

Neurónové siete pre počítačové videnie

Trénovanie neurónových sietí II

RNDr. Zuzana Černeková, PhD.

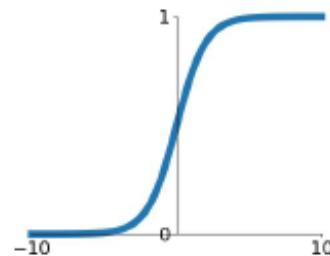
Ing. Viktor Kocur, PhD.

Opakovanie

- Aktivačné funkcie

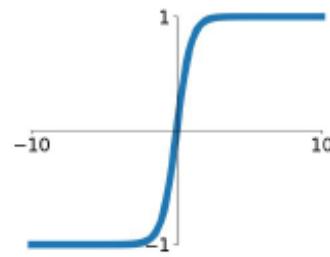
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



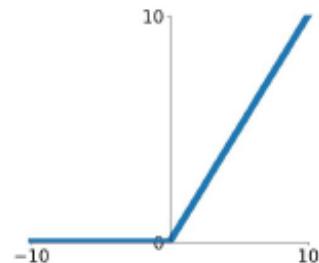
tanh

$$\tanh(x)$$



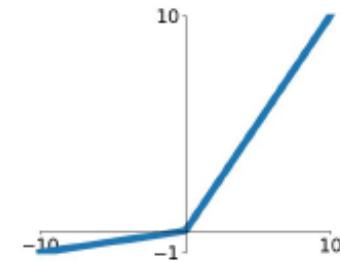
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

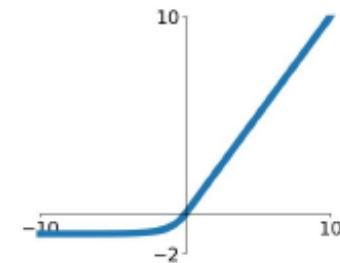


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

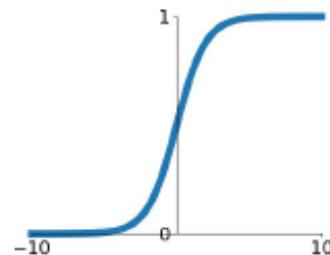


Opakovanie 2

- Aktivačné funkcie

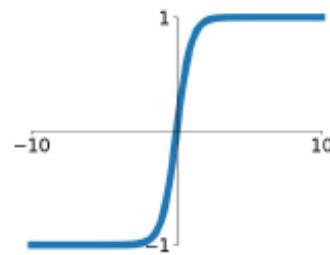
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

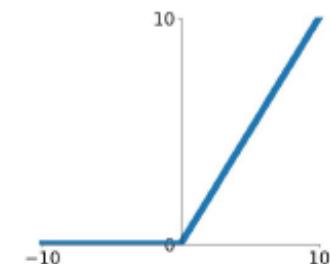
$$\tanh(x)$$



ReLU

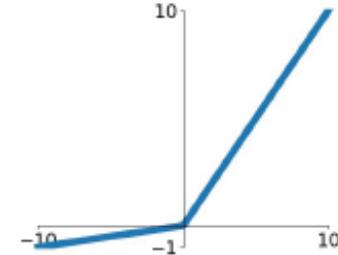
$$\max(0, x)$$

Dobrý počiatočný výber



Leaky ReLU

$$\max(0.1x, x)$$

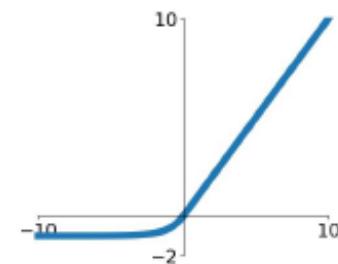


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

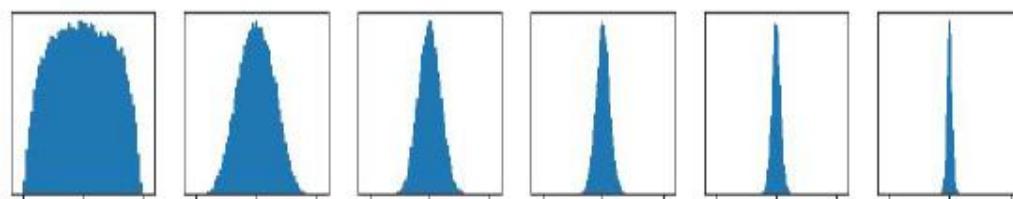
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



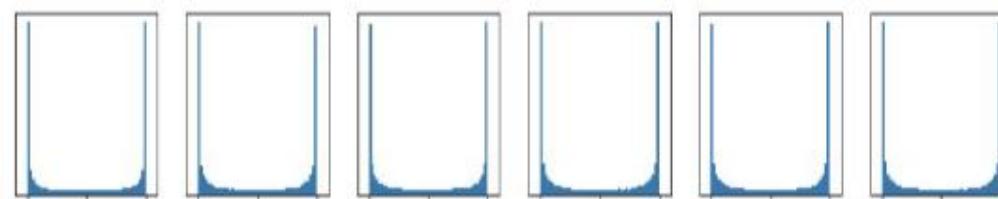
Opakovanie 3

• Inicializácia váh



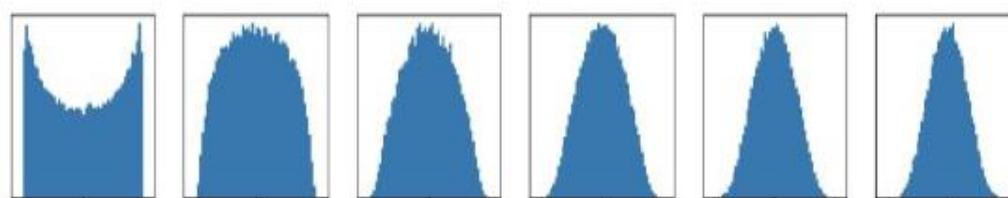
Inicializácie príliš malé:

Aktivácie aj gradienty idú k 0
Žiadne učenie ☹



Inicializácie príliš veľké:

Aktivácie sú saturované (\tanh),
gradienty 0, **žiadne učenie** ☹

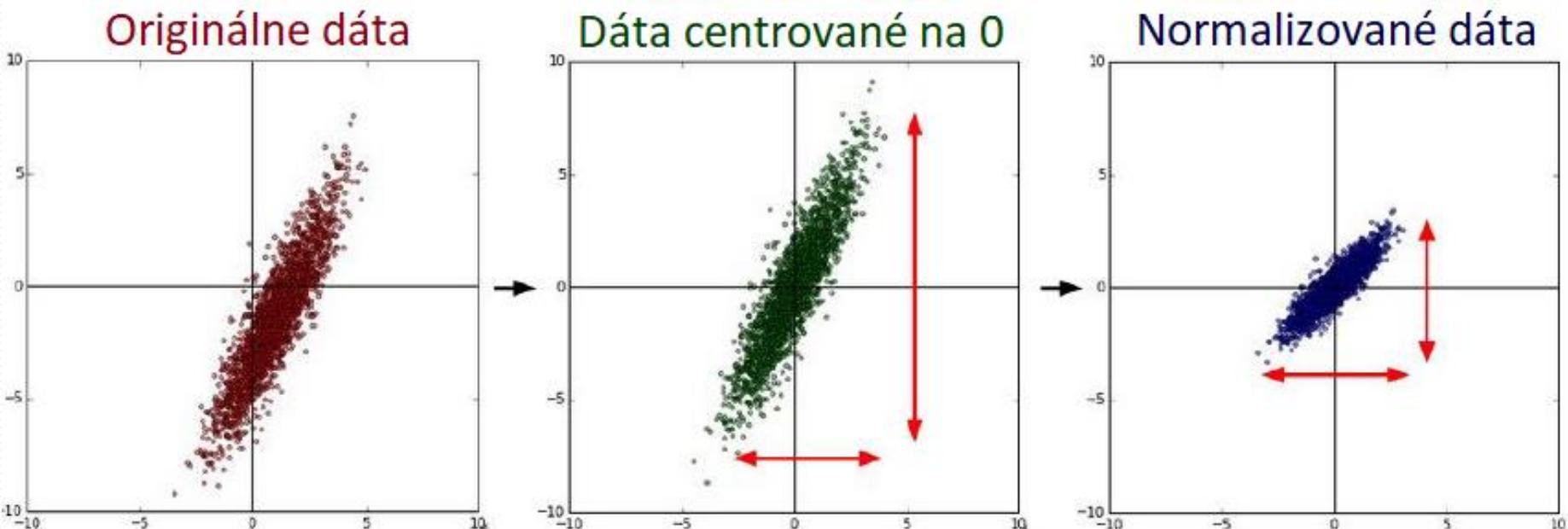


Inicializácie akurát:

Pekná distribúcia aktivácií vo
všetkých vrstvách, učenie sa
vyvíja pekne

Opakovanie 4

• Predspracovanie dát



```
X -= np.mean(X, axis = 0) . X /= np.std(X, axis = 0)
```

Opakovanie 5 (BN)

Vstup: $x: N \times D$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Stredná hodnota po kanáloch, rozmer je D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Odchýlka po kanáloch, rozmer je D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalizované x , rozmer je $N \times D$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Výstup, rozmer je D

Parametre škály a posunu nastavované učením
 $\gamma, \beta: D$
 $\gamma = \sigma, \beta = \mu$
znamená identickú funkciu = žiadnu BN

