∇L s krokem γ

hyperparametry

jaký krok učení?

Zastavíme:

- po fixním počtu iterací
- parametry θ se ustálí
- hodnota kritéria $L(\theta)$ již delší dobu neklesá

kolik iterací? jak dlouho?

Stochastic Gradient Descent (SGD)

Minibatch Gradient Descent

- Uvedený postup je velmi neefektivní – vždy je nutné vyhodnotit celý dataset
- Např. ImageNet má přitom cca 14 milionů obrázků
- Loss l na datasetu X si můžeme představit jako odhad skutečné střední hodnoty lossu

$$E_{X \sim P_{\text{data}}}(L(X, \theta)) \approx \frac{1}{N} \sum_{n=1}^N L_n(x_n, y_n, \theta)$$

pokud bychom měli všechna relevantní data v mnohovesmíru a pravděpodobnost jejich výskytu popsanou rozdělením P_{data} → vlastně i dataset je pouhým vzorkem reality

- Možná na odhad stř. hodnoty lossu potažmo gradientů stačí i mnohem menší vzorek (**minibatch**). Možná není nutné vidět N obrázků, ale pouze B obrázků, kde $B \ll N$
- Takto vznikne tzv. **Minibatch Gradient Descent**
- Pokud pouze jeden vzorek → **Stochastic Gradient Descent (SGD)**
 - https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Varianty GD a názvosloví dle velikosti batche

velikost batche B	různé názvy pro totéž
	Gradient descent
$B = N$	Batch gradient descent Steepest descent
$1 < B \ll N$	Minibatch gradient descent
$B = 1$	Stochastic gradient descent (SGD) Online gradient descent Incremental gradient descent

- Aby to nebylo jednoduché, obvykle minibatch = batch
- Minibatch gradient descent najdeme v knihovnách pod jménem Stochastic gradient descent (SGD) s volitelným batch_size parametrem (B)
- „Pořádek je pro blbce, intelligent zvládá chaos.“

Stochastic Gradient descent (SGD) pro multiclass logistickou regresi

Inicializujeme:

- $\theta = \{W, b\}$ na náhodné hodnoty

← jak budeme inicializovat?

Opakujeme:

1. Pro každý vzorek x_n v navzorkované dívce x_1, \dots, x_B
 - a. predikujeme pravděpodobnosti \hat{p}_n
 - b. vypočteme dílčí kritérium l_n a akumulujeme k celkovému l
 - c. vypočteme dílčí gradient ∇L_n a akumulujeme k celkovému ∇L
2. updatujeme parametry θ akumulovaným gradientem ∇L s krokem γ

← jak budeme vzorkovat?

← jak velké B (batch size)?

hyperparametry

← jaký krok učení?

Zastavíme:

- po fixním počtu iterací
- parametry θ se ustálí
- hodnota kritéria $L(\theta)$ již delší dobu neklesá

← kolik iterací? jak dlouho?

Stochastic Gradient descent (SGD) pro multiclass logistickou regresi

Incializujeme:

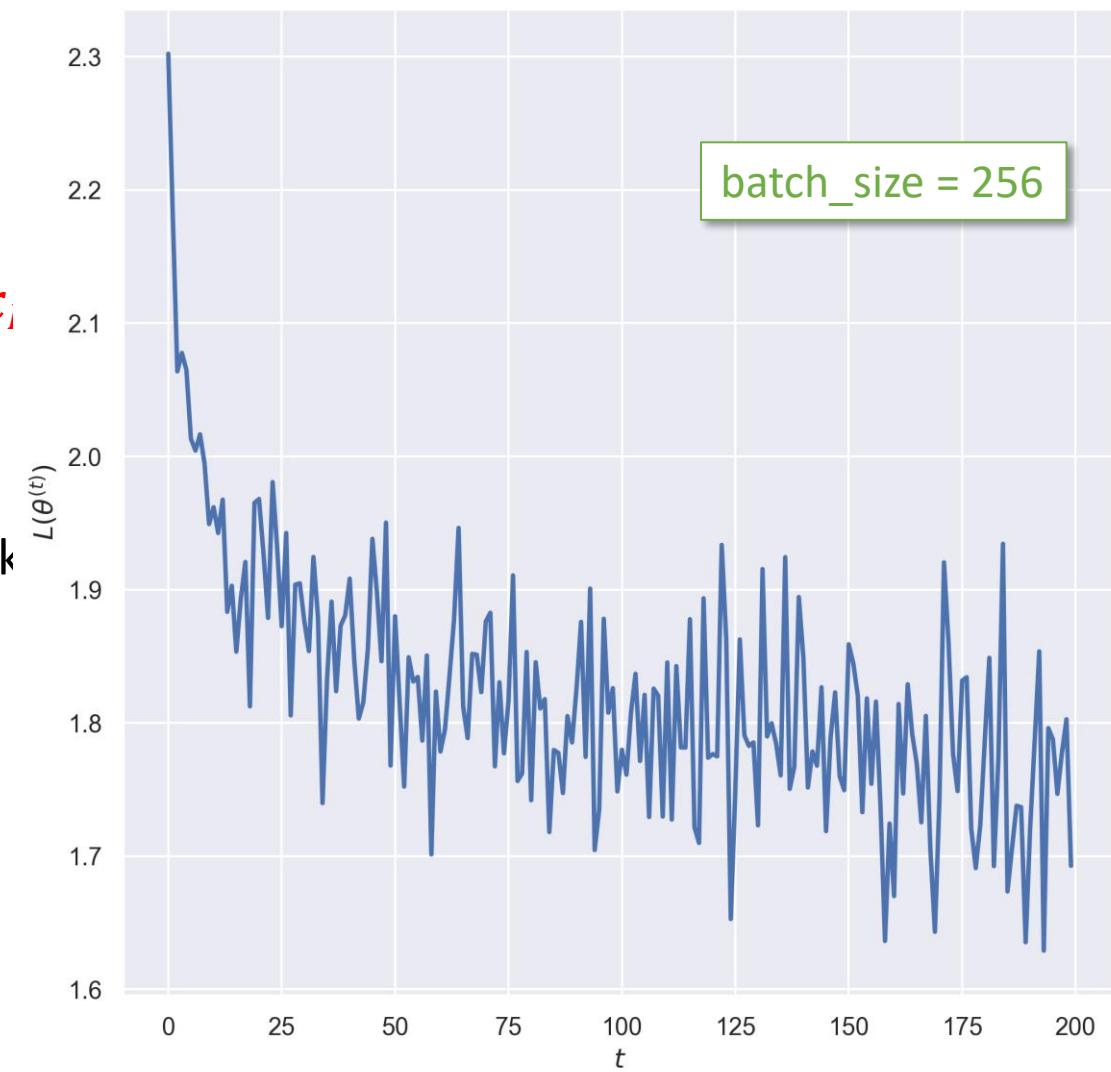
- $\theta = \{W, b\}$ na náhodné hodnoty

Opakujeme:

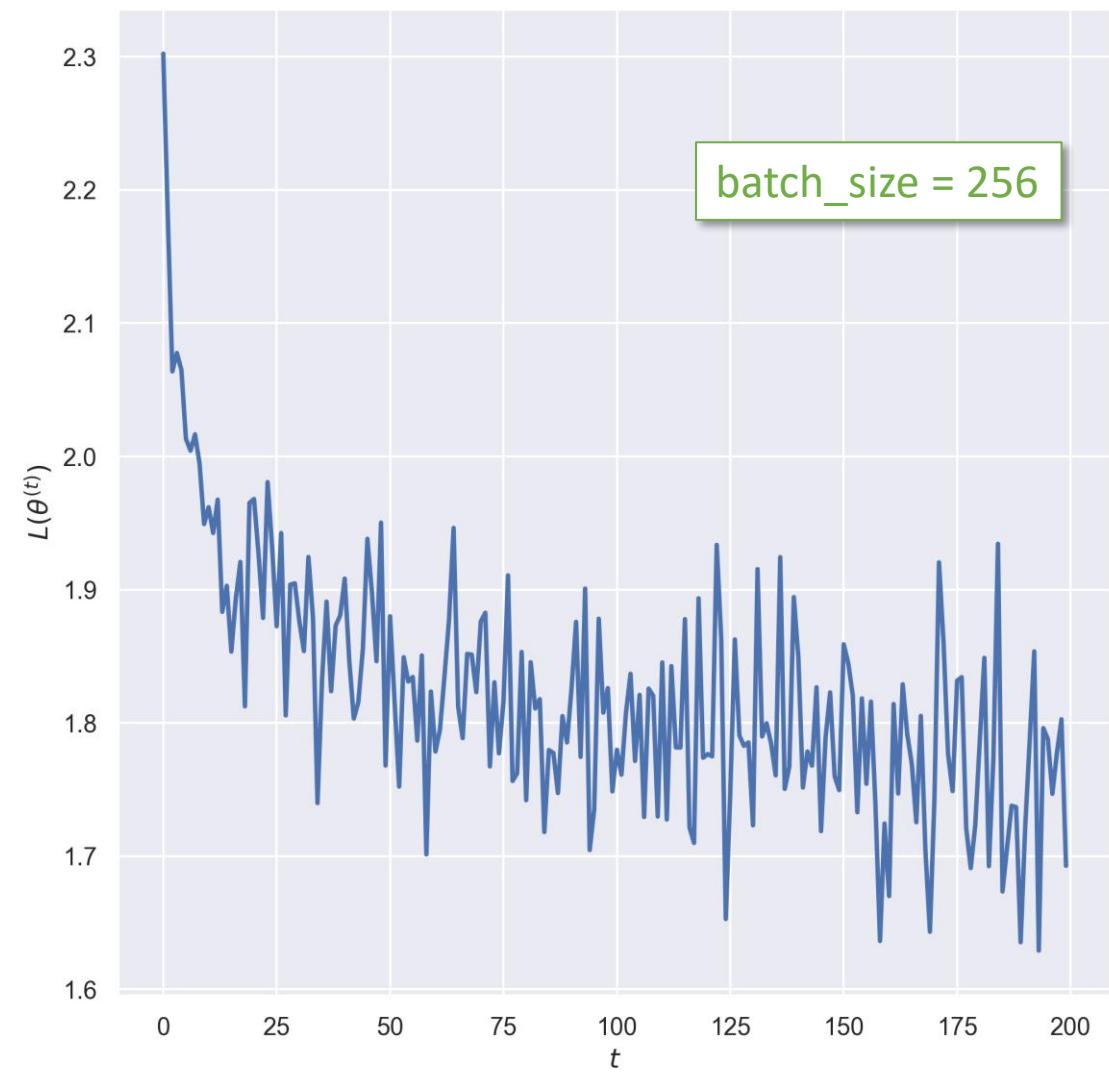
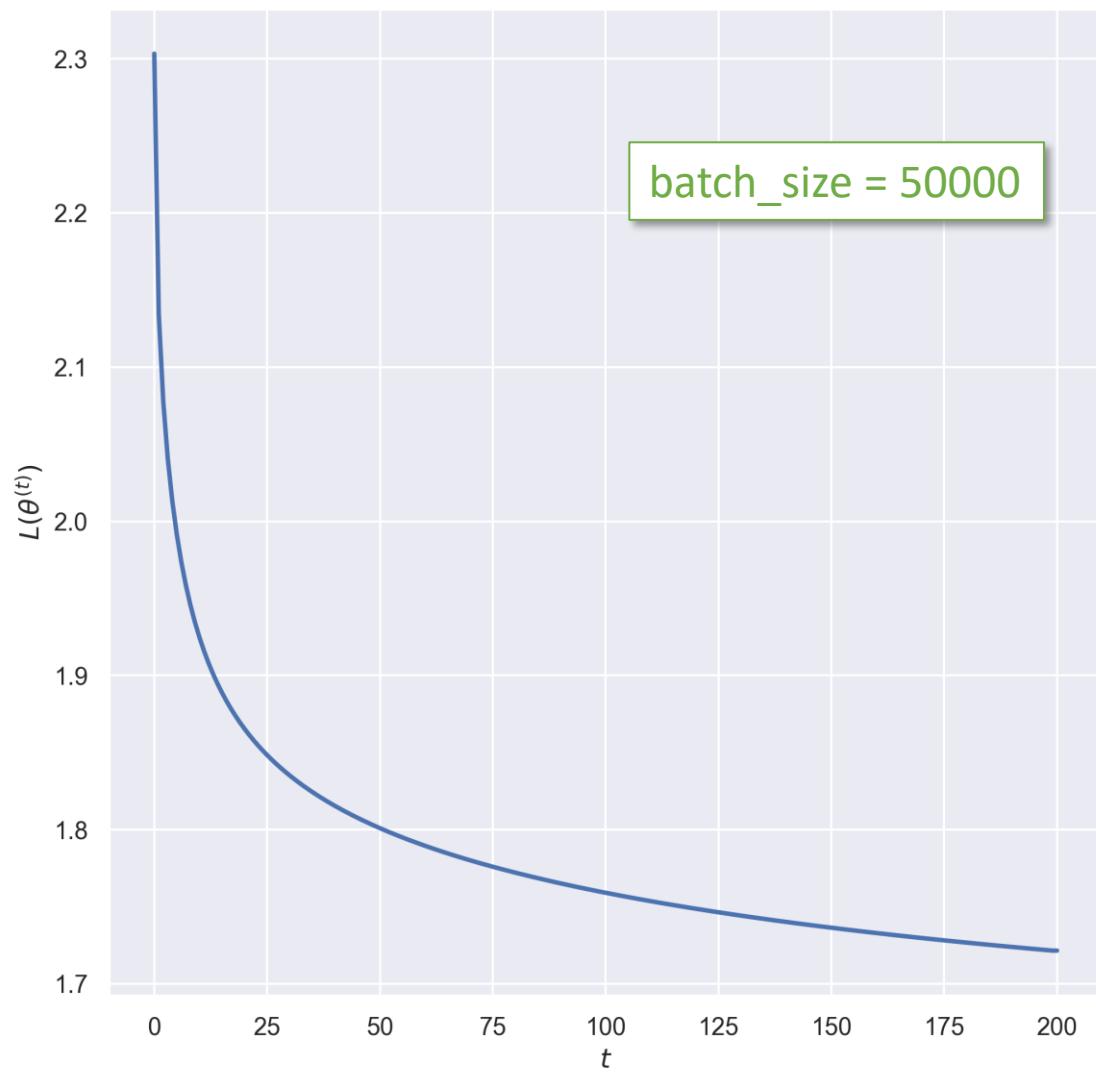
1. Pro každý vzorek x_n v navzorkované dívce x_1, \dots, x_j
 - a. predikujeme pravděpodobnosti \hat{p}_n
 - b. vypočteme dílčí kritérium l_n a akumulujeme k celkovému l
 - c. vypočteme dílčí gradient ∇L_n a akumulujeme k celkovému ∇L
2. updatujeme parametry θ akumulovaným gradientem ∇L s krokem γ

Zastavíme:

- po fixním počtu iterací
- parametry θ se ustálí
- hodnota kritéria $L(\theta)$ již delší dobu neklesá



GD vs SGD



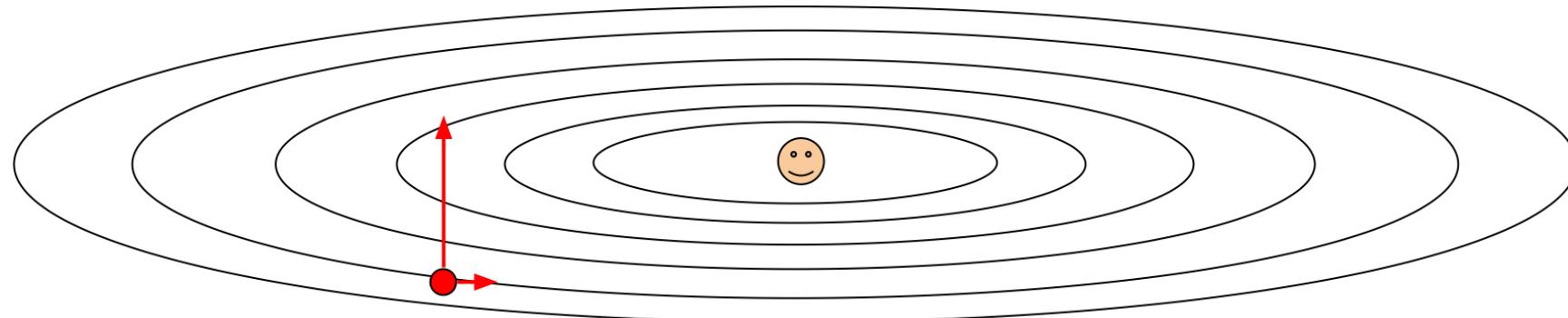
GD vs SGD

Gradient descent (GD)	Stochastic gradient descent (SGD)
skutečný gradient	pouze aproximuje gradient
stabilnější konvergence	rychlejší konvergence
konverguje do minima	osciluje kolem minima
náchylnější k upadnutí do lokálního minima	robustnější vůči lokálním minimům

Především vzhledem k výpočetním nárokům plného GD se pro učení neuronových sítí se prakticky výhradně používá Stochastic/Minibatch GD

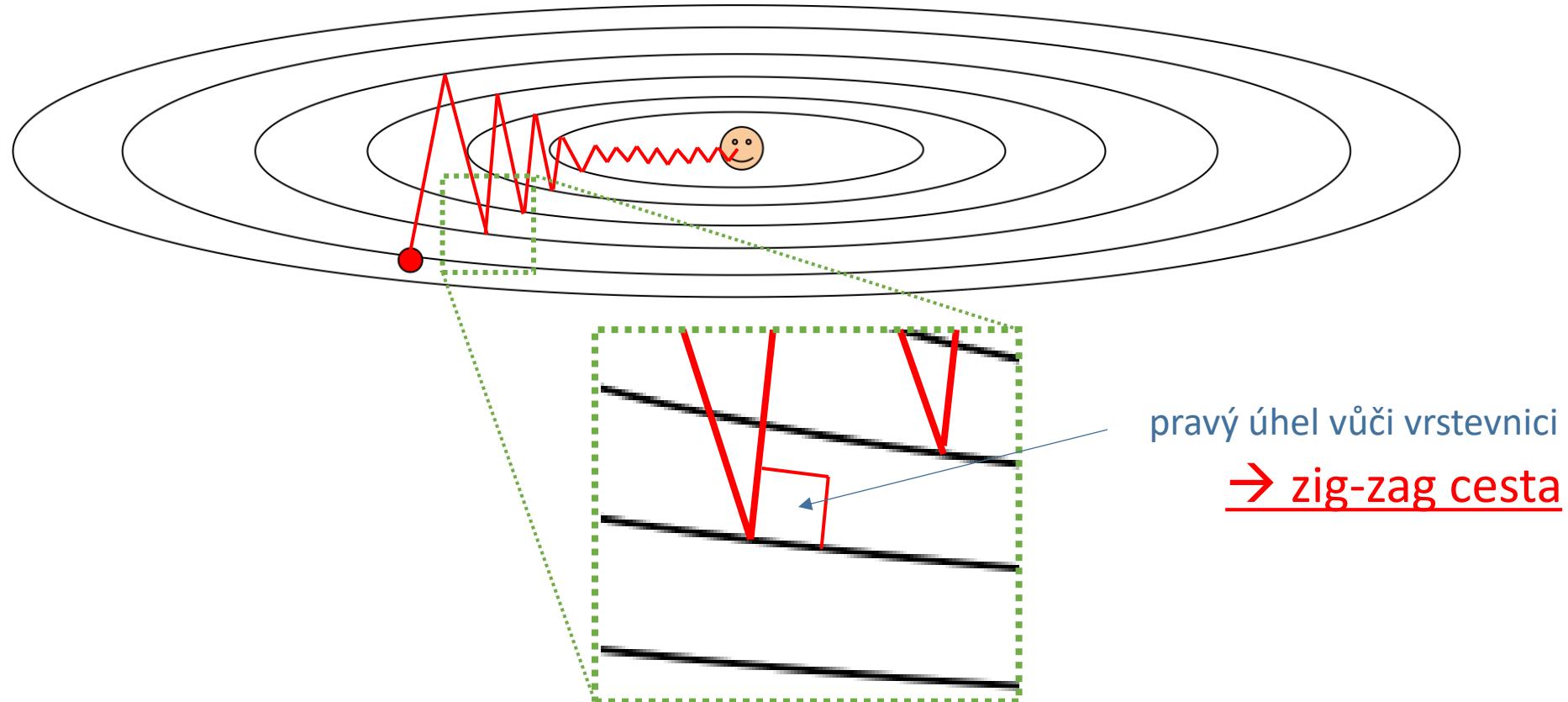
Problémy SGD: vysoká podmíněnost

funkce, která je v jednom směru mnohem citlivější na změnu



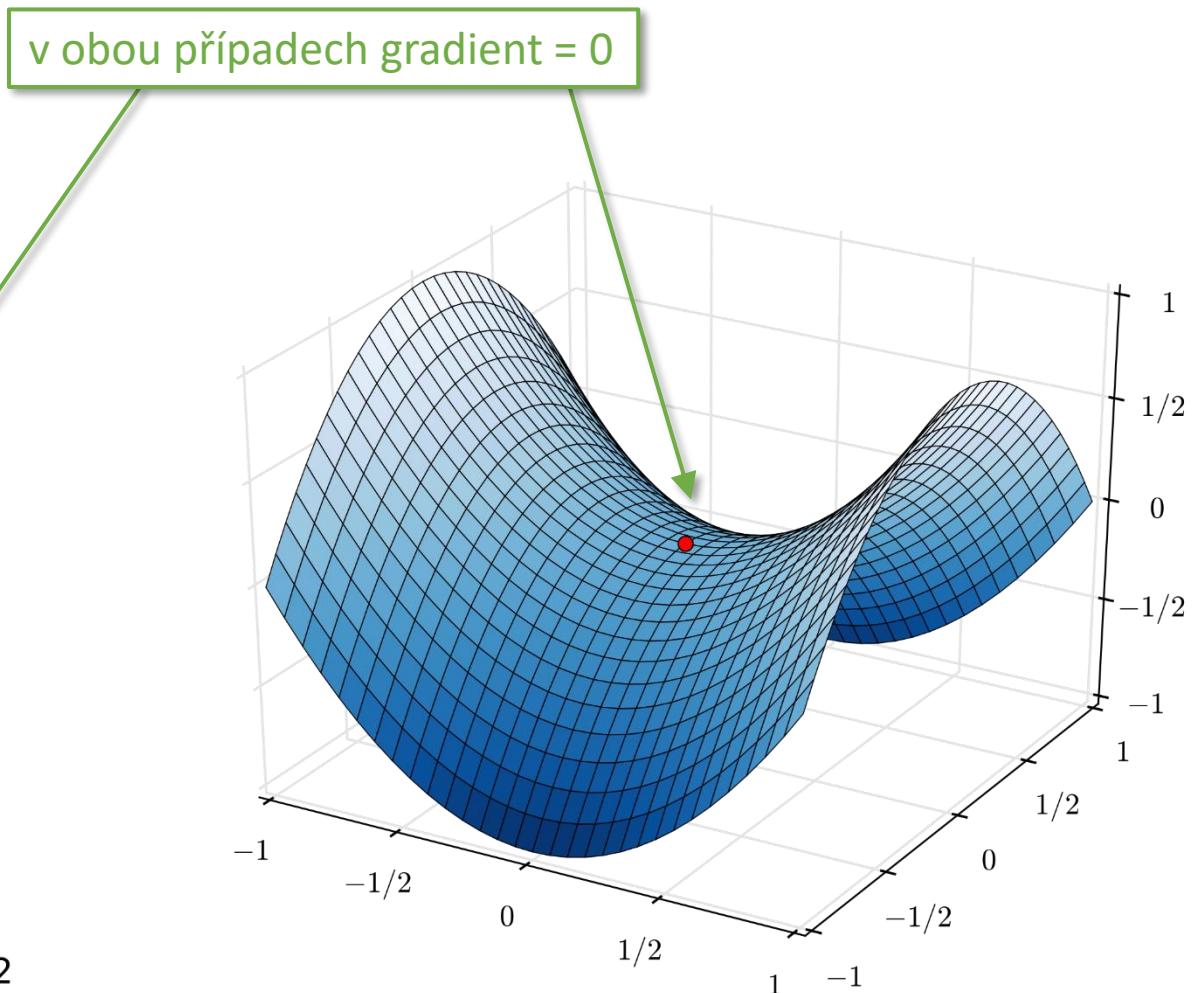
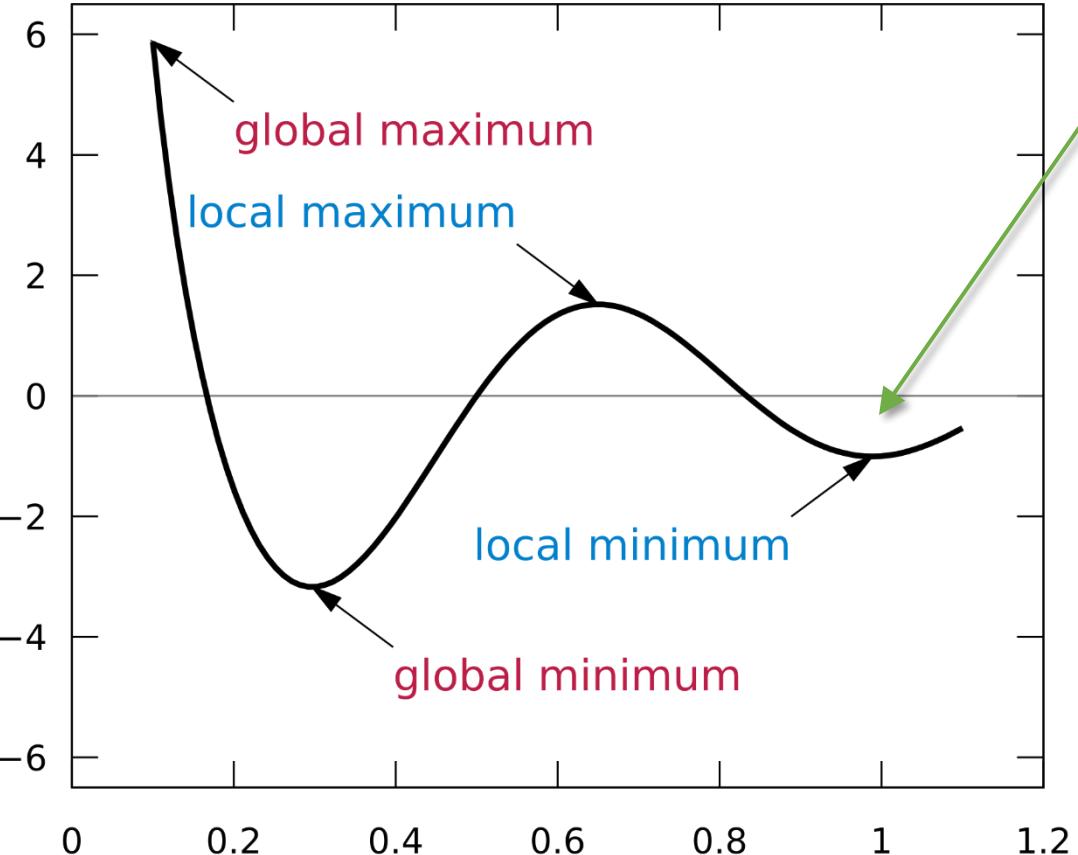
Problémy SGD: vysoká podmíněnost