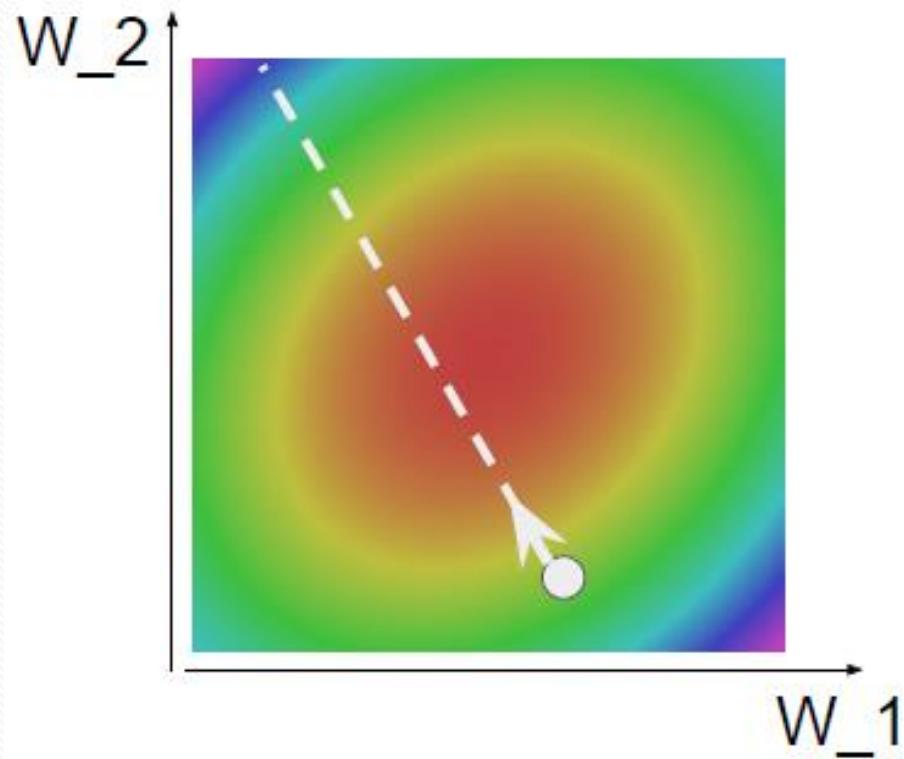
Štruktúra dnešnej prednášky

- **Zlepšenie trénovacej chyby:**
 - Algoritmy optimalizácie
 - Schémy na nastavenie kroku učenia
- **Zlepšenie testovacej chyby:**
 - Regularizácia
 - Výber hyperparametrov

Optimalizácia

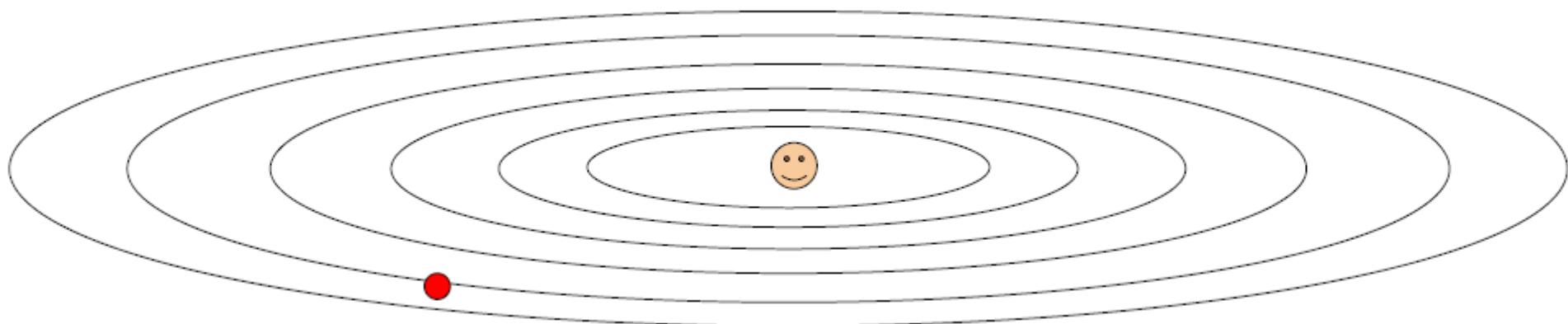
```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```



Optimalizácia: problémy s SGD

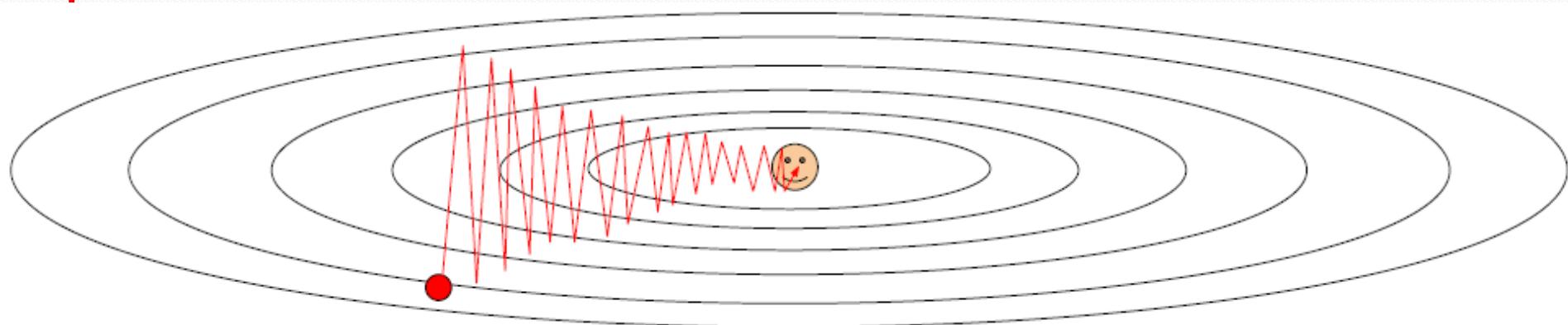
- Čo keď sa strata mení rýchlo v jednom smere a pomaly v druhom? Čo vtedy robí zostup gradientu?



- Stratová funkcia má veľké **podmienené číslo**: pomery medzi najväčšou a najmenšou singulárhou hodnotou v matici Hessiánu je veľký

Optimalizácia: problémy s SGD

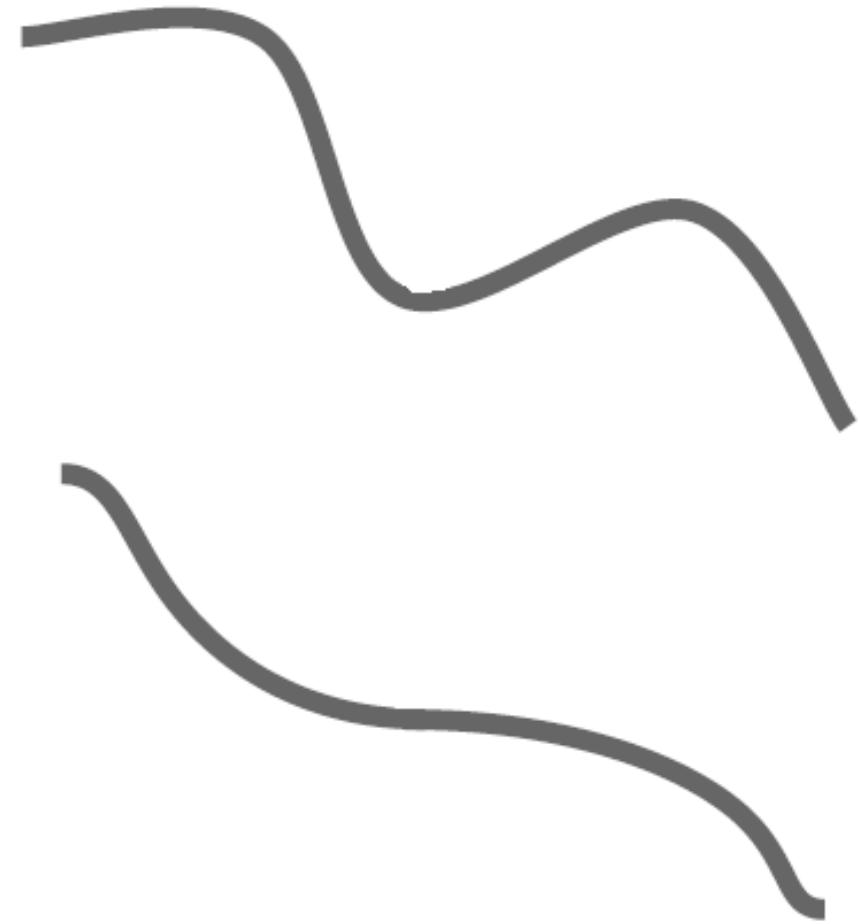
- Čo keď sa strata mení rýchlo v jednom smere a pomaly v druhom? Čo vtedy robí zostup gradientu?
Veľmi pomalý progres cez plytký rozmer a cik-cak cez strmý rozmer. Vo vysokých rozmeroch častejší problém.



- Stratová funkcia má veľké **podmienené číslo**: pomery medzi najväčšou a najmenšou singulárnoch hodnotami v matici Hessiánu je veľký