funkce, která je v jednom směru mnohem citlivější na změnu



obrázek: <http://cs231n.stanford.edu/>

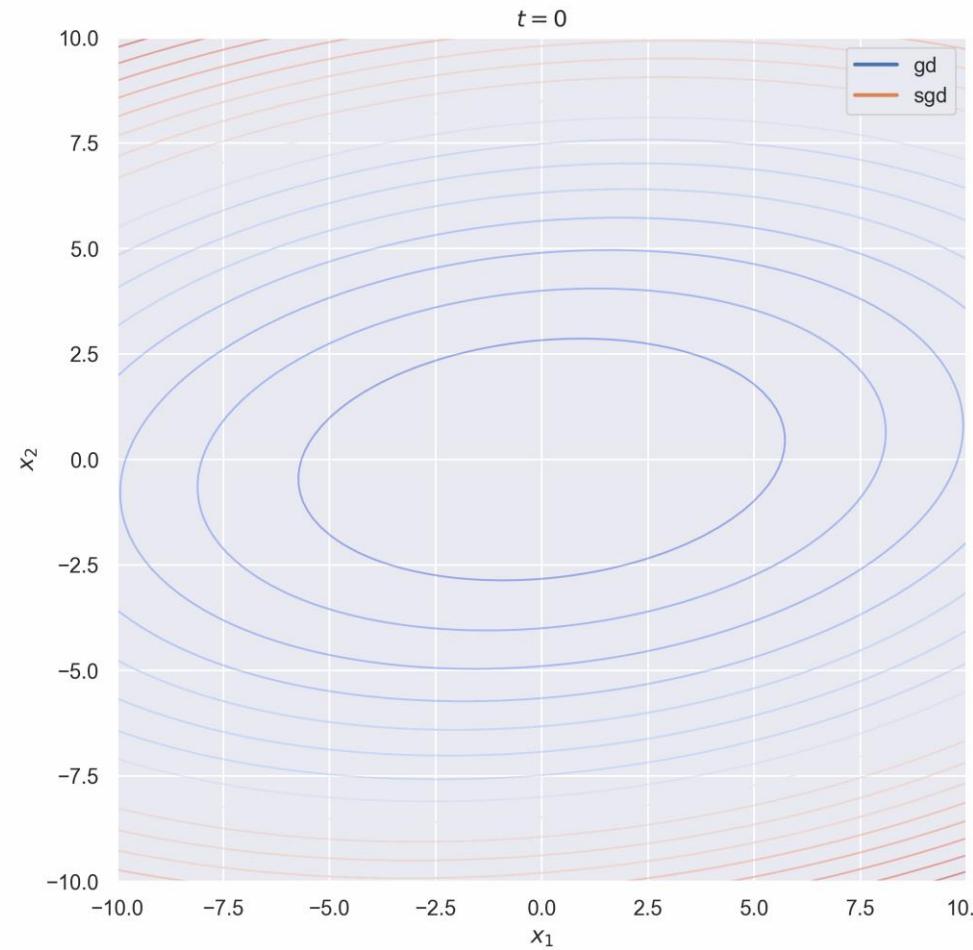
Problémy SGD: lokální minima a sedlové body



obrázek: https://en.wikipedia.org/wiki/Maximum_and_minimum

obrázek: https://en.wikipedia.org/wiki/Saddle_point

Problémy SGD: šum v gradientu způsobený vzorkováním

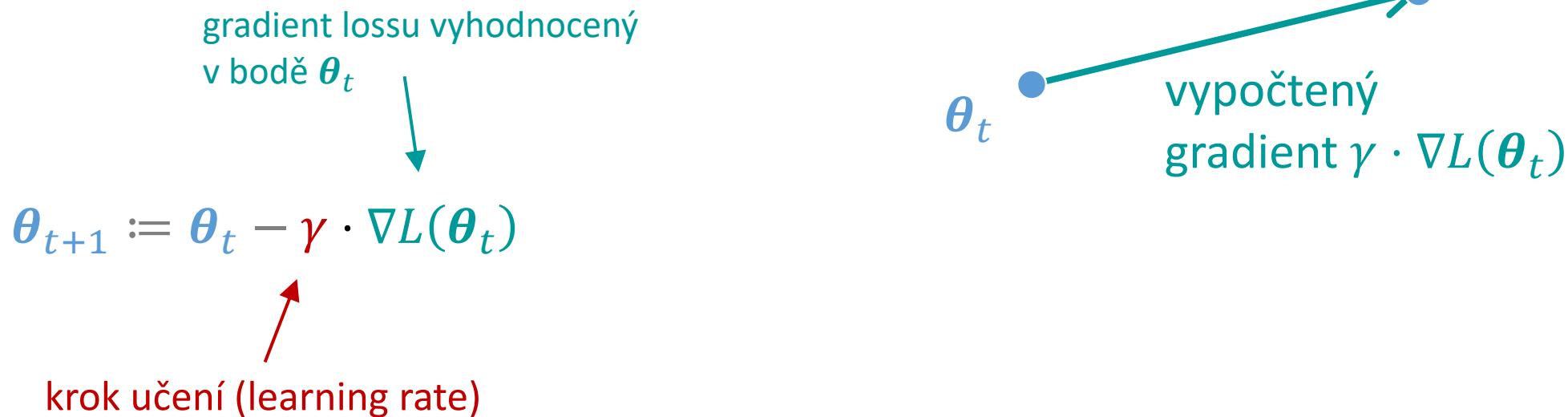


Optimalizační metody založené na momentum

Momentum SGD, Nesterov Accelerated Gradient

Update pravidlo SGD

- Optimalizujeme parametry θ , tj. např. váhovou matici a bias
- Aktuální hodnota je θ_t
- Nový odhad bude θ_{t+1}
- Všechno jsou vektory



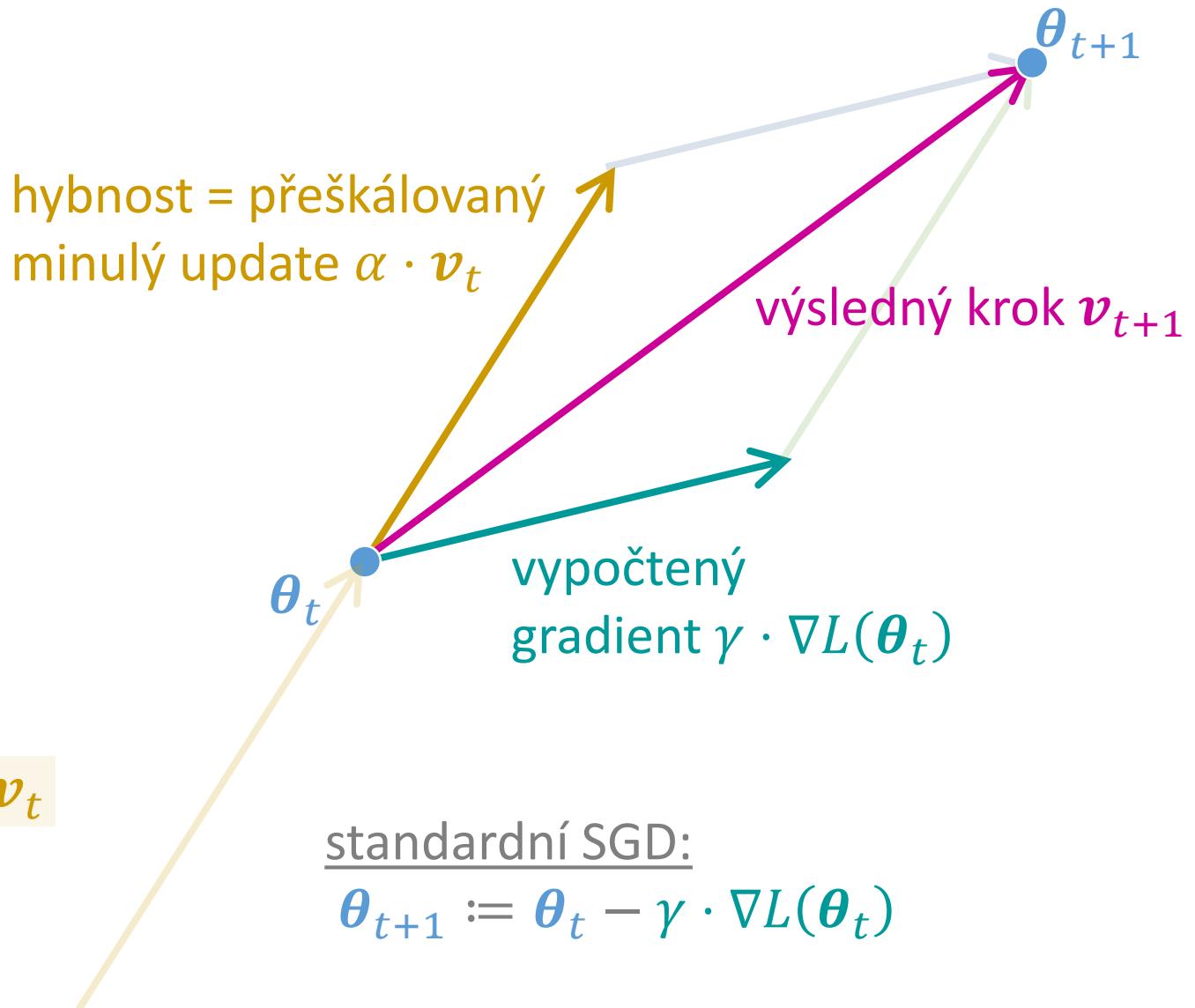
Momentum SGD

- Pamatuje si předchozí update
- Přičítá k novému
- Momentum = hybnost, aneb simuluje “koulení” z kopce
- Obvykle konverguje rychleji

hyperparametr, např. $\alpha = 0.95$

$$(1) \quad \mathbf{v}_{t+1} := \alpha \cdot \mathbf{v}_t - \gamma \cdot \nabla L(\boldsymbol{\theta}_t)$$

$$\begin{aligned} (2) \quad \boldsymbol{\theta}_{t+1} &:= \boldsymbol{\theta}_t + \mathbf{v}_{t+1} \\ &:= \boldsymbol{\theta}_t - \gamma \cdot \nabla L(\boldsymbol{\theta}_t) + \alpha \cdot \mathbf{v}_t \end{aligned}$$

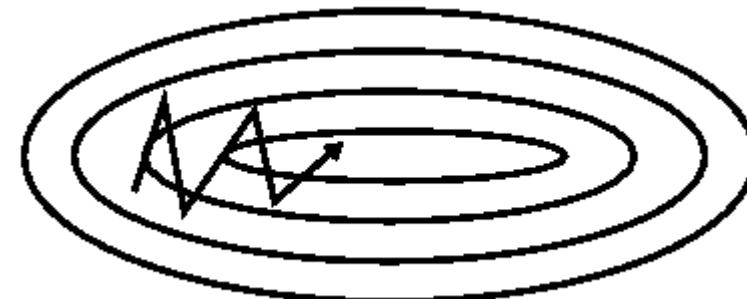


Momentum SGD

Obyčejné SGD

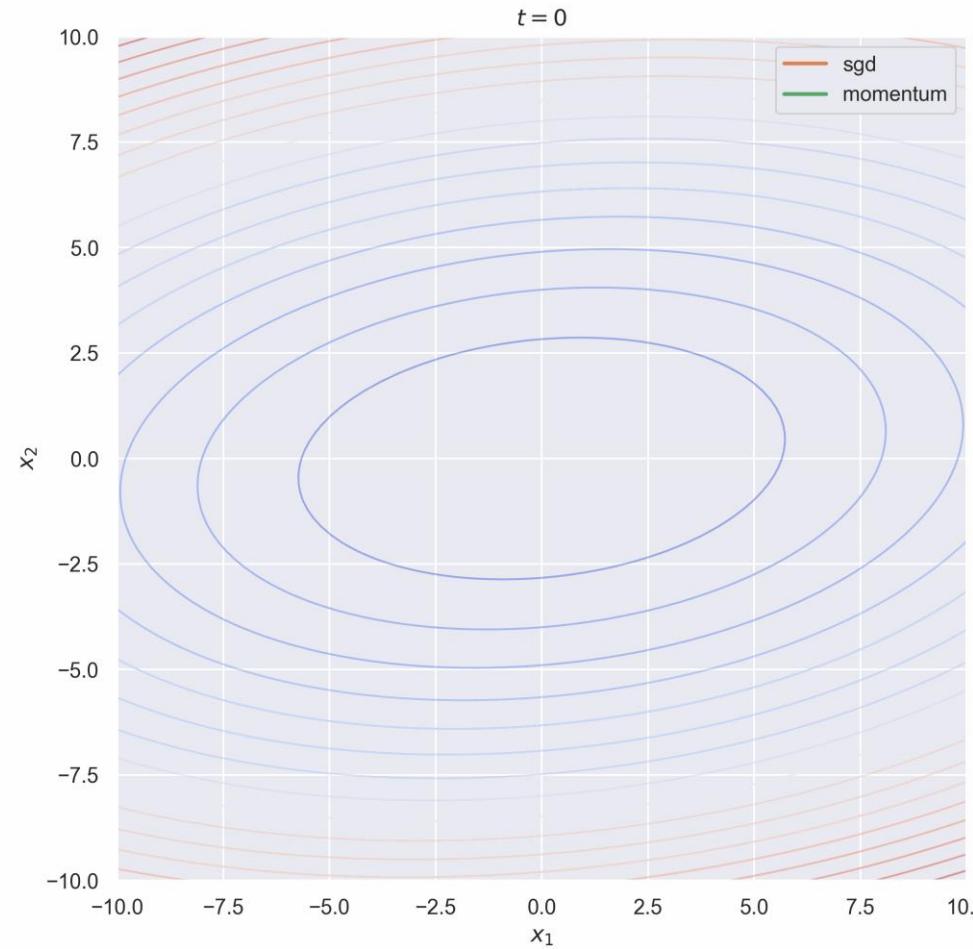


SGD + momentum



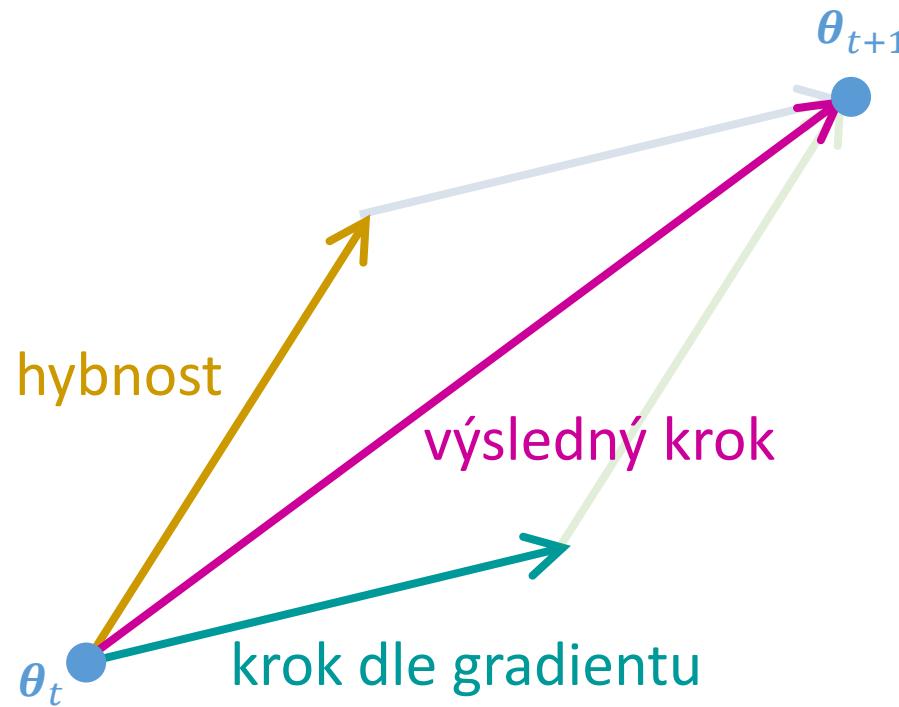
v horizontálním směru postupně nabírá rychlosť

Momentum SGD



Nesterov Accelerated Gradient (NAG)

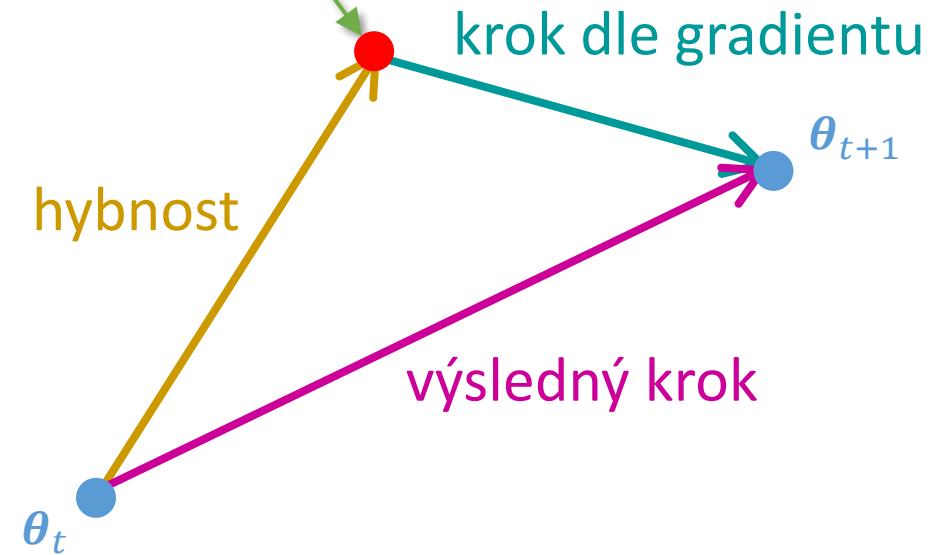
SGD + momentum



$$\boldsymbol{v}_{t+1} := \alpha \cdot \boldsymbol{v}_t - \gamma \cdot \nabla L(\boldsymbol{\theta}_t)$$

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}$$

gradient se vychodnocuje
v bodě $\boldsymbol{\theta}_t + \alpha \cdot \boldsymbol{v}_t$



$$\boldsymbol{v}_{t+1} := \alpha \cdot \boldsymbol{v}_t - \gamma \cdot \nabla L(\boldsymbol{\theta}_t + \alpha \cdot \boldsymbol{v}_t)$$

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}$$

Nesterov

nejprve update hybností,
poté teprve výpočet gradientu

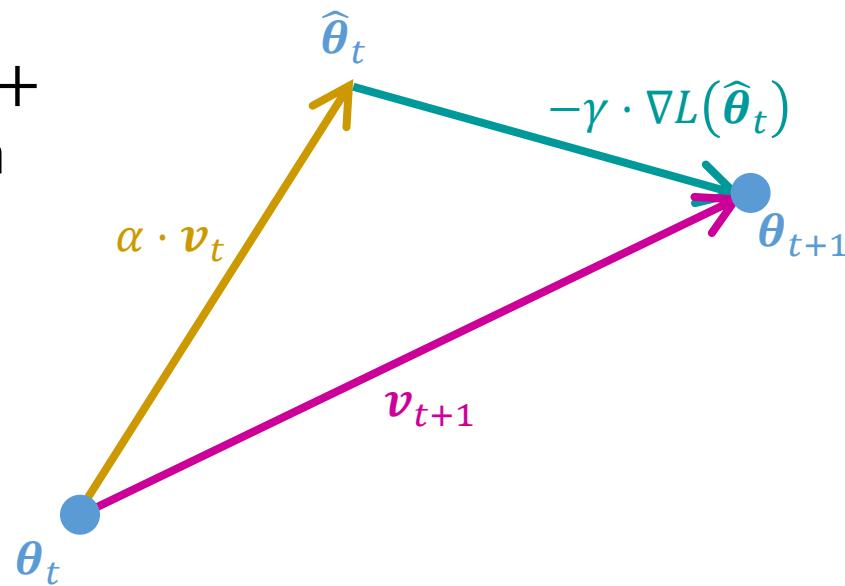
Nesterov Accelerated Gradient (NAG)

- Nesterov mechanismus:

$$\boldsymbol{v}_{t+1} := \alpha \cdot \boldsymbol{v}_t - \gamma \cdot \nabla L(\boldsymbol{\theta}_t + \alpha \cdot \boldsymbol{v}_t)$$

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}$$

- Nutnost vyhodnotit gradient v bodě $\boldsymbol{\theta}_t + \alpha \boldsymbol{v}_t$ je nepříjemná vlastnost a nezapadá do optimalizačního “API” (loss je vyhodnocen s jinými parametry, než ze kterých je počítán gradient)
- V naivní implementaci bychom navíc museli počítat dopředný průchod (vyhodnotit loss) dvakrát



Nesterov Accelerated Gradient (NAG)

- Nesterov mechanismus:

$$\boldsymbol{v}_{t+1} := \alpha \cdot \boldsymbol{v}_t - \gamma \cdot \nabla L(\boldsymbol{\theta}_t + \alpha \cdot \boldsymbol{v}_t)$$

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}$$

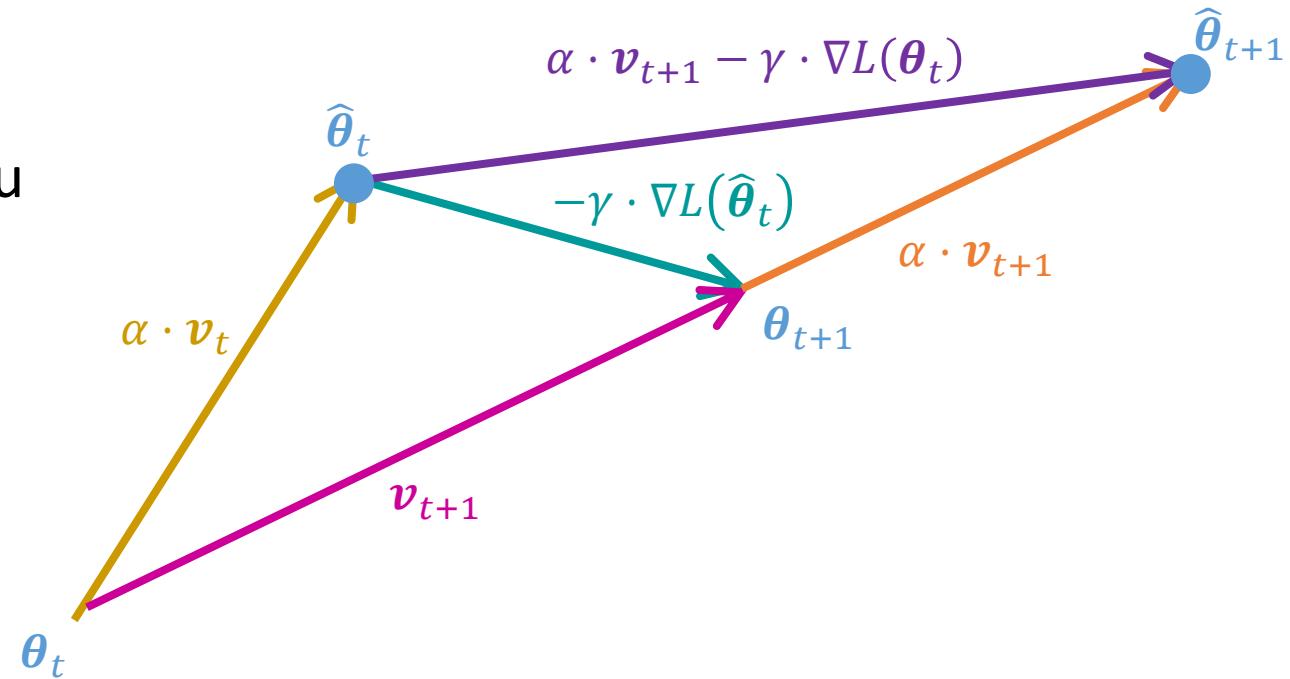
- Pravidlo ale lze přeformulovat z pohledu "posunutých" parametrů $\widehat{\boldsymbol{\theta}}_t$:

$$\boldsymbol{v}_{t+1} := \alpha \cdot \boldsymbol{v}_t - \gamma \cdot \nabla L(\widehat{\boldsymbol{\theta}}_t)$$

$$\widehat{\boldsymbol{\theta}}_{t+1} := \widehat{\boldsymbol{\theta}}_t - \gamma \cdot \nabla L(\widehat{\boldsymbol{\theta}}_t) + \alpha \cdot \boldsymbol{v}_{t+1}$$

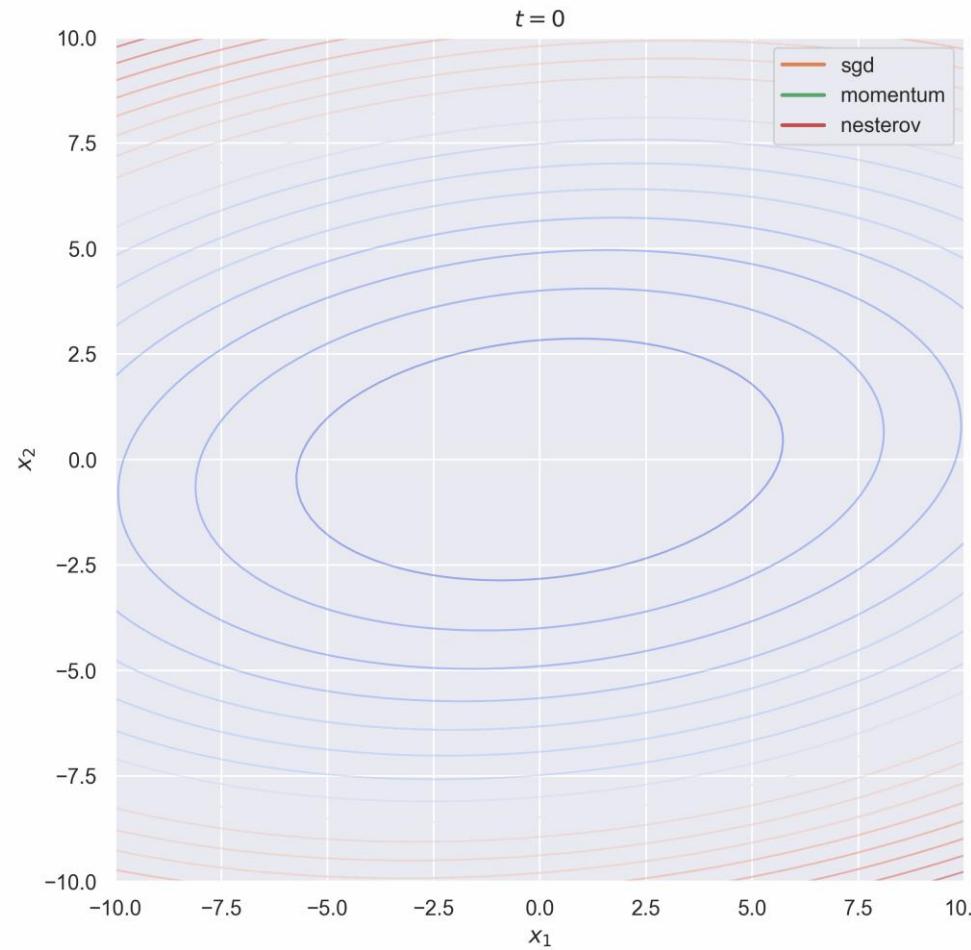
$$:= \widehat{\boldsymbol{\theta}}_t + \boldsymbol{v}_{t+1} - \alpha \cdot \boldsymbol{v}_t + \alpha \cdot \boldsymbol{v}_{t+1}$$

$$:= \widehat{\boldsymbol{\theta}}_t + \boldsymbol{v}_{t+1} + \alpha \cdot (\boldsymbol{v}_{t+1} - \boldsymbol{v}_t)$$



- Tj. výpočet hybnosti zůstává stejný, ale update parametrů má **extra korekční člen navíc**

Nesterov Accelerated Gradient (NAG)



Adaptivní metody

AdaGrad, RMSprop, Adam

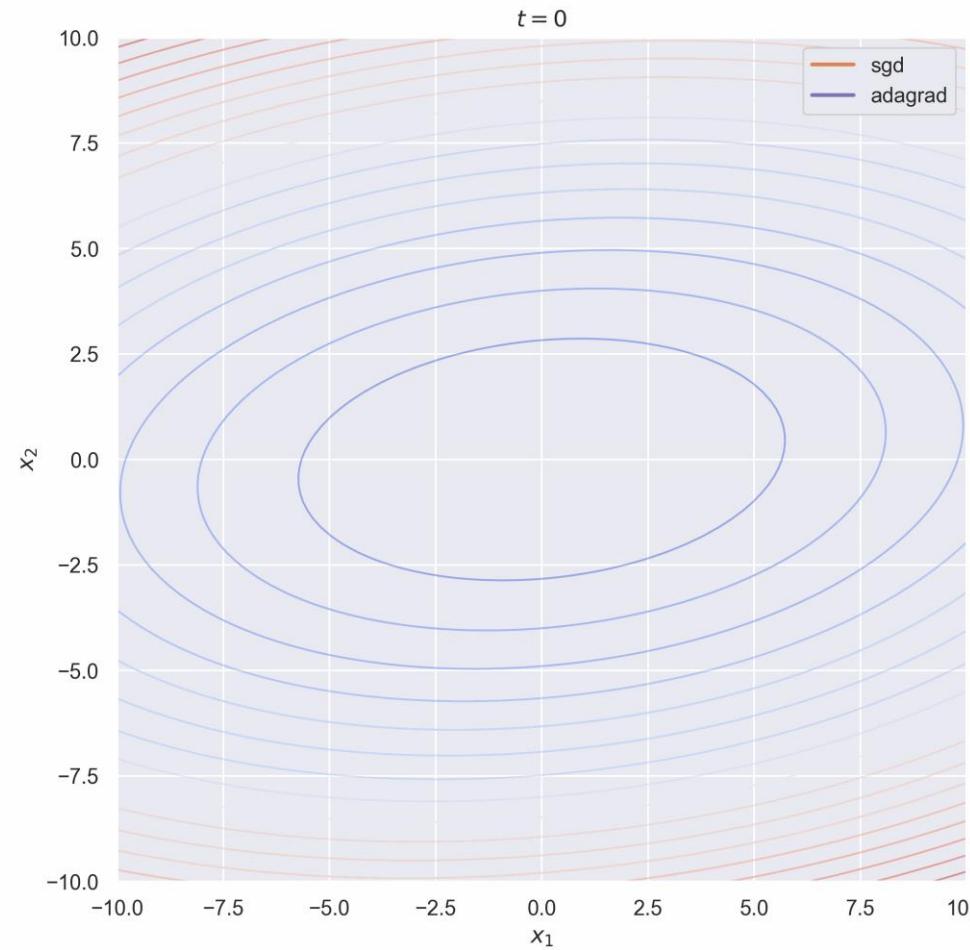
Adaptive Gradient (AdaGrad)