Optimalizácia: problémy s SGD

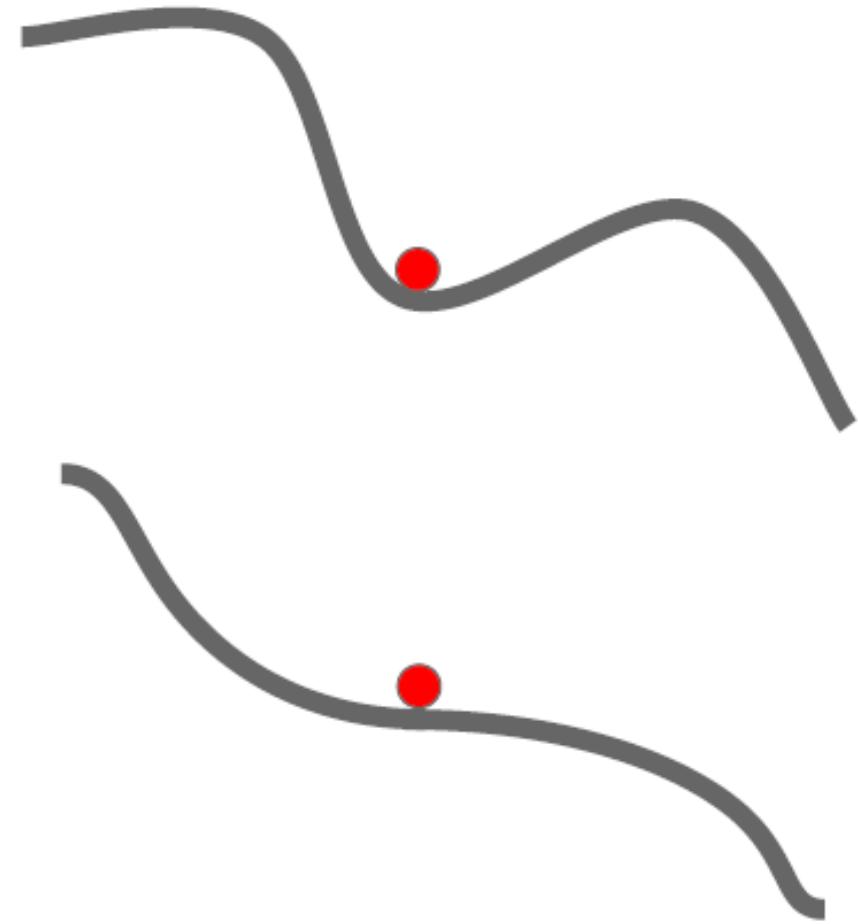
Čo ked' stratová funkcia
má **lokálne minimá**
alebo **sedlový bod**?



Optimalizácia: problémy s SGD

Čo ked' stratová funkcia
má **lokálne minimá**
alebo **sedlový bod**?

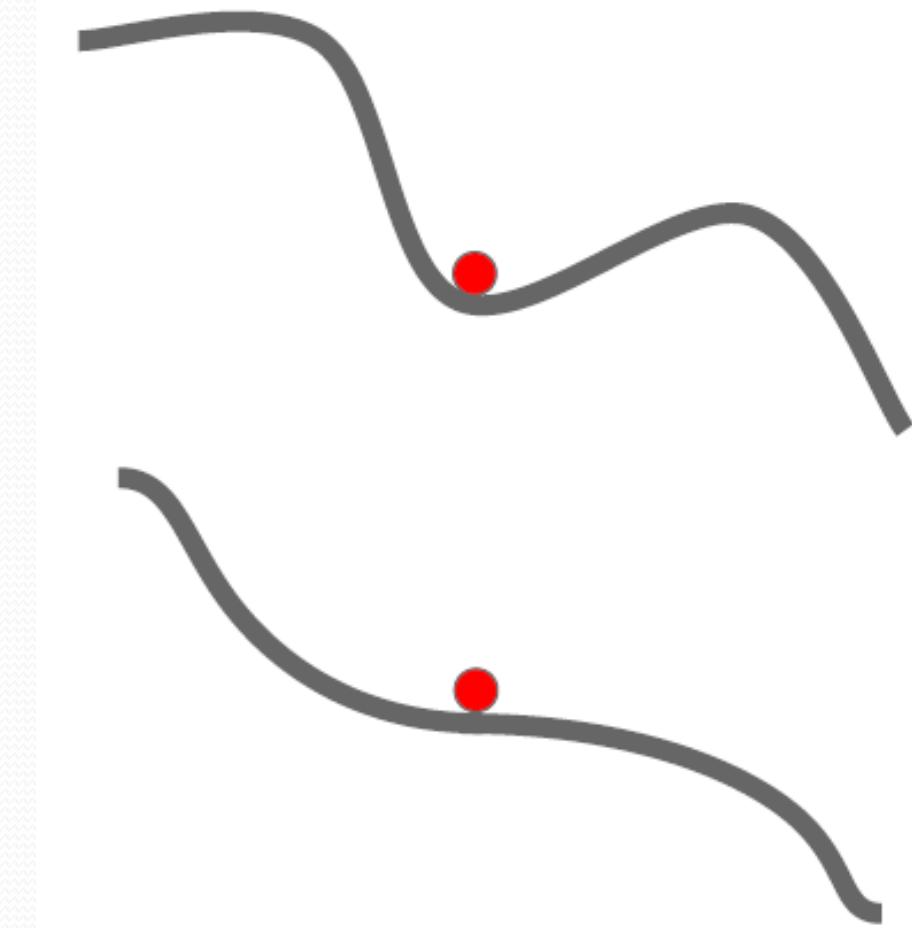
Nulový gradient,
zostup gradientu sa
zasekne



Optimalizácia: problémy s SGD

Čo ked' stratová funkcia
má **lokálne minimá**
alebo **sedlový bod**?

Vo vyšších rozmeroch
sú sedlové body
častejšie ako lokálne
minimá

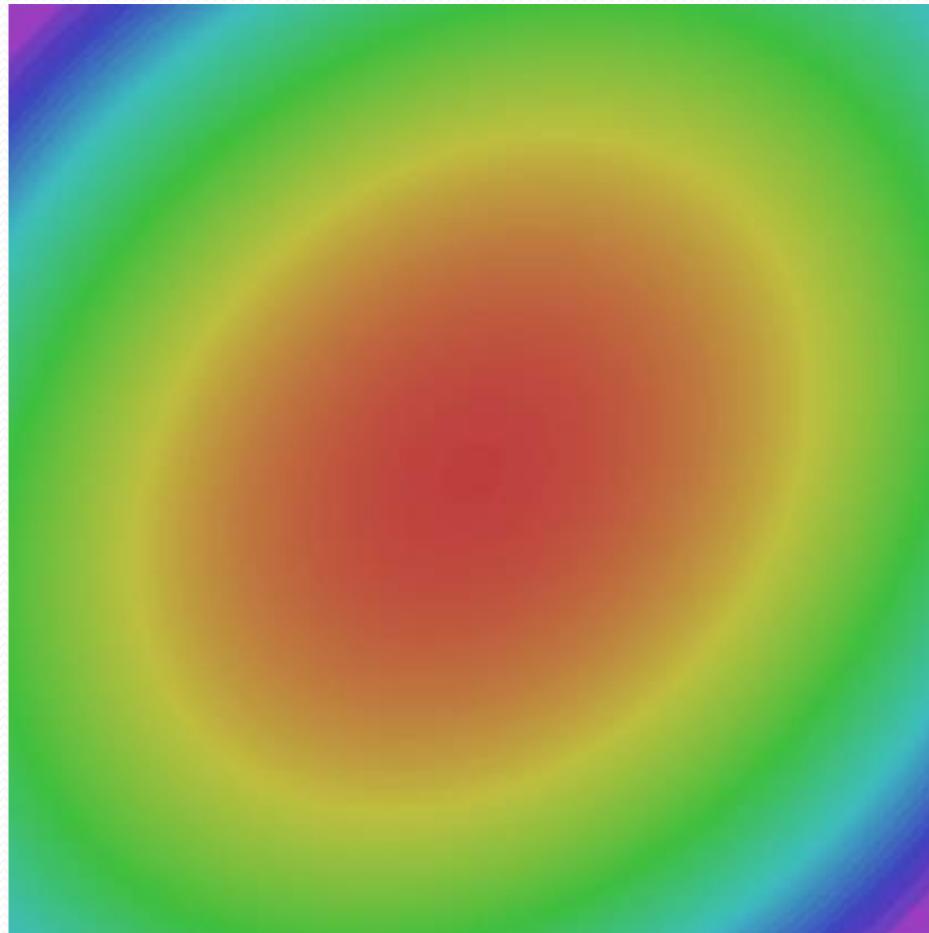


Optimalizácia: problémy s SGD

Naše gradienty pochádzajú z minidávok a môžu byť zašumené!

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



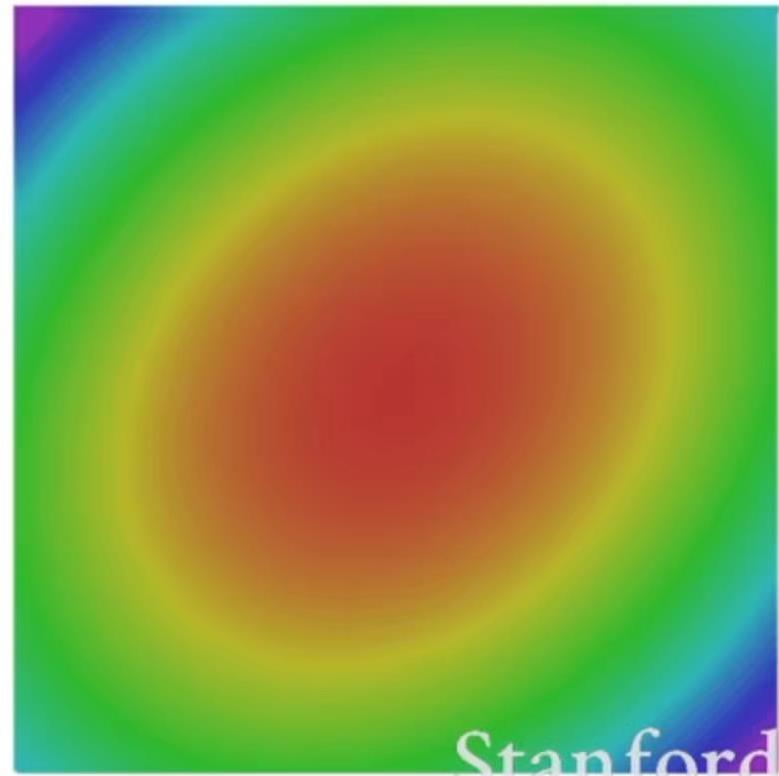
Optimalizácia: problémy s SGD

Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



Stanford

University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 20

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

$$vx = 0$$