- Upravuje krok pro jednotlivé parametry, postupně sčítá jejich kvadráty (“energii”)
- Pokud je gradient v některých směrech neustále vyšší než jiné → normalizace
- Tzn. zmenšuje “protáhlé” dimenze = zvětšuje “splácnuté” → narovnává
- Pro každý prvek gradientu samostatně → každý parametr vlastní learning rate
- Pravidlo updatu je

$$\begin{aligned} \mathbf{g}_{t+1} &:= \mathbf{g}_t + \nabla L(\boldsymbol{\theta}_t)^2 && \xleftarrow{\text{prvkově na druhou}} \\ \boldsymbol{\theta}_{t+1} &:= \boldsymbol{\theta}_t - \gamma \cdot \frac{\nabla L(\boldsymbol{\theta}_t)}{\sqrt{\mathbf{g}_{t+1}} + \epsilon} && \xleftarrow{\text{dělení po prvcích}} \\ && \epsilon \approx 10^{-7} \text{ pouze zabraňuje dělení nulou} & \end{aligned}$$

\mathbf{g}_t postupně akumuluje kvadrát gradientu

Adaptive Gradient (AdaGrad)



RMSprop (a AdaDelta)

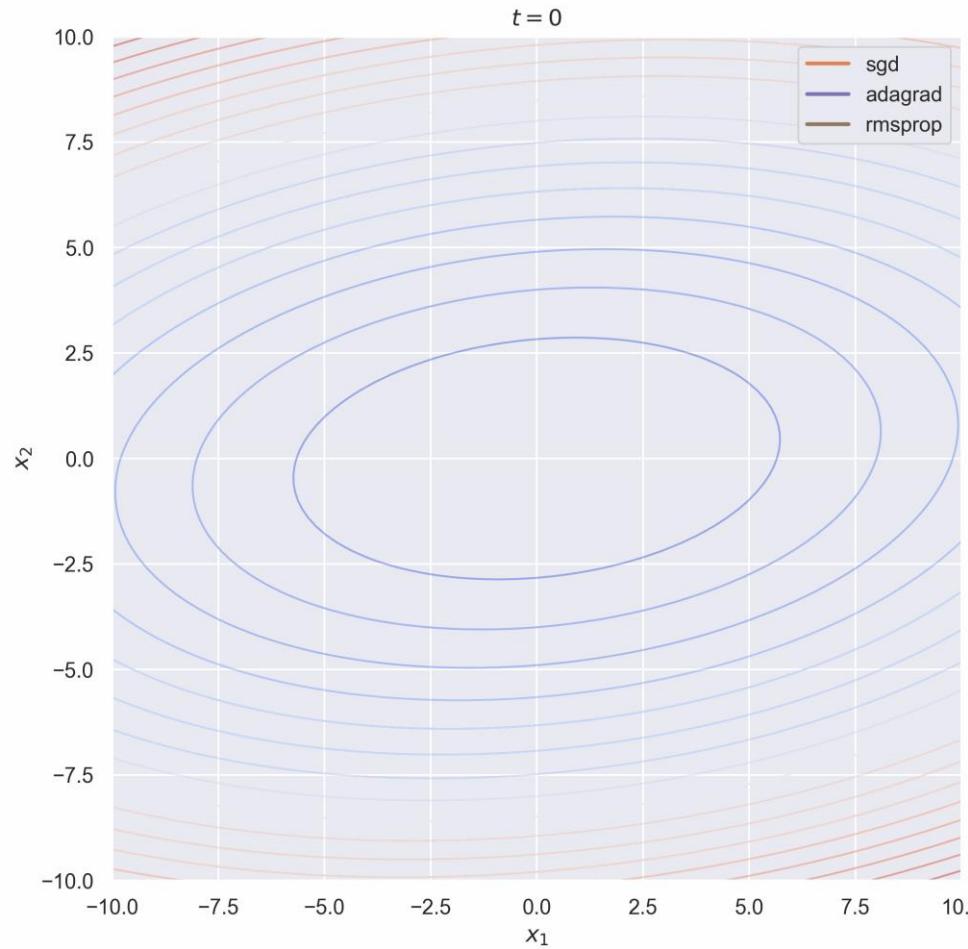
- Root mean square = průběžně upravuje odhad kvadrátu jako klouzavý průměr
- Řeší problém s postupným nárůstem jmenovatele a zmenšování updatů

decay rate ... hyperparametr, typicky $\beta = 0.99$

průměrná norma gradientů →
(prvkově pro každý parameter
→ stejný rozměr jako gradient)

$$\begin{aligned} \mathbf{u}_{t+1} &:= \beta \cdot \mathbf{u}_t + (1 - \beta) \cdot \nabla L(\boldsymbol{\theta}_t)^2 \\ \boldsymbol{\theta}_{t+1} &:= \boldsymbol{\theta}_t - \gamma \cdot \frac{\nabla L(\boldsymbol{\theta}_t)}{\sqrt{\mathbf{u}_{t+1}} + \epsilon} \end{aligned}$$

RMSprop



Adaptive Momentum (Adam)

- Kombinace Momentum* SGD + RMSprop*

momentum*: $\boldsymbol{v}_{t+1} := \alpha \cdot \boldsymbol{v}_t + (1 - \alpha) \cdot \nabla L(\boldsymbol{\theta}_t)$

rmsprop: $\boldsymbol{u}_{t+1} := \beta \cdot \boldsymbol{u}_t + (1 - \beta) \cdot \nabla L(\boldsymbol{\theta}_t)^2$

Adam update: $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \gamma \cdot \frac{\boldsymbol{v}_{t+1}}{\sqrt{\boldsymbol{u}_{t+1}} + \epsilon}$

- Obvykle funguje dobře i s výchozím nastavením hyperparametrů, $\alpha = 0.9$, $\beta = 0.999$
- Dobrá výchozí volba
- Co ale počáteční podmínky, kdy $\boldsymbol{v}_t = \boldsymbol{u}_t = \mathbf{0}$?

Adaptive Momentum (Adam)

- Kombinace Momentum* SGD + RMSprop*

momentum*: $v_{t+1} := \alpha \cdot v_t + (1 - \alpha) \cdot \nabla L(\theta_t)$

rmsprop: $u_{t+1} := \beta \cdot u_t + (1 - \beta) \cdot \nabla L(\theta_t)^2$

korekce:

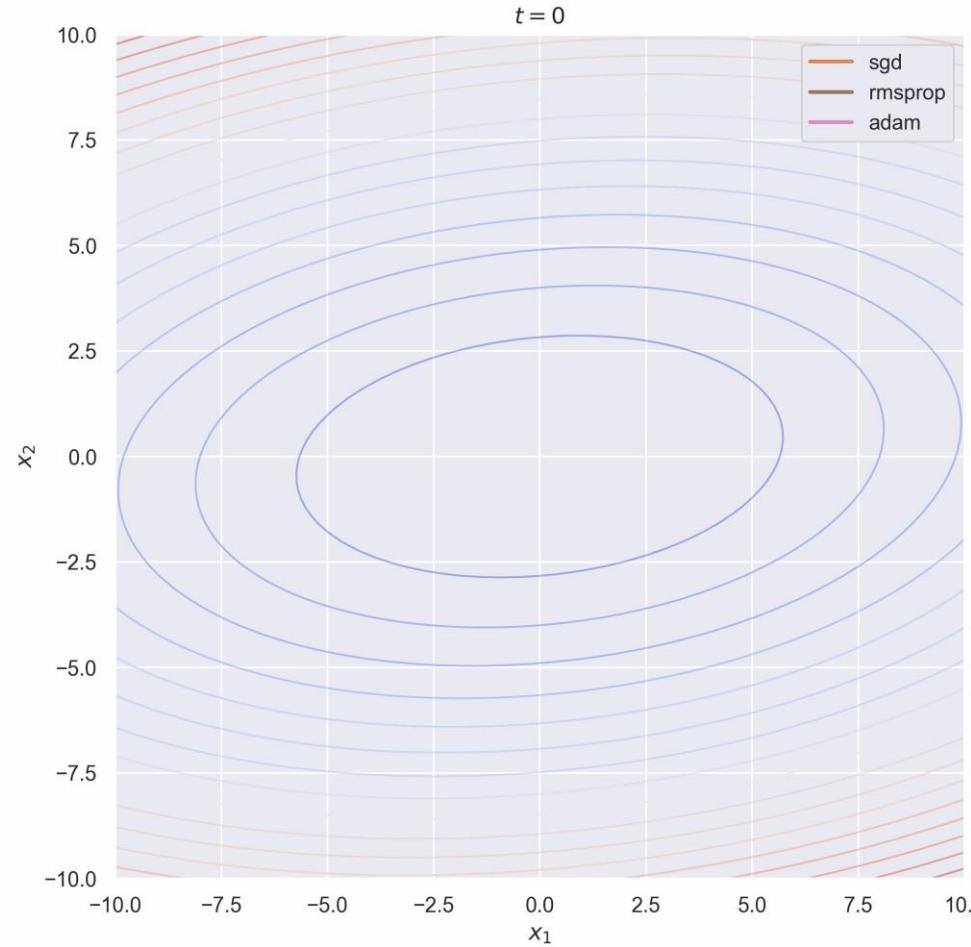
$$\hat{v}_{t+1} := \frac{v_{t+1}}{1 - \alpha^t}$$

$$\hat{u}_{t+1} := \frac{u_{t+1}}{1 - \beta^t}$$

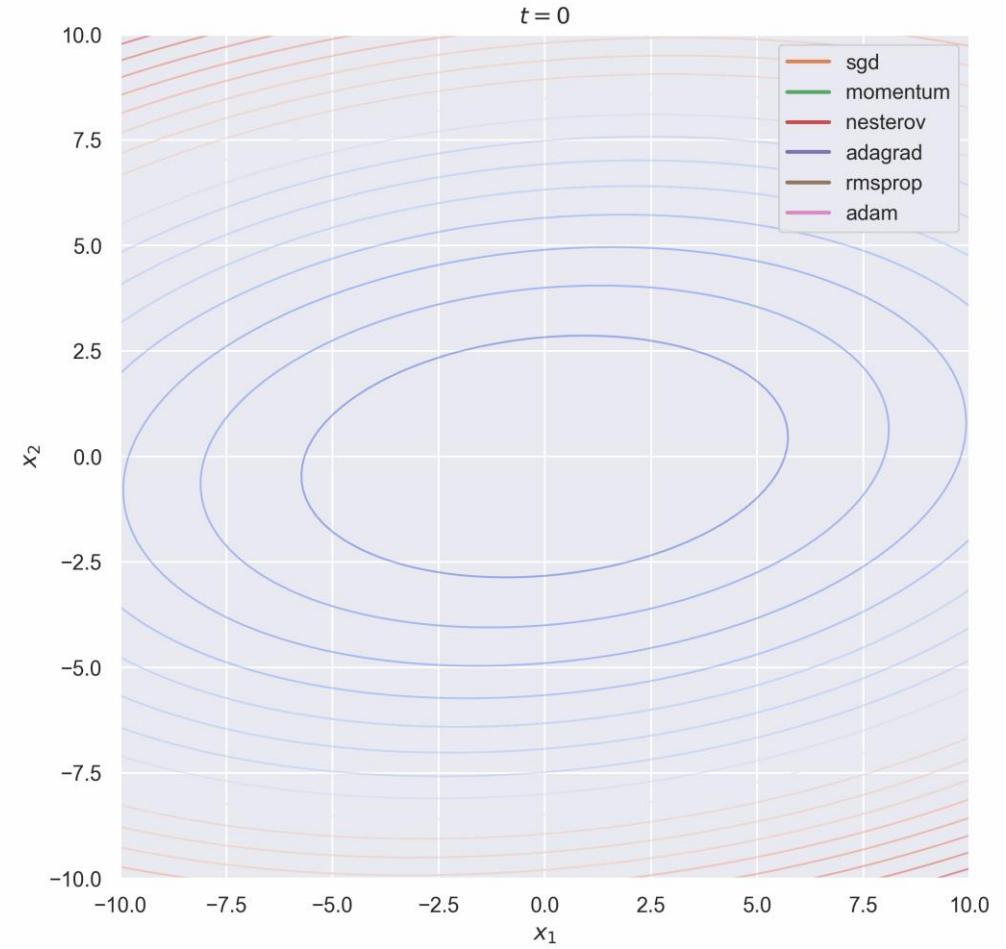
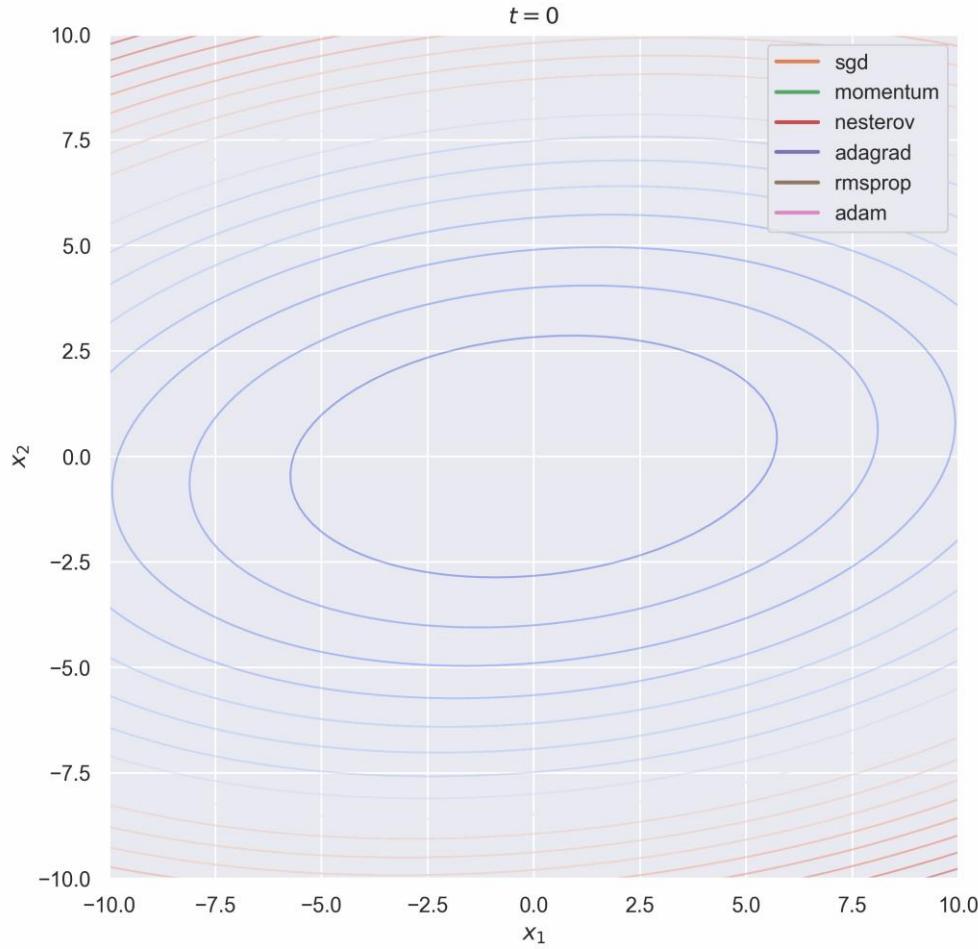
$1 - 0.9^1 = 0.1$
 $1 - 0.9^2 = 0.19$
 \dots
 $1 - 0.9^{100} = 0.99997$