```
while True:
```

```
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x -= learning_rate * vx
```

- Vytvára „rýchlosť“ ako priebežný priemer gradientov
- ρ pridáva „trenie“; typicky sa $\rho = 0,9$ alebo $0,99$

SGD + Momentum

SGD+Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx - learning_rate * dx
    x += vx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

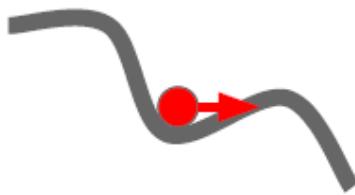
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

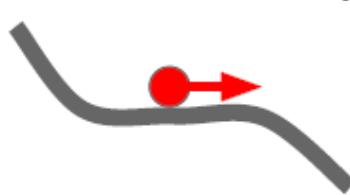
Ukážka SGD+Momentum v dvoch formuláciách, ale tie sú ekvivalentné - dávajú rovnakú postupnosť x

SGD + Momentum

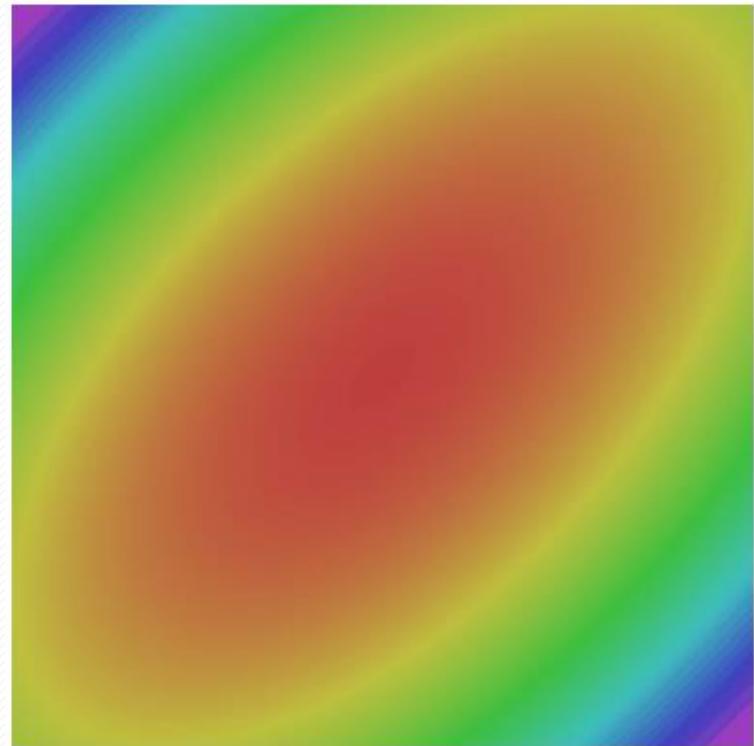
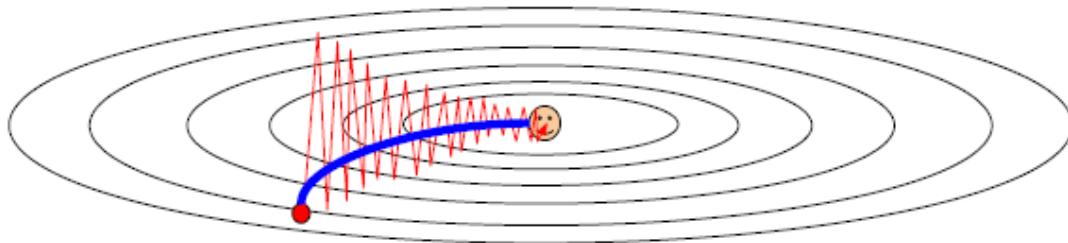
Lokálne minimá



Sedlové body



Slabé výsledky



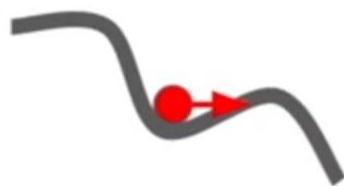
SGD

SGD+Momentum

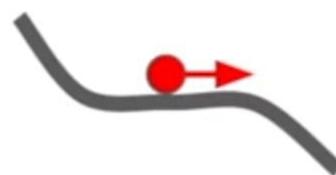
SGD + Momentum

SGD + Momentum

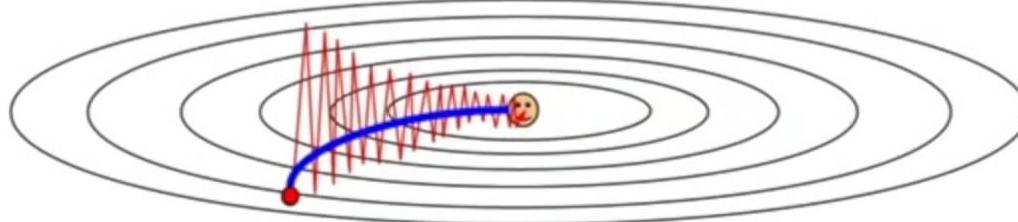
Local Minima



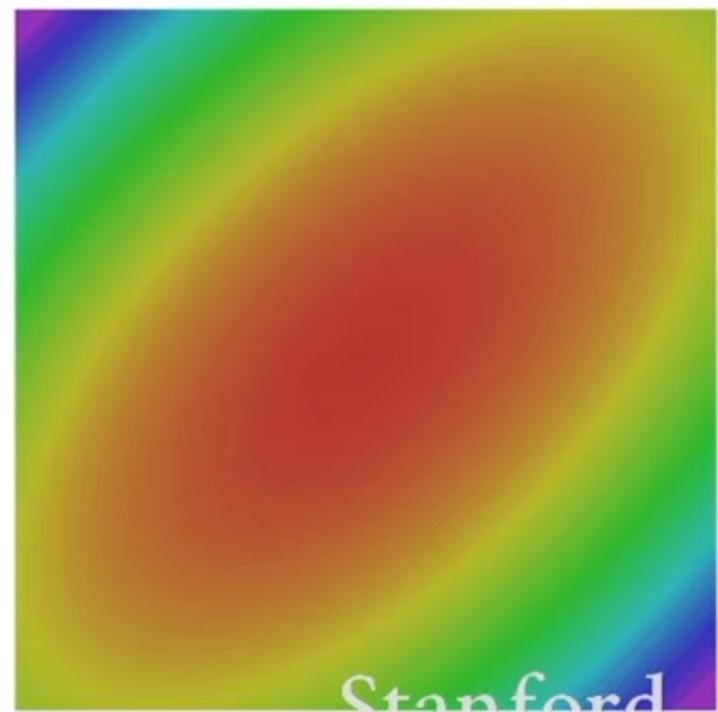
Saddle points



Poor Conditioning

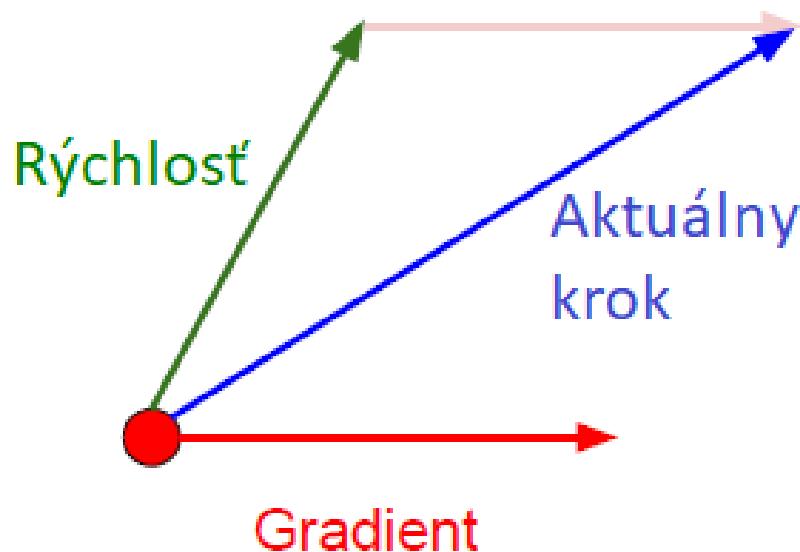


Gradient Noise



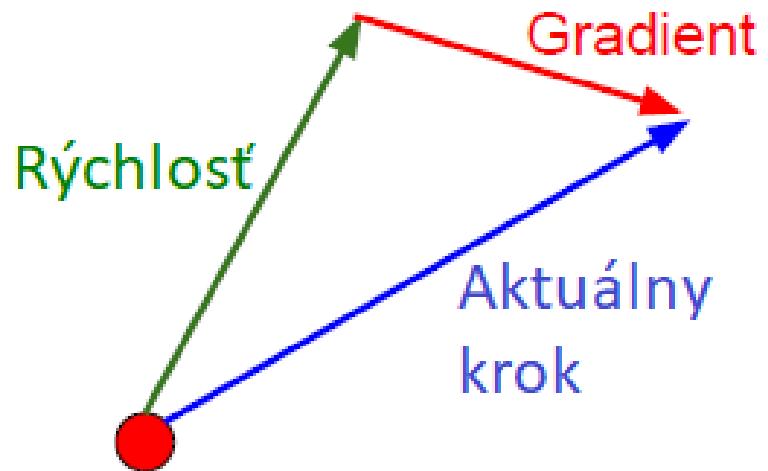
Nesterovovo momentum

Aktualizácia momentu



Kombinuje gradient v danom bode s rýchlosťou na získanie kroku na aktualizáciu váh

Nesterovovo momentum

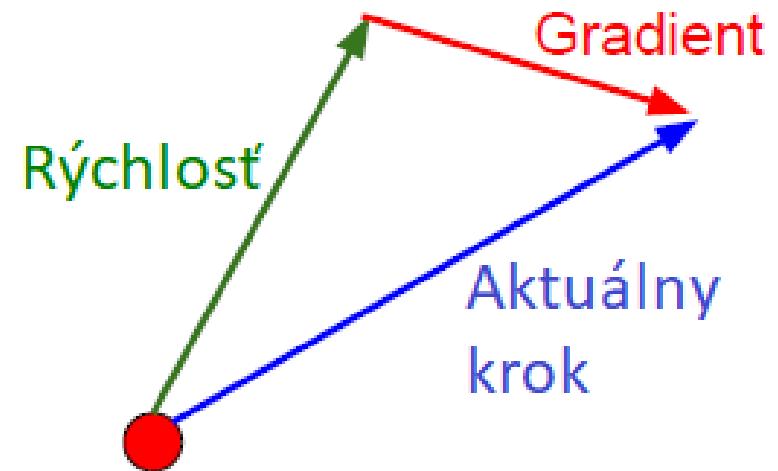


„Pozerá“, kam by nás zobraťa rýchlosť, tam vypočítá gradient a spojí to s rýchlosťou na výpočet smeru aktualizácie

Nesterovovo momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$



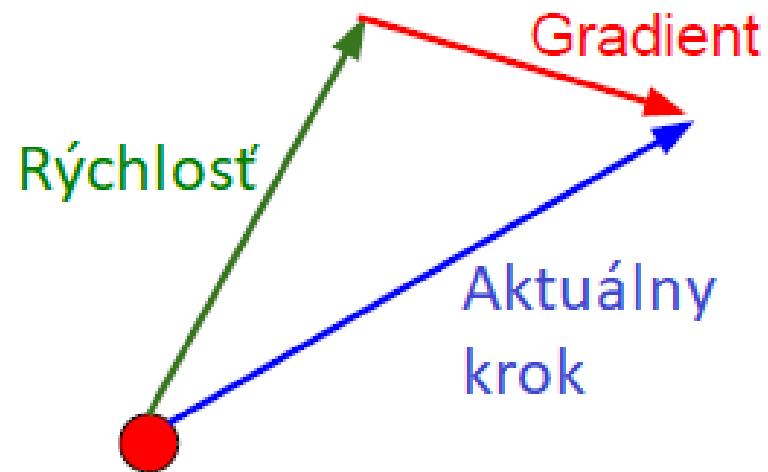