Adam update: $\theta_{t+1} := \theta_t - \gamma \cdot \frac{\hat{v}_{t+1}}{\sqrt{\hat{u}_{t+1}} + \epsilon}$

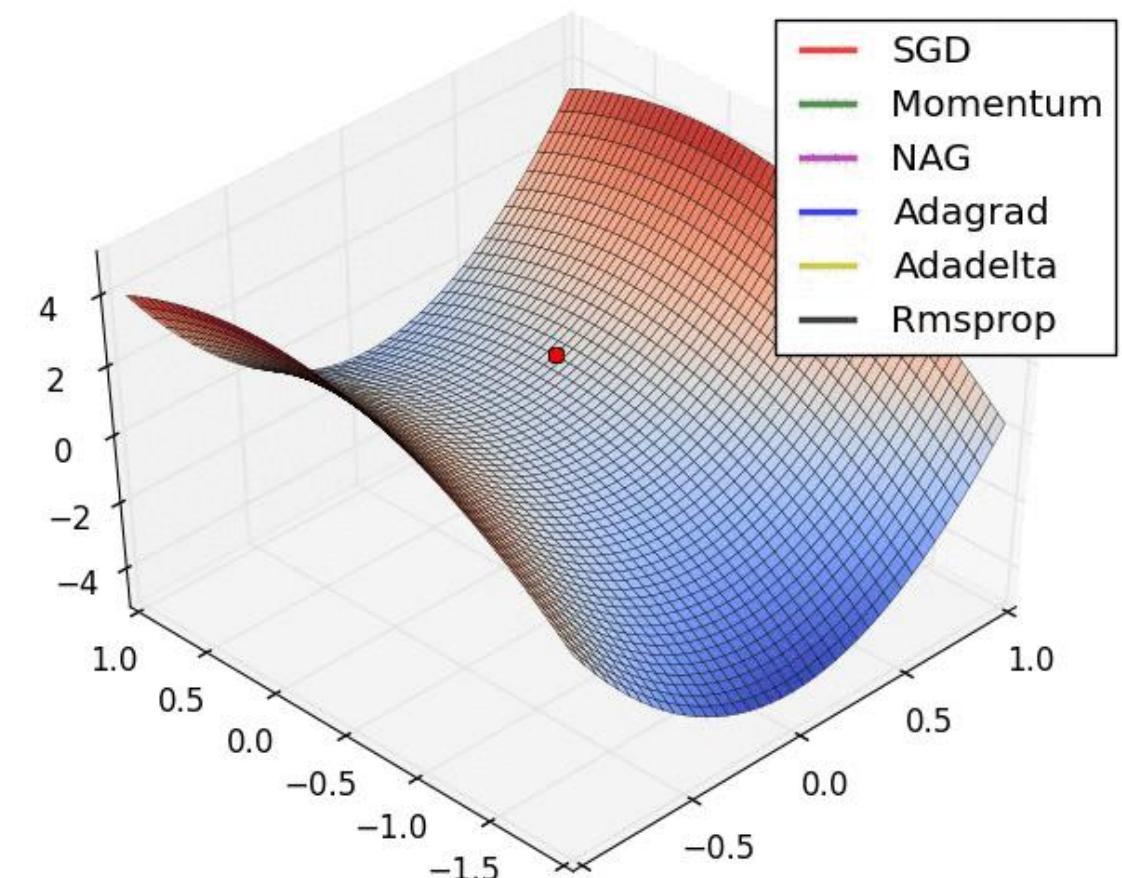
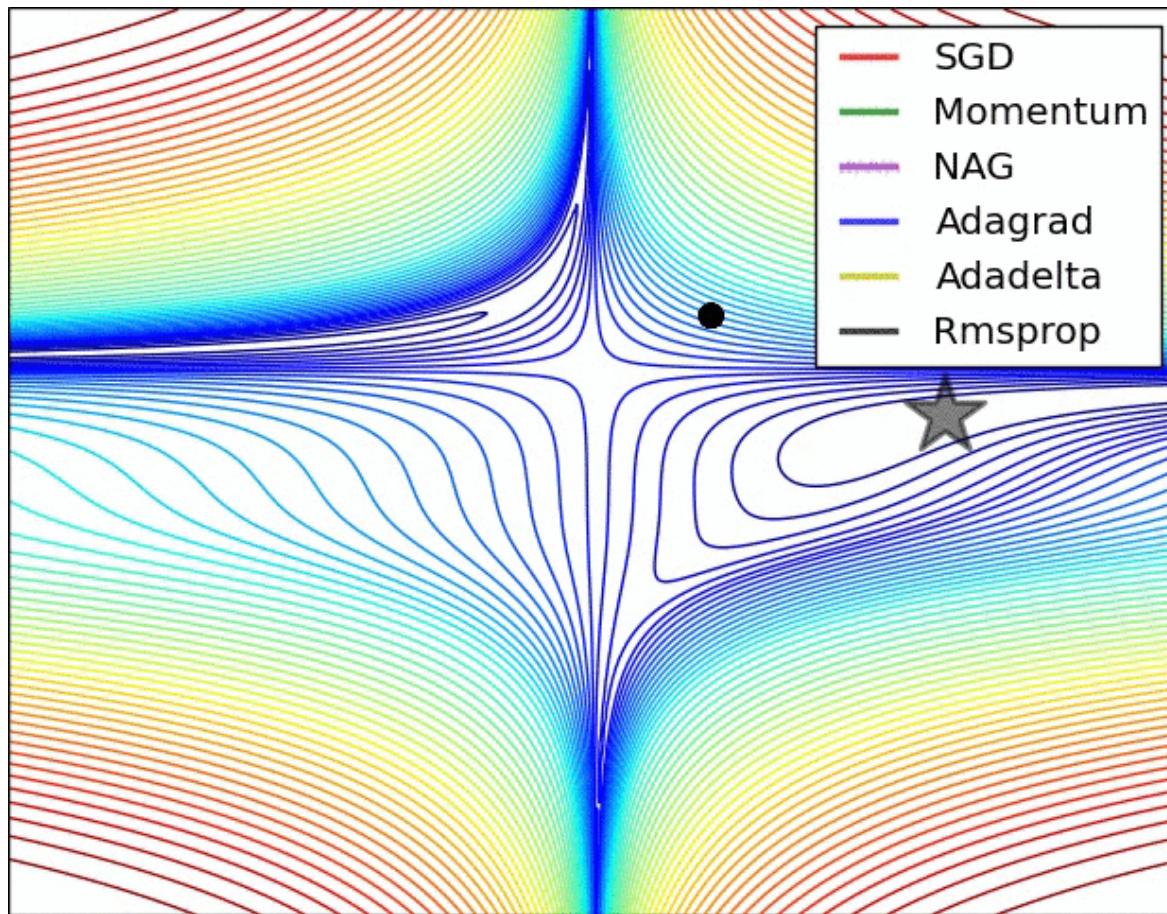
Adaptive Momentum (Adam)



Šum v gradientu (velikost dávky)



Další vizualizace



animace + výborný přehled včetně dalších metod : <https://ruder.io/optimizing-gradient-descent/>

Který optimizér zvolit?

- Adam(W)!
- Obvykle funguje dobře i s výchozím nastavením hyperparametrů
- Robustní vůči šumu v gradientu (velikosti dávky)
- Díky tomu je Adam je vhodná volba jako výchozí optimizér
- Pokud vše funguje s Adamem, bývá možné dosáhnout o něco lepších výsledků především na neviděných datech pomocí SGD+momentum

L2 regularizace vs weight decay

L2 regularizace vs weight decay

- Loss s L2 regularizací má obecně formu

$$L(\mathbf{X}, \mathbf{w}, \mathbf{b}) = L_{\text{data}}(\mathbf{X}, \mathbf{w}, \mathbf{b}) +$$

L2 regularizace

$$\lambda \cdot \sum_{k,d} w_{k,d}^2$$

- Gradient pak je

$$\frac{\partial L(\mathbf{X}, \mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \frac{\partial L_{\text{data}}(\mathbf{X}, \mathbf{w}, \mathbf{b})}{\partial \mathbf{w}}$$

příspěvek do gradientu
díky datům

$$+ \lambda \cdot 2 \cdot \mathbf{w}$$

příspěvek do gradientu
díky L2 regularizaci

L2 regularizace vs weight decay

- Gradient s regularizací

$$\frac{\partial L(\mathbf{X}, \mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \mathbf{d}\mathbf{w} + \lambda \cdot 2 \cdot \mathbf{w}$$

```
CLASS torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0,  
nesterov=False, *, maximize=False, foreach=None, differentiable=False) [SOURCE]
```

Implements stochastic gradient descent (optionally with momentum).

input : γ (lr), θ_0 (params), $f(\theta)$ (objective), λ (weight decay),
 μ (momentum), τ (dampening), *nesterov*, *maximize*

for $t = 1$ **to** ... **do**

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$

if $\lambda \neq 0$

$g_t \leftarrow g_t + \lambda \theta_{t-1}$

<https://github.com/pytorch/pytorch/blob/main/torch/optim/sgd.py#L244>

```
244. for i, param in enumerate(params):  
245.     # ...  
246.     if weight_decay != 0:  
247.         d_p = d_p.add(param, alpha=weight_decay)
```

<https://pytorch.org/docs/stable/generated/torch.optim.SGD.html>

L2 regularizace vs weight decay

- Gradient s regularizací

$$\frac{\partial L(\mathbf{X}, \mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \mathbf{dw} + \lambda \cdot 2 \cdot \mathbf{w}$$

- Update vah v SGD je

$$\begin{aligned}\mathbf{w} &:= \mathbf{w} - \gamma \cdot \frac{\partial L(\mathbf{X}, \mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} \\ &:= \mathbf{w} - \gamma \cdot \mathbf{dw} - \gamma \cdot \lambda \cdot 2 \cdot \mathbf{w} \\ &:= (1 - 2 \cdot \gamma \cdot \lambda) \cdot \mathbf{w} - \gamma \cdot \mathbf{dw}\end{aligned}$$

L2 regularizace se často označuje jako weight decay, protože neustálým násobením číslem menším než 1 postupně ubývají váhy na velikostí

L2 regularizace vs weight decay

- Gradient s regularizací

$$\frac{\partial L(\mathbf{X}, \mathbf{w}, \mathbf{b})}{\partial \mathbf{w}} = \mathbf{d}\mathbf{w} + \lambda \cdot 2 \cdot \mathbf{w}$$

- U metod prvního řádu jako SGD a momentum jsou L2 regularizace a weight decay ekvivalentní
- Ale co adaptivní metody jako AdaGrad, RMSprop nebo Adam, které při výpočtu updatu využívají druhou mocninu gradientu?
- **Obecně proto L2 regularizace není to samé jako weight decay**

L2 regularizace vs weight decay

- L2 regularizace:

$$L := L_{\text{data}} + \lambda \cdot \sum_p \theta_p^2$$

$$\nabla L := \nabla L_{\text{data}} + \lambda \cdot 2 \cdot \theta$$

$$\Delta\theta := \text{optimizer}(\nabla L)$$

$$\theta := \theta - \gamma \cdot \Delta\theta$$

- Weight decay:

$$L := L_{\text{data}}$$

$$\nabla L := \nabla L_{\text{data}}$$

$$\Delta\theta := \text{optimizer}(\nabla L) + \lambda \cdot 2 \cdot \theta$$

$$\theta := \theta - \gamma \cdot \Delta\theta$$

AdamW: Adam + weight decay

Algorithm 2 Adam with L₂ regularization and Adam with decoupled weight decay (AdamW)

- 1: **given** $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\lambda \in \mathbb{R}$
 - 2: **initialize** time step $t \leftarrow 0$, parameter vector $\theta_{t=0} \in \mathbb{R}^n$, first moment vector $m_{t=0} \leftarrow \theta$, second moment vector $v_{t=0} \leftarrow \theta$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
 - 3: **repeat**
 - 4: $t \leftarrow t + 1$
 - 5: $\nabla f_t(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$ \triangleright select batch and return the corresponding gradient
 - 6: $g_t \leftarrow \nabla f_t(\theta_{t-1}) + \lambda \theta_{t-1}$
 - 7: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ \triangleright here and below all operations are element-wise
 - 8: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
 - 9: $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ $\triangleright \beta_1$ is taken to the power of t
 - 10: $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ $\triangleright \beta_2$ is taken to the power of t
 - 11: $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ \triangleright can be fixed, decay, or also be used for warm restarts
 - 12: $\theta_t \leftarrow \theta_{t-1} - \eta_t \left(\alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda \theta_{t-1} \right)$
 - 13: **until** stopping criterion is met
 - 14: **return** optimized parameters θ_t
-

L2 regularizace vs weight decay v PyTorch

ADAM ⚡

```
CLASS torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0,
    amsgrad=False, *, foreach=None, maximize=False, capturable=False, differentiable=False,
    fused=None) [SOURCE]
```