„Pozerá“, kam nás by nás zo-brala rýchlosť, tam vypočíta gradient a spojí to s rýchlosťou na výpočet smeru aktualizácie

Nesterovovo momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Ale obvykle chceme aktualizovať pomocou x_t a $\nabla f(x_t)$



„Pozerá“, kam nás by nás zoobrala rýchlosť, tam vypočíta gradient a spojí to s rýchlosťou na výpočet smeru aktualizácie

Nesterovovo momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

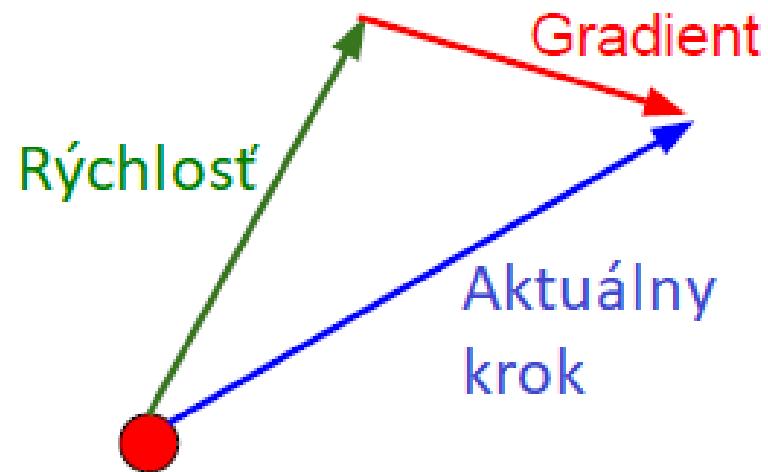
$$x_{t+1} = x_t + v_{t+1}$$

Zmeníme premenné $\tilde{x}_t = x_t + \rho v_t$ a preformulujeme

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\begin{aligned}\tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)\end{aligned}$$

Ale obvykle chceme aktualizovať pomocou x_t a $\nabla f(x_t)$



„Pozerá“, kam nás by nás zoobrala rýchlosť, tam vypočítá gradient a spojí to s rýchlosťou na výpočet smeru aktualizácie

Nesterovovo momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Ale obvykle chceme aktualizovať pomocou x_t a $\nabla f(x_t)$

Zmeníme premenné $\tilde{x}_t = x_t + \rho v_t$ a prefomulujeme

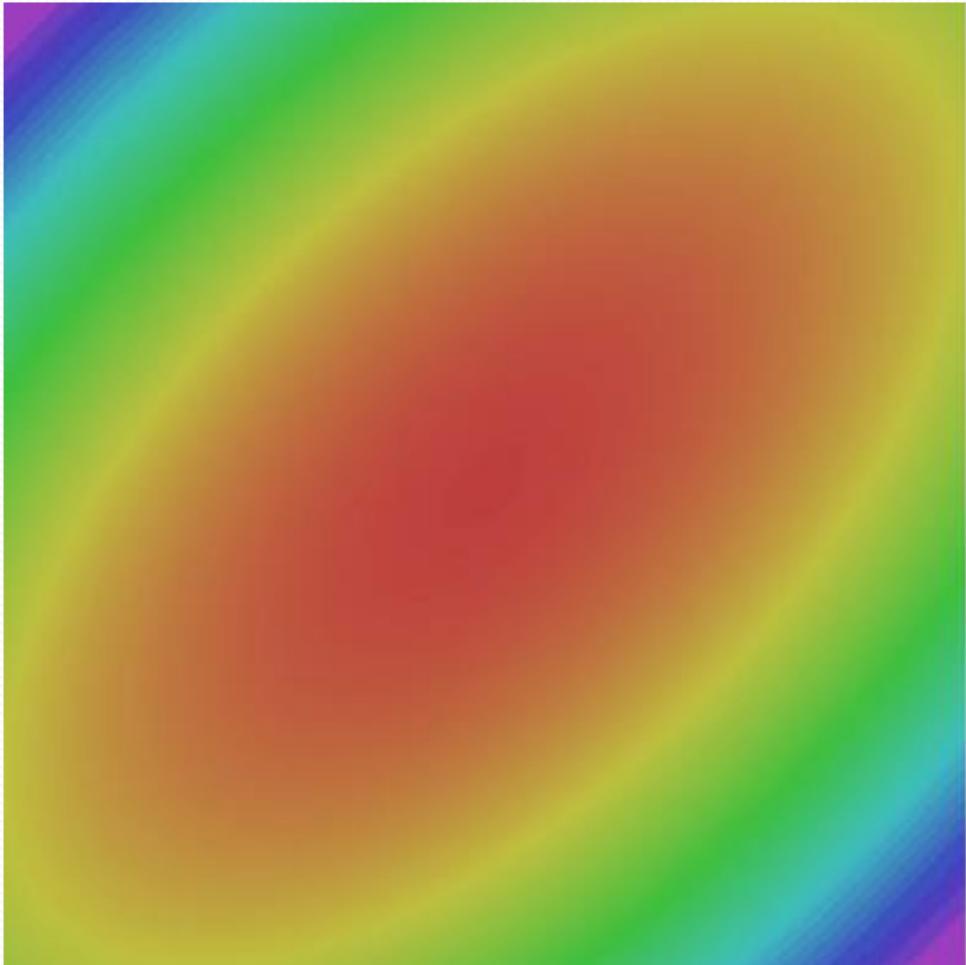
$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1}$$

$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

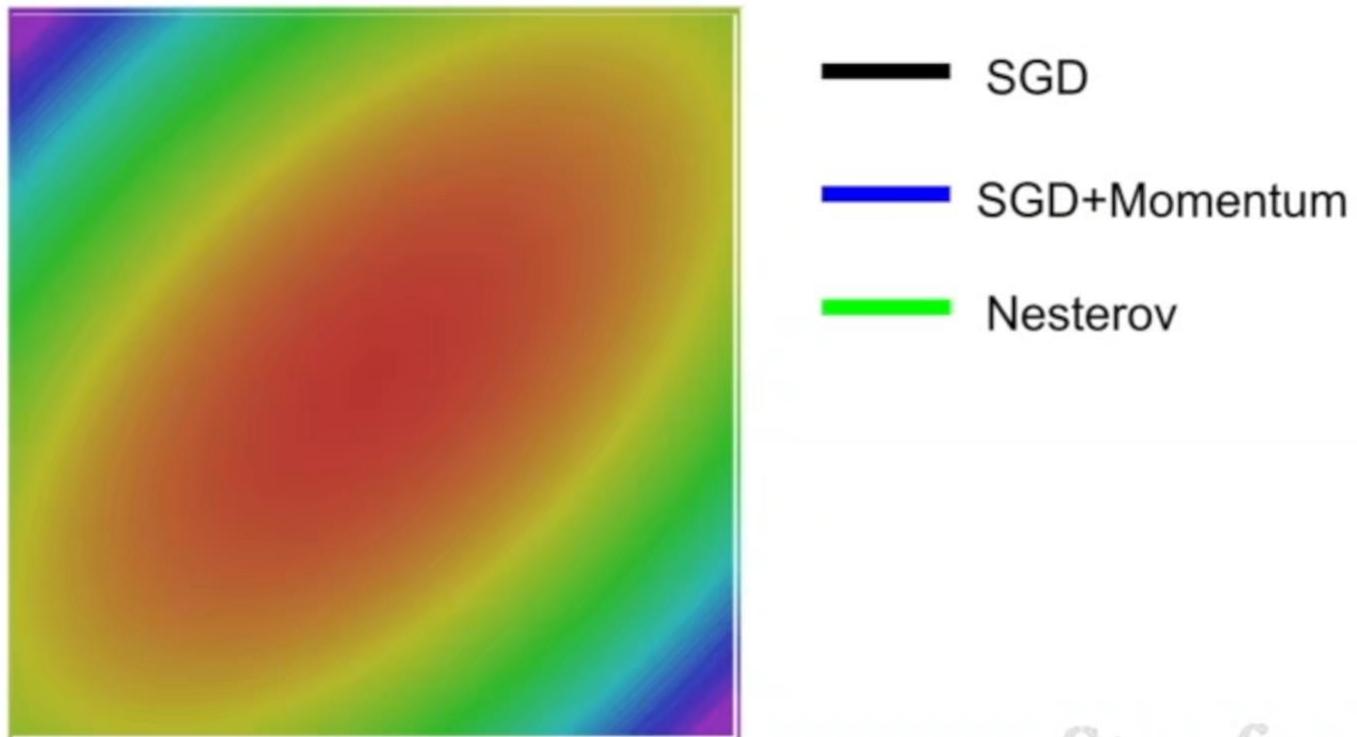
Nesterovovo momentum



- SGD
- SGD+Momentum
- Nesterov

Nesterovovo momentum

Nesterov Momentum



Fei-Fei Li & Justin Johnson & Serena Yeung

Stanford University April 20, 2017
Lecture 6 - 28

AdaGrad

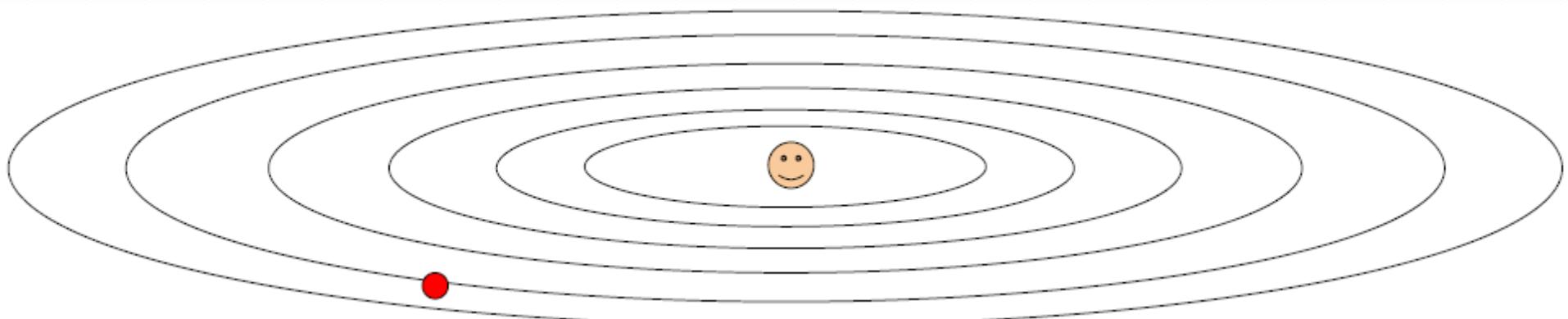
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Pridáva škálovanie gradientu po zložkách
založené na historických sumách
štvorcov v každom rozmere

„Kroky učenia podľa parametrov“ alebo
“adaptívne kroky učenia“

AdaGrad

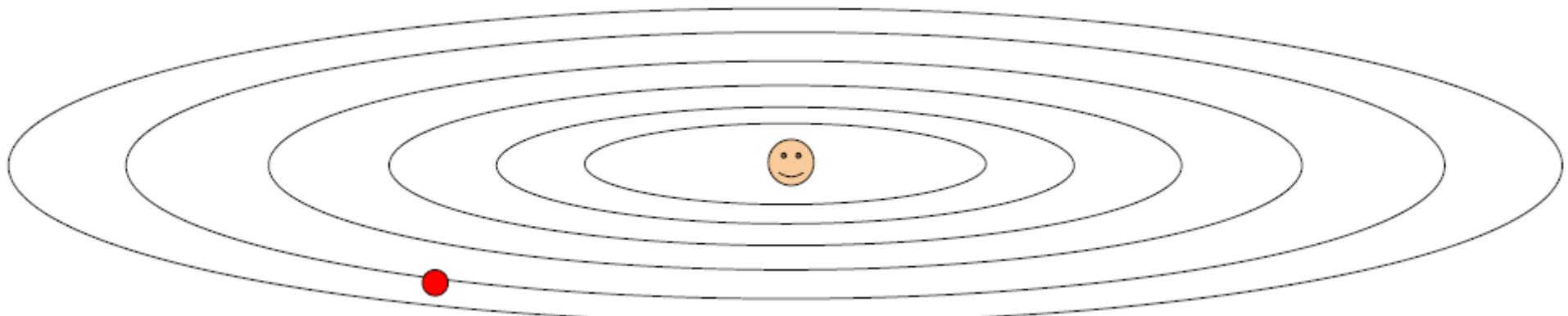
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Ot: Čo sa stane s AdaGrad?

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

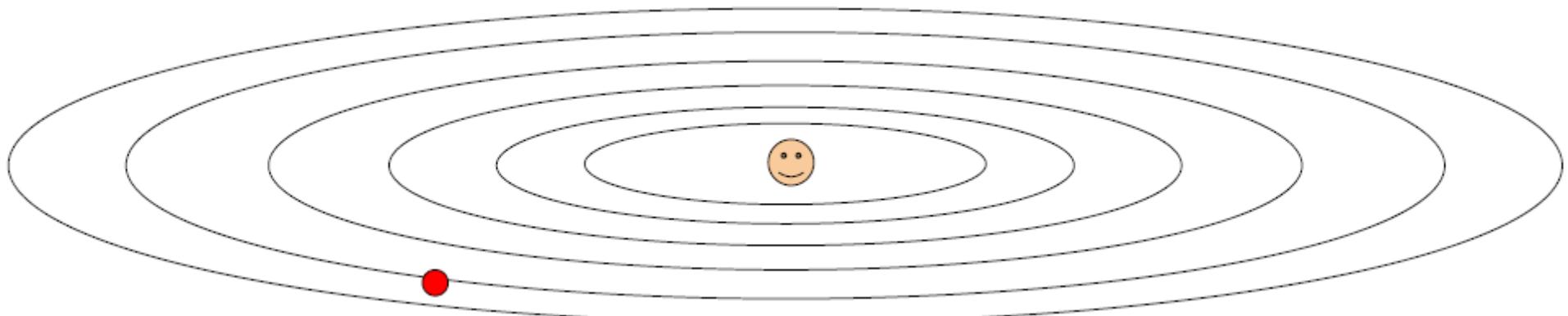


Ot: Čo sa stane s AdaGrad?

Postup cez „strmé smery“ sa spomalí
Postup cez „plytké smery“ sa zrýchli

AdaGrad

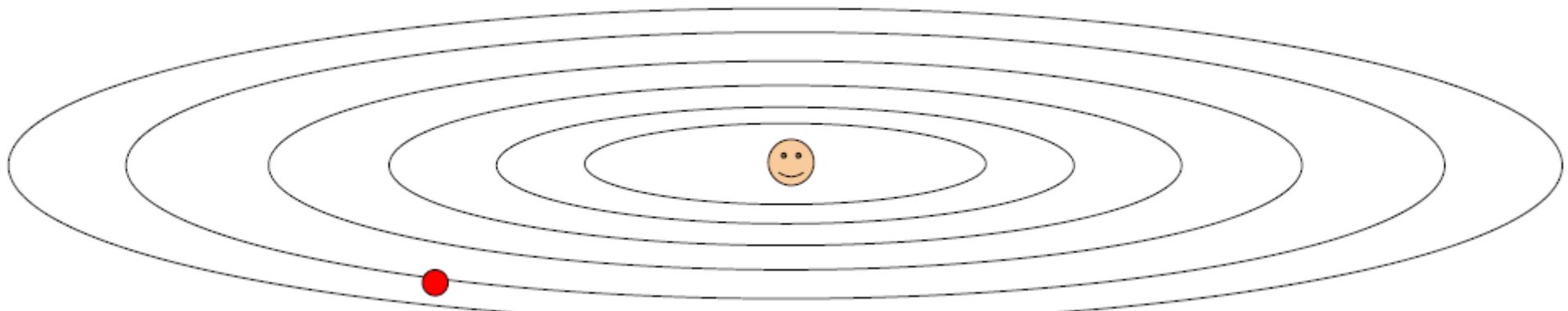
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Ot: Čo sa stane s veľkosťou kroku po
dlhšom čase?

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Ot: Čo sa stane s veľkosťou kroku po dlhšom čase? Zníži sa na malú hodnotu

RMSProp: „presakujúci“ AdaGrad

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

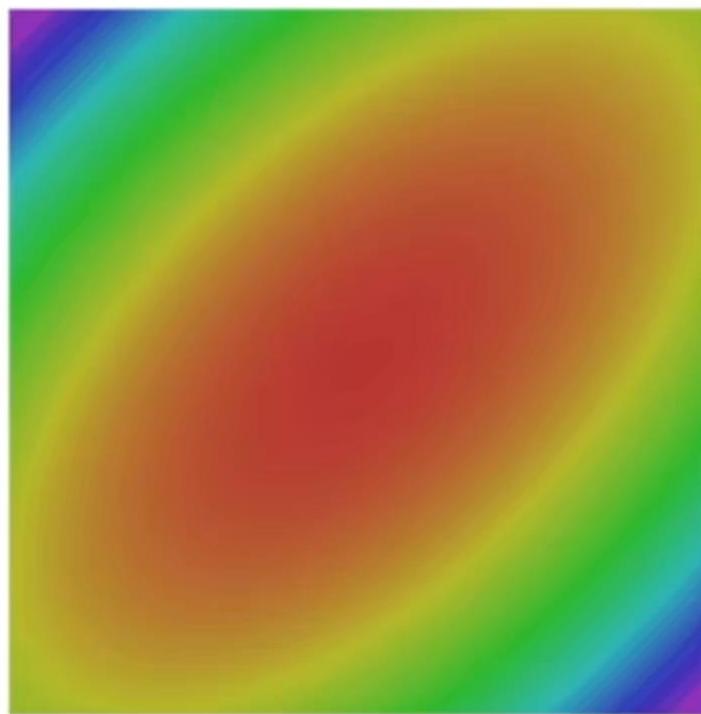


RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

RMSProp



- SGD
- SGD+Momentum
- RMSProp

Stanford
University April 26, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 33

Adam (takmer)

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Adam (takmer)

The diagram illustrates the Adam optimization update rule. It shows two parallel arrows pointing upwards from the AdaGrad/RMSProp and Momentum components to a single red box containing the full Adam update code.

Momentum

AdaGrad/RMSProp

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7)
```