Implements Adam algorithm.

```
input :  $\gamma$  (lr),  $\beta_1, \beta_2$  (betas),  $\theta_0$  (params),  $f(\theta)$  (objective)
        $\lambda$  (weight decay), amsgrad, maximize
initialize :  $m_0 \leftarrow 0$  ( first moment),  $v_0 \leftarrow 0$  (second moment),  $\widehat{v}_0^{max} \leftarrow 0$ 

for  $t = 1$  to ... do
    if maximize :
         $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ 
    else
         $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
        if  $\lambda \neq 0$ 
             $g_t \leftarrow g_t + \lambda \theta_{t-1}$ 
         $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
         $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
         $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
         $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
        if amsgrad
             $\widehat{v}_t^{max} \leftarrow \max(\widehat{v}_t^{max}, \widehat{v}_t)$ 
             $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t^{max}} + \epsilon)$ 
        else
             $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ 

return  $\theta_t$ 
```

I přesto, že označeno jako weight decay, provádí torch.optim.Adam spíše L2 regularizaci

ADAMW

```
CLASS torch.optim.AdamW(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01,
    amsgrad=False, *, maximize=False, foreach=None, capturable=False, differentiable=False,
    fused=None) [SOURCE]
```

Implements AdamW algorithm.

```
input :  $\gamma$ (lr),  $\beta_1, \beta_2$ (betas),  $\theta_0$ (params),  $f(\theta)$ (objective),  $\epsilon$  (epsilon)
        $\lambda$ (weight decay), amsgrad, maximize
initialize :  $m_0 \leftarrow 0$  ( first moment),  $v_0 \leftarrow 0$  ( second moment),  $\widehat{v}_0^{max} \leftarrow 0$ 

for  $t = 1$  to ... do
    if maximize :
         $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ 
    else
         $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
         $\theta_t \leftarrow \theta_{t-1} - \gamma \lambda \theta_{t-1}$ 
         $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
         $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
         $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
         $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
        if amsgrad
             $\widehat{v}_t^{max} \leftarrow \max(\widehat{v}_t^{max}, \widehat{v}_t)$ 
             $\theta_t \leftarrow \theta_t - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t^{max}} + \epsilon)$ 
        else
             $\theta_t \leftarrow \theta_t - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ 

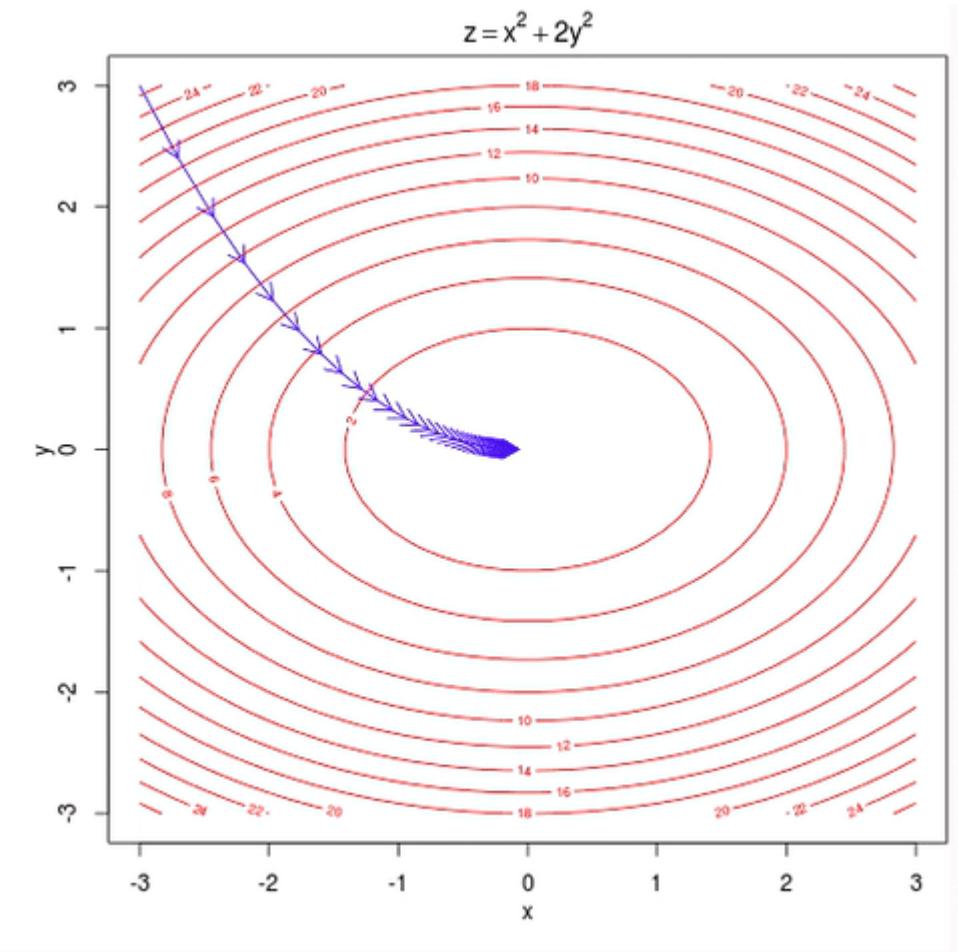
return  $\theta_t$ 
```

weight decay

Learning rate

Velikost kroku γ

- Hyperparametr
- V kontextu sítí obvykle tzv. learning rate
- **Výrazný vliv na výsledný model**
- Typické hodnoty $\gamma \approx 10^{-3}$



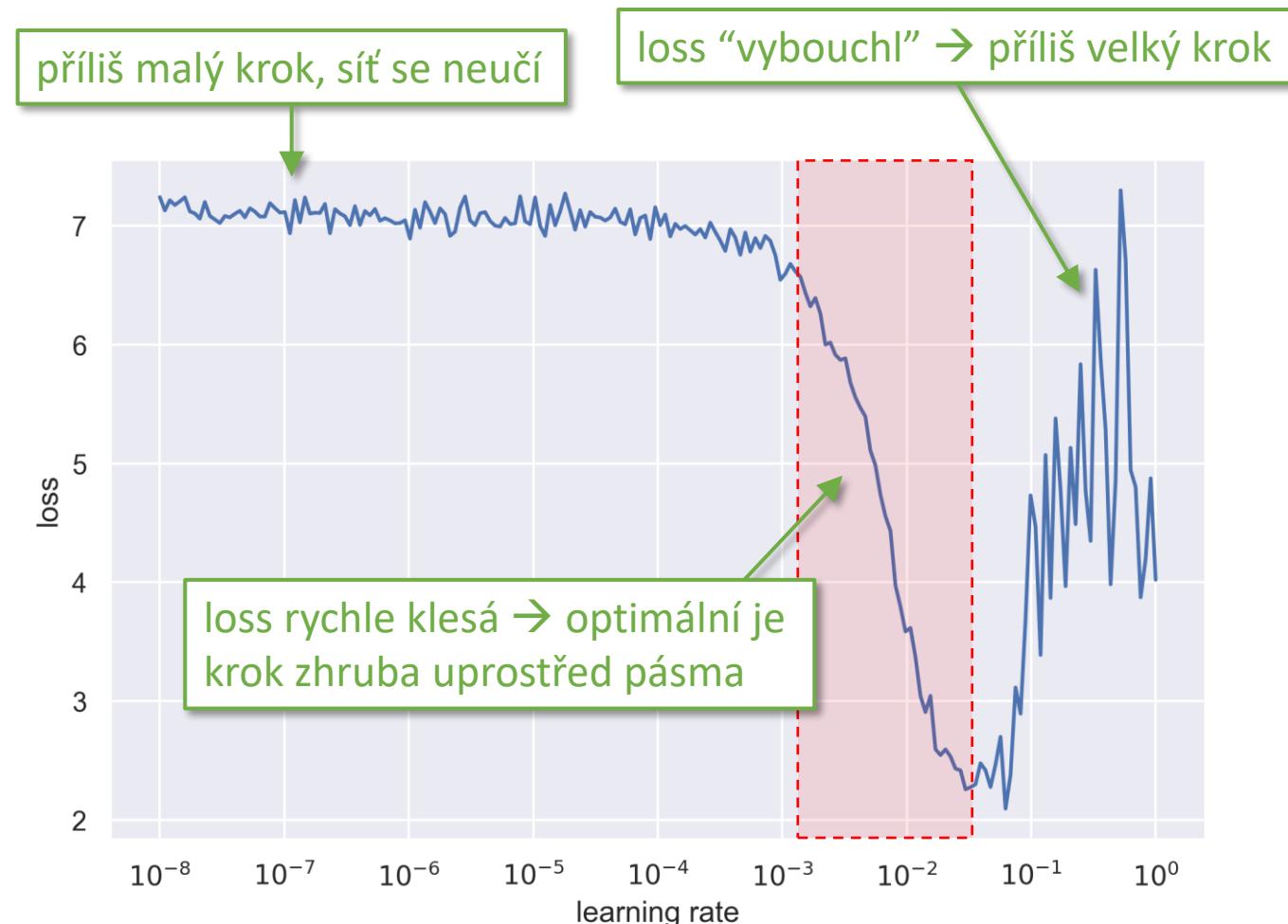
animace: <http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r> (nefunkční)

Jak zvolit krok učení automaticky (learning rate finder)

```
learning_exps = np.linspace(-8, 0, num=200)
data_iter = iter(train_loader)
model = torchvision.models.resnet18()
crit = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(
    model.parameters(),
    lr = 10 ** learning_exps[0]
)
losses = np.zeros((len(learning_exps),))

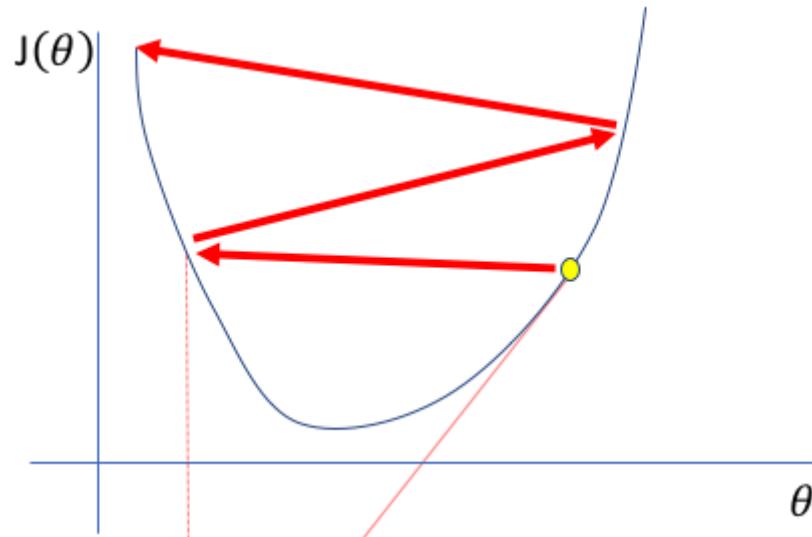
for i, le in enumerate(learning_exps):
    inputs, targets = next(data_iter)
    scores = model(inputs)
    loss = crit(scores, targets)
    losses[i] = float(loss)
    optimizer.zero_grad()
    loss.backward()
    for pg in optimizer.param_groups:
        pg['lr'] = 10 ** le
    optimizer.step()

plt.plot(learning_exps, losses)
```



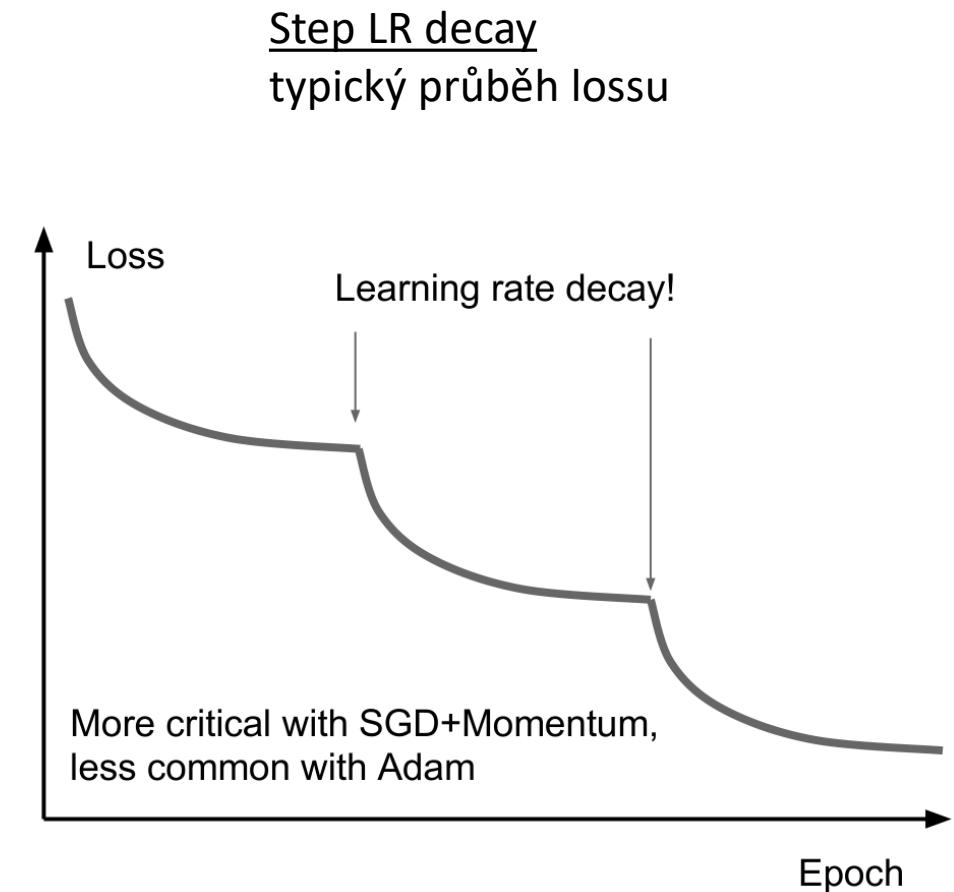
Dynamický krok učení (learning rate scheduling/annealing)