Niečo ako RMSProp + Momentum

Ot: Čo sa stane v prvom časovom kroku?

Adam (plná forma)

Momentum, Korekcia prahu, AdaGrad/RMSProp

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Korekcia prahu preto, lebo
odhad prvého a druhého
momentu sa začína na nule

Adam (plná forma)

Momentum, Korekcia prahu, AdaGrad/RMSProp

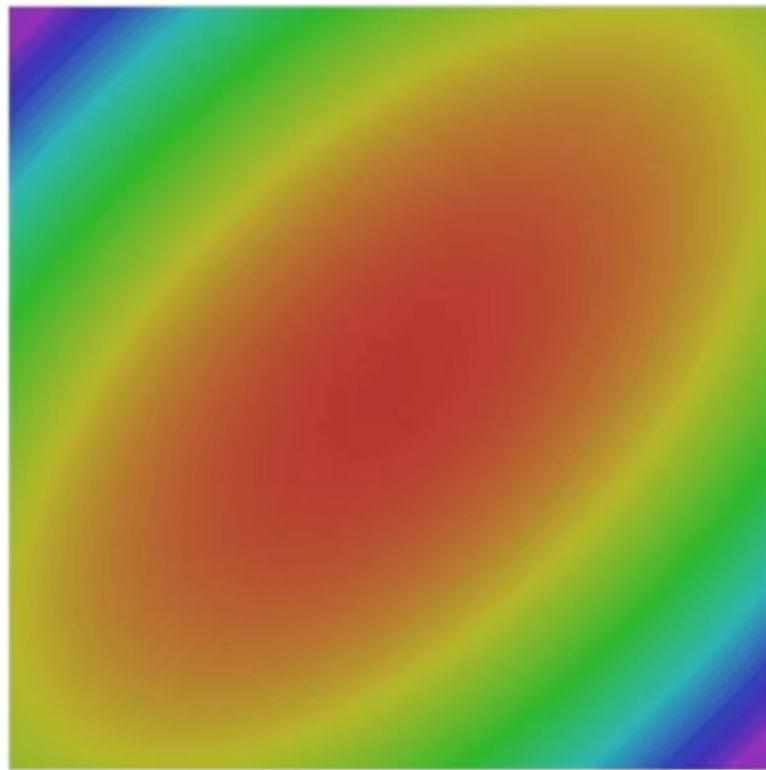
```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

Korekcia prahu preto, lebo odhad prvého a druhého momentu sa začína na nule

Adam s Beta1 = 0,9, Beta2 = 0,999 a krokom učenia 1e-3 al. 1e-4 je výborný štartovací bod pre mnoho modelov

Adam

Adam



- SGD
- SGD+Momentum
- RMSProp
- Adam

Stanford

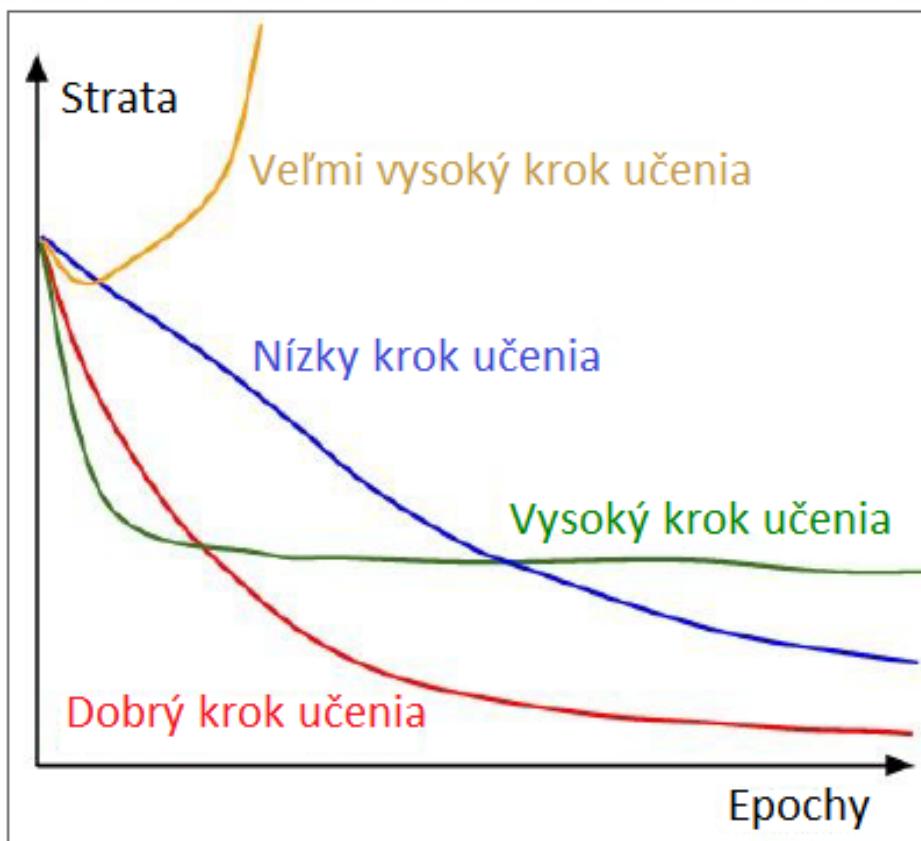
University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 38

Krok učenia

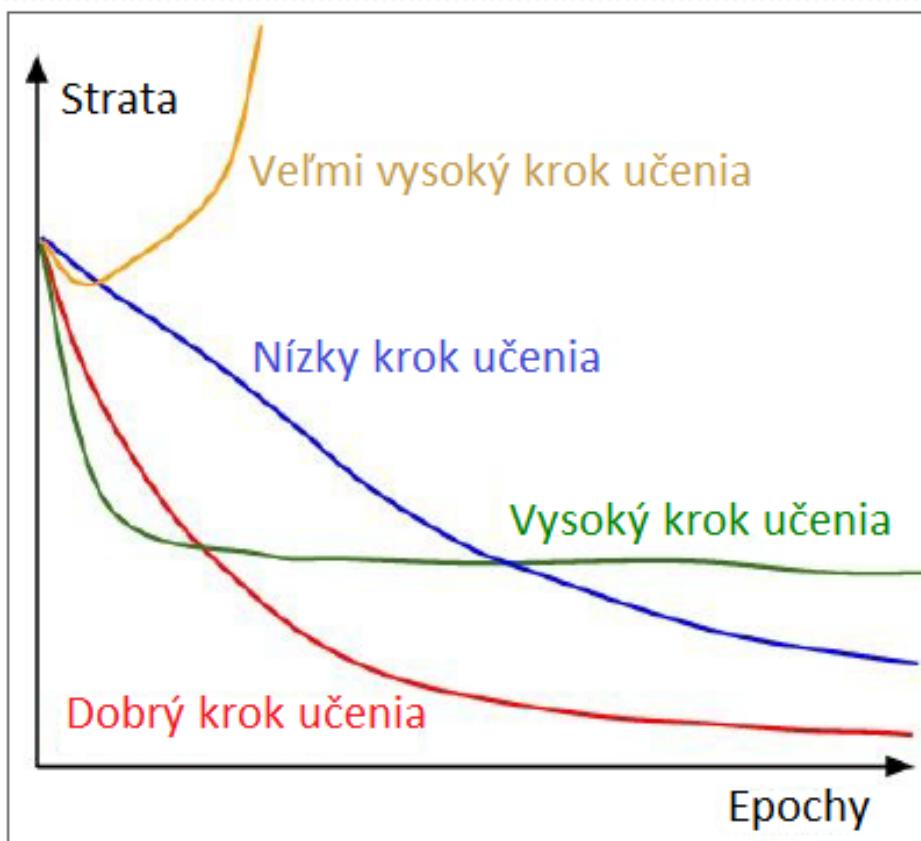
SGD, SGD+Momentum, AgaGrad, RMSProp, Adam majú krok učenia α ako hyperparameter



Ot: Ktorý z týchto krokov učenia je najlepšie použiť?

Krok učenia

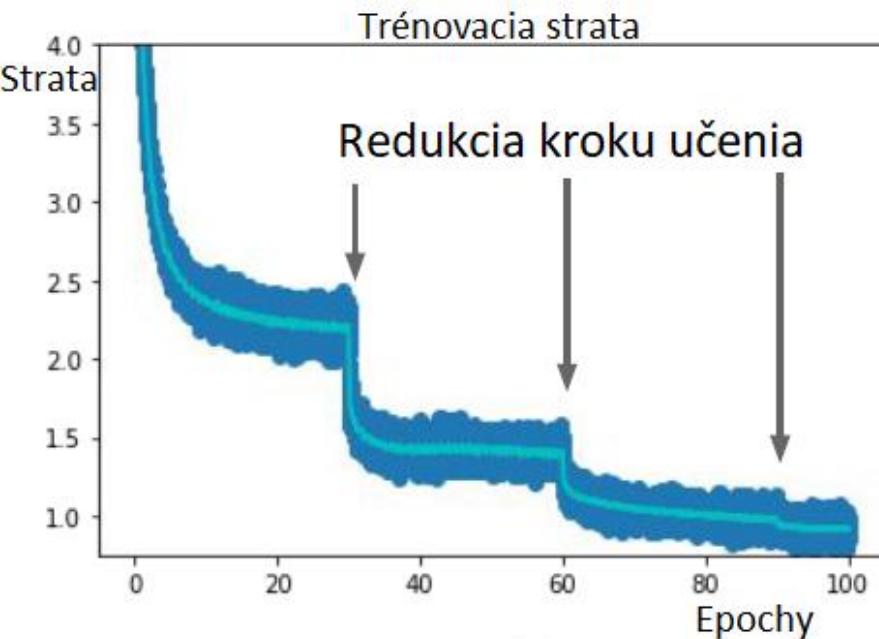
SGD, SGD+Momentum, AgaGrad, RMSProp, Adam majú krok učenia ako hyperparameter



Ot: Ktorý z týchto krokov učenia je najlepšie použiť?

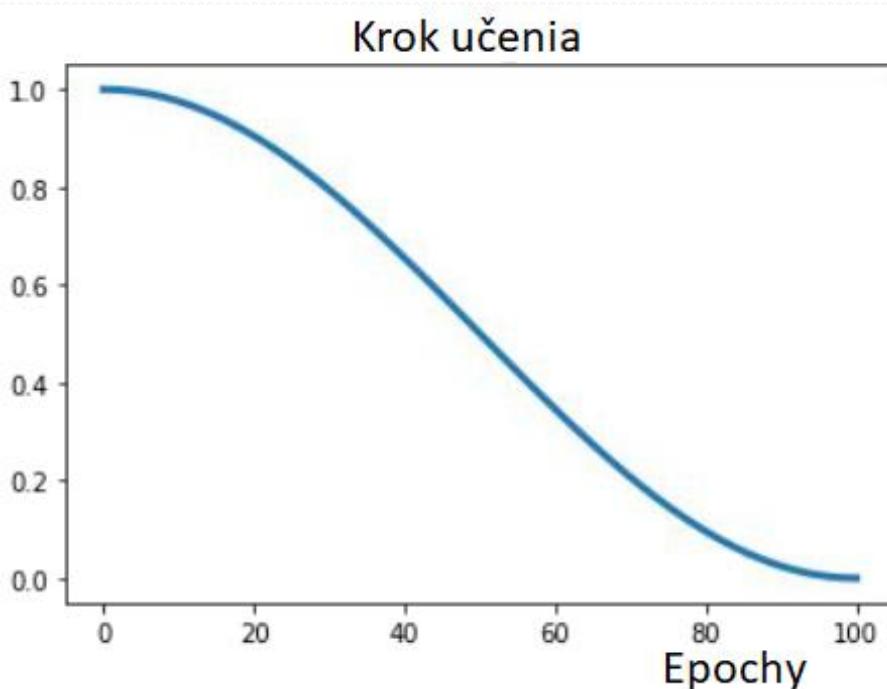
Od: Všetky! Začnite s vysokým krokom a postupne ho znižujte

Znižovanie kroku učenia



Krokové: Redukcia kroku učenia v pár fixovaných bodoch, napr. násobením 0,1 po epoche 30, 60 a 90

Znižovanie kroku učenia



Krokové: Redukcia kroku učenia v pár fixovaných bodoch, napr. násobením 0,1 po epoche 30,60 a 90

Kosínusové:

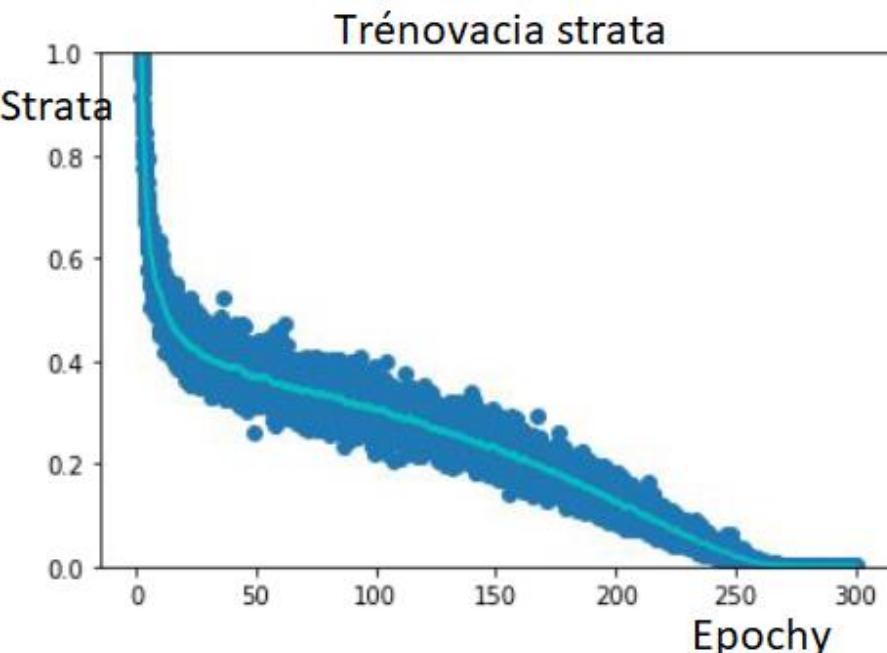