- Pokud během trénování loss neklesá



obrázek: <https://www.jeremyjordan.me/nn-learning-rate/>

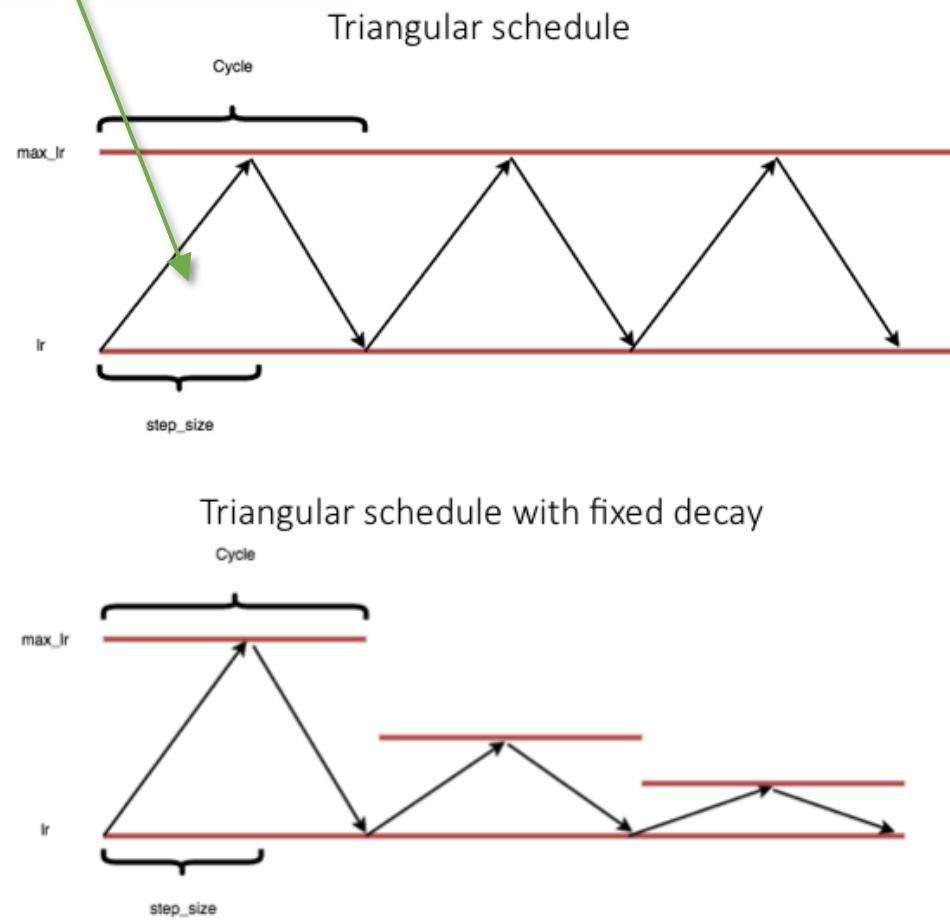
- Je možné zkusit zmenšit learning rate, obvykle např. na polovinu nebo desetinu
- Lze opakovat vícekrát
- Např. 50 epoch lr=0.1, pak 25x lr=0.01 a nakonec 25x lr=0.001 (celkem 100 epoch)



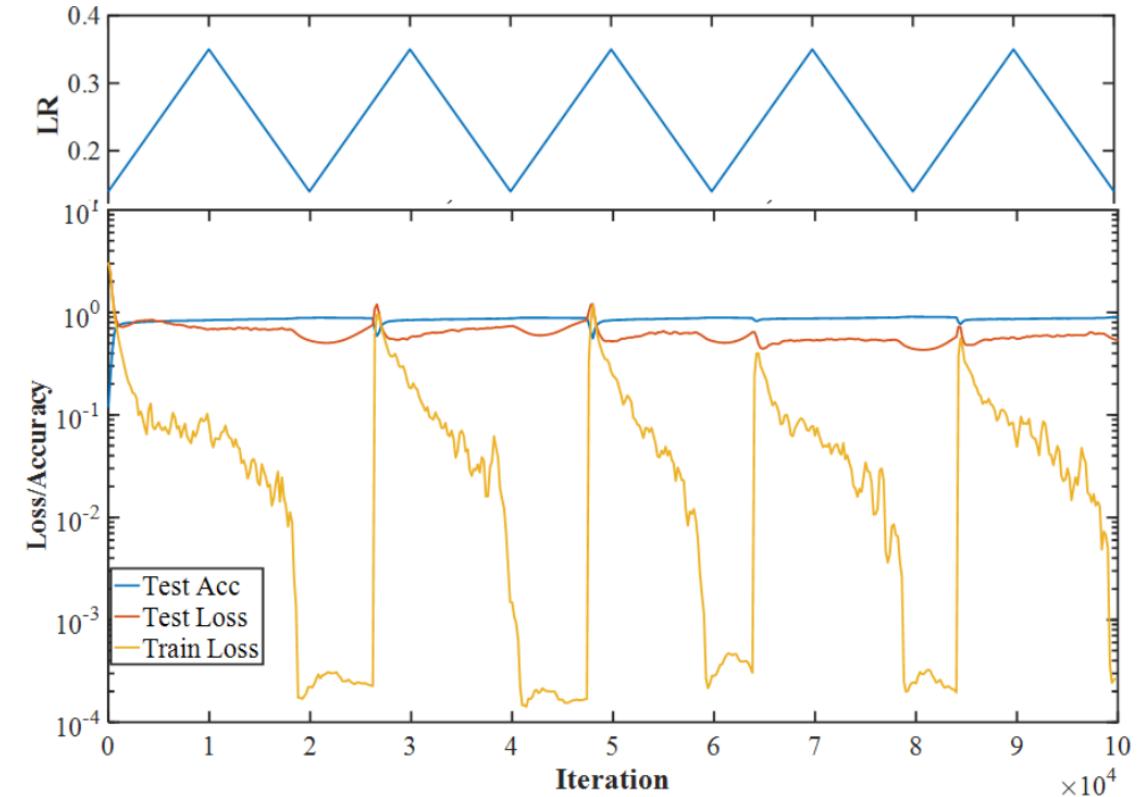
obrázek: <http://cs231n.stanford.edu/>

Cyklický krok učení

batch iterace, ale i epochy



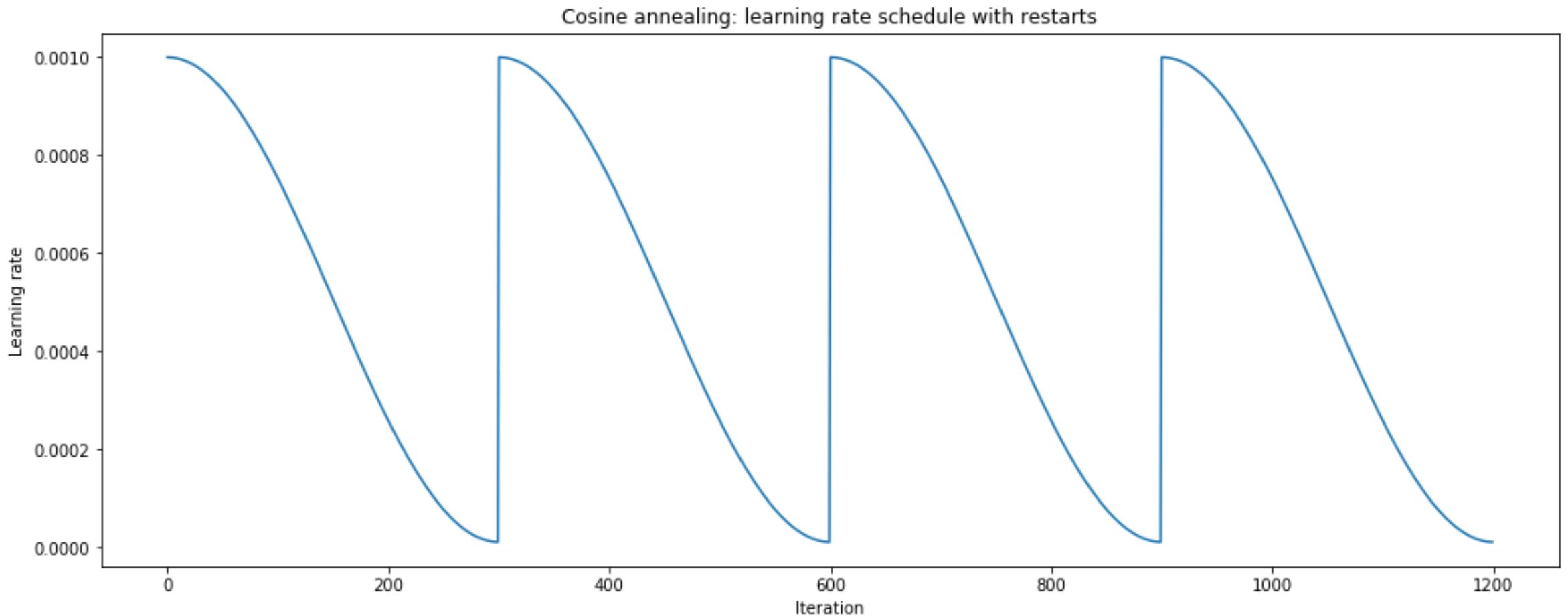
obrázek: <https://www.jeremyjordan.me/nn-learning-rate/>



(a) Cyclical learning rate between $LR=0.1$ and $LR=0.35$ with stepsize=10K.

Smith, Topin: Exploring loss function topology with cyclical learning rates

Stochastic Gradient Descent with Warm Restarts (SGDR)



obrázek + výborný přehled: <https://www.jeremyjordan.me/nn-learning-rate/>

Závěr

Stochastic Gradient descent (SGD) pro multiclass logistickou regresi

Incializujeme:

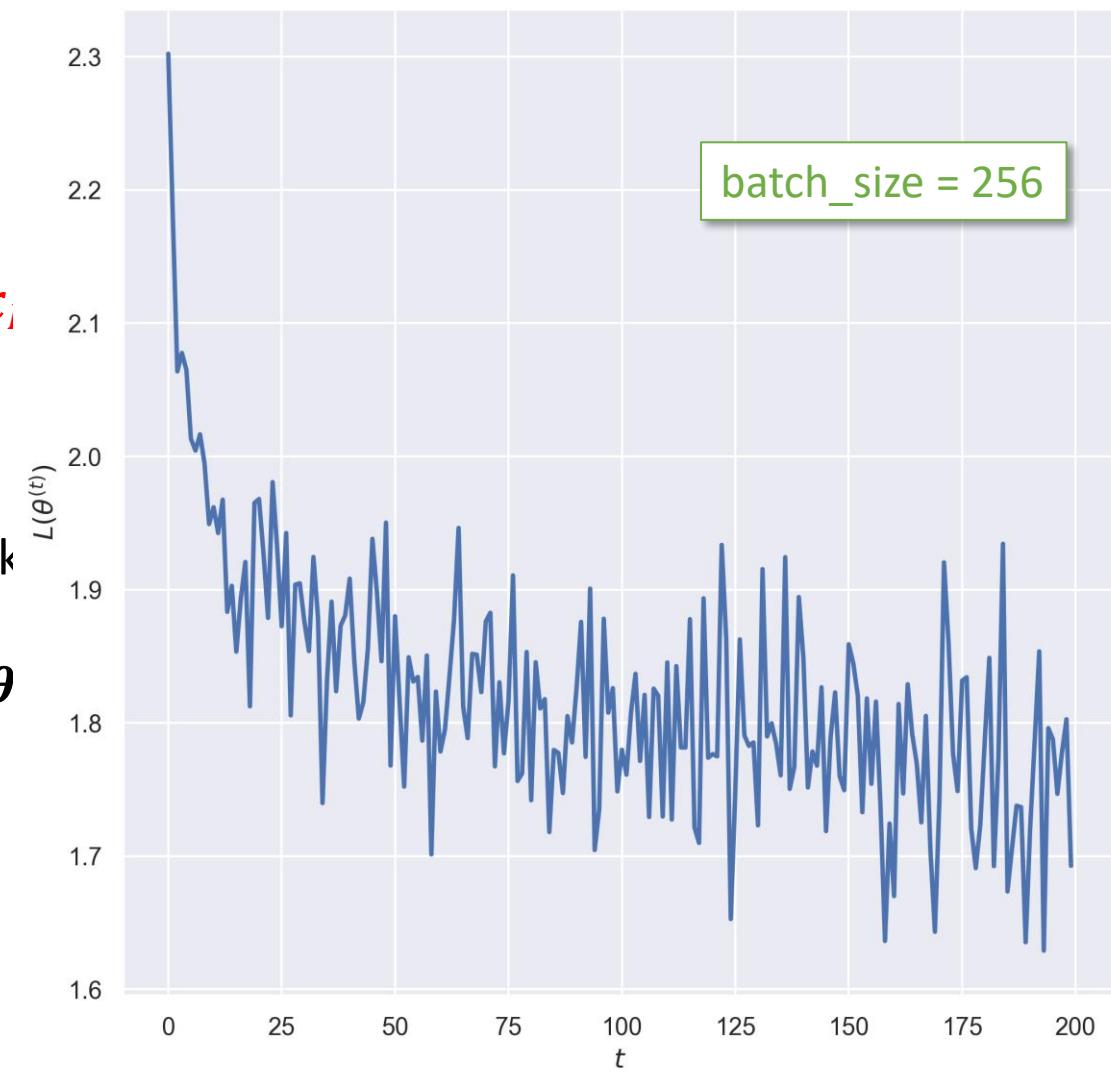
- $\theta = \{W, b\}$ na náhodné hodnoty

Opakujeme:

1. Pro každý vzorek x_n v navzorkované dívce x_1, \dots, x_j
 - a. predikujeme pravděpodobnosti \hat{p}_n
 - b. vypočteme dílčí kritérium l_n a akumulujeme k celkovému l
 - c. vypočteme dílčí gradient ∇L_n a akumulujeme k celkovému ∇L
2. updatujeme dle zvoleného optimizéra parametry θ akumulovaným gradientem ∇L s krokem γ

Zastavíme:

- po fixním počtu iterací
- parametry θ se ustálí
- hodnota kritéria $L(\theta)$ již delší dobu neklesá



Závěr

- Numerický gradient je pro neurosítě příliš výpočetně náročný → potřebujeme analytický gradient
- Existují rozšíření SGD založené na momentum (Momentum SGD, Nesterov) a adaptivní metody (AdaGrad, RMSprop, Adam/W)
- Jako výchozí volba pro optimalizaci je díky své robustnosti vůči nastavení hyperparametrů obvykle nevhodnější Adam/AdamW
- Krok učení (learning rate) volíme tak, aby loss poměrně rychle klesal. Pokud klesat přestane, krok zmenšíme např. na desetinu a pokračujeme (tzv. learning rate decay scheduling)
- Momentum SGD s pokročilejším lr-schedulingem lépe nechat na konec na poslední “ždímání” skóre