$$\alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(t\pi/T))$$

α_0 : Počiatočný krok učenia

α_t : Krok učenia v epoche t

T : Celkový počet epoch

Znižovanie kroku učenia



Krokové: Redukcia kroku učenia v pár fixovaných bodoch, napr. násobením 0,1 po epoche 30, 60 a 90

Kosínusové:

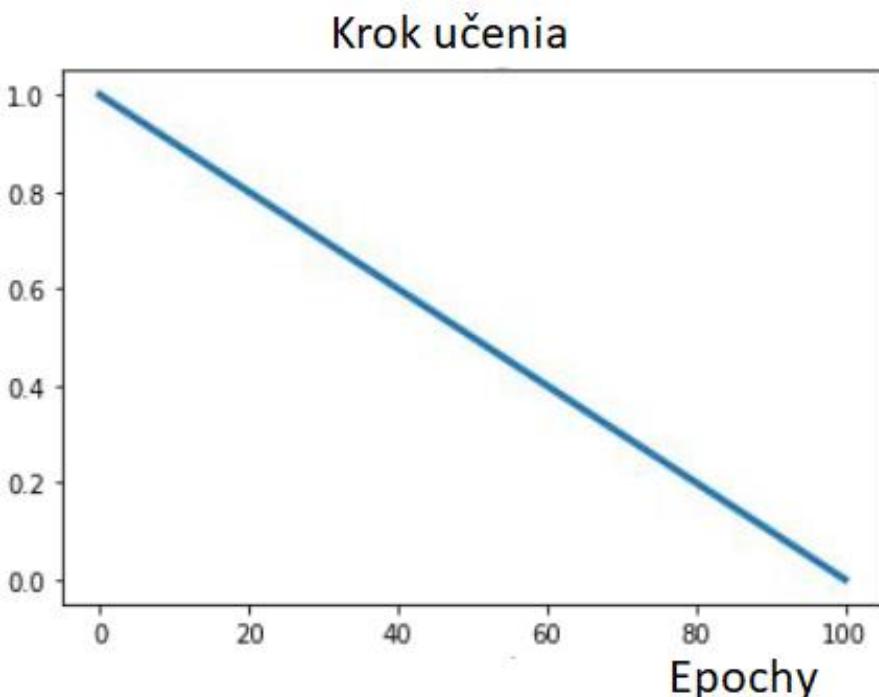
$$\alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(t\pi/T))$$

α_0 : Počiatočný krok učenia

α_t : Krok učenia v epoche t

T : Celkový počet epoch

Znižovanie kroku učenia



- α_0 : Počiatočný krok učenia
- α_t : Krok učenia v epoche t
- T : Celkový počet epoch

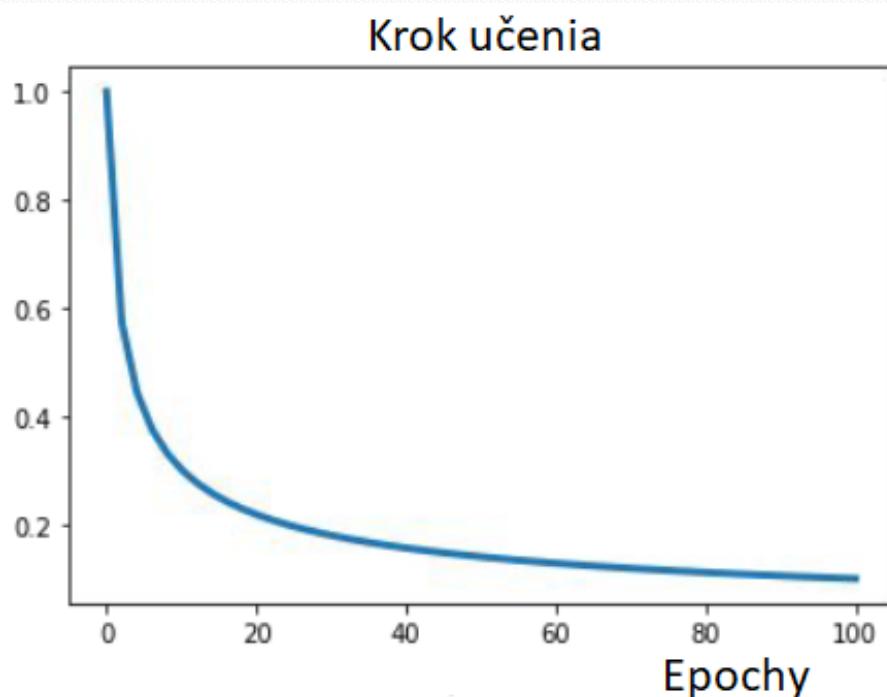
Krokové: Redukcia kroku učenia v pár fixovaných bodoch, napr. násobením 0,1 po epoche 30, 60 a 90

Kosínusové:

$$\alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(t\pi/T))$$

Lineárne: $\alpha_t = \alpha_0(1 - t/T)$

Znižovanie kroku učenia



α_0 : Počiatočný krok učenia

α_t : Krok učenia v epoche t

T : Celkový počet epoch

Krokové: Redukcia kroku učenia v pár fixovaných bodoch, napr. násobením 0,1 po epoche 30,60 a 90

Kosínusové:

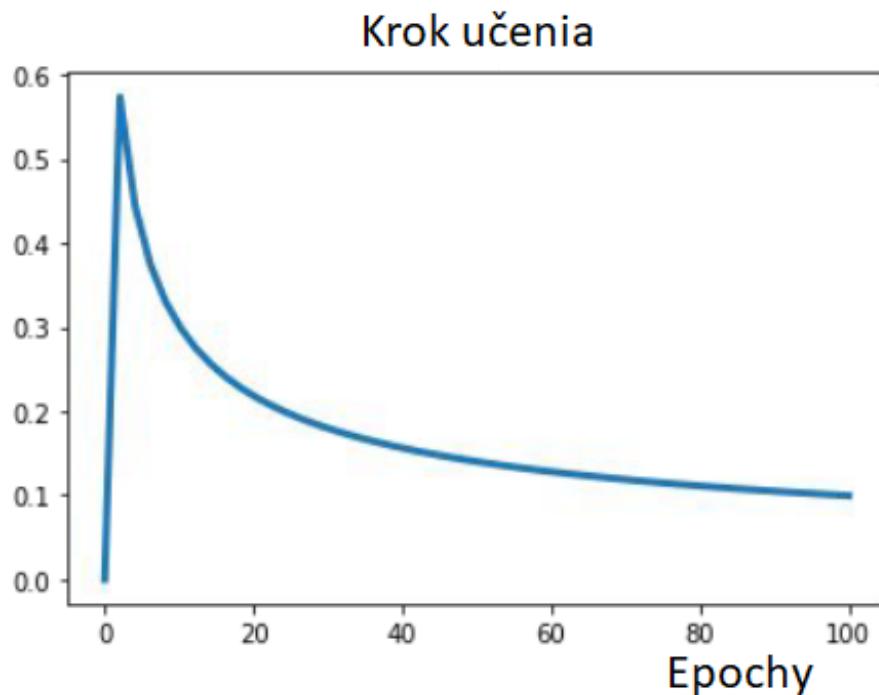
$$\alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(t\pi/T))$$

Lineárne: $\alpha_t = \alpha_0(1 - t/T)$

**Inverznou
odmocnicou:**

$$\alpha_t = \alpha_0 / \sqrt{t}$$

Znižovanie kroku učenia

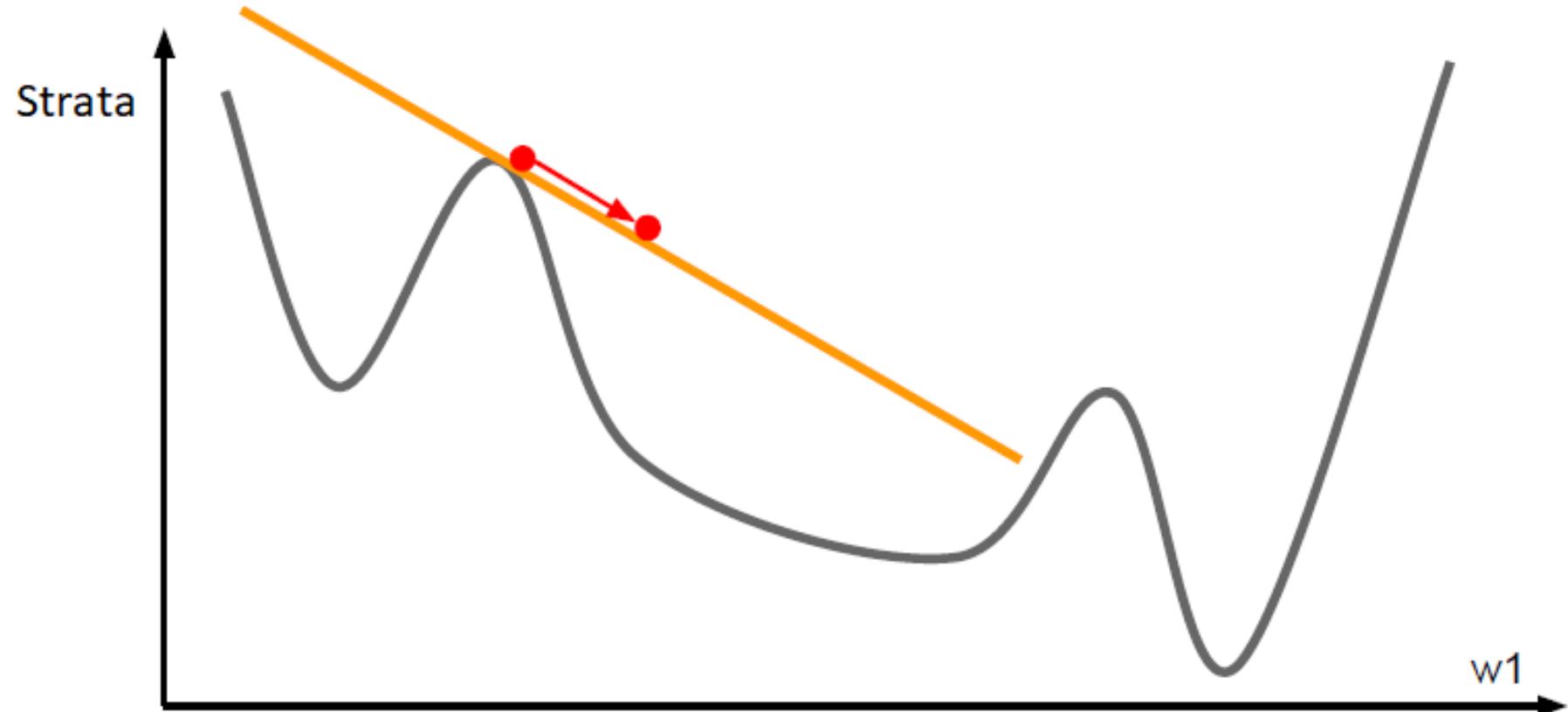


Vysoký počiatočný krok učenia môže spôsobiť explóziu vo výške straty; lineárne zvyšovanie kroku z 0 až po cca 5.000 iterácií tomu môže zabrániť

Empirické pravidlo: Ak zvyšujete veľkosť dávky N krát, tak škálujte aj počiatočný krok učenia faktorom N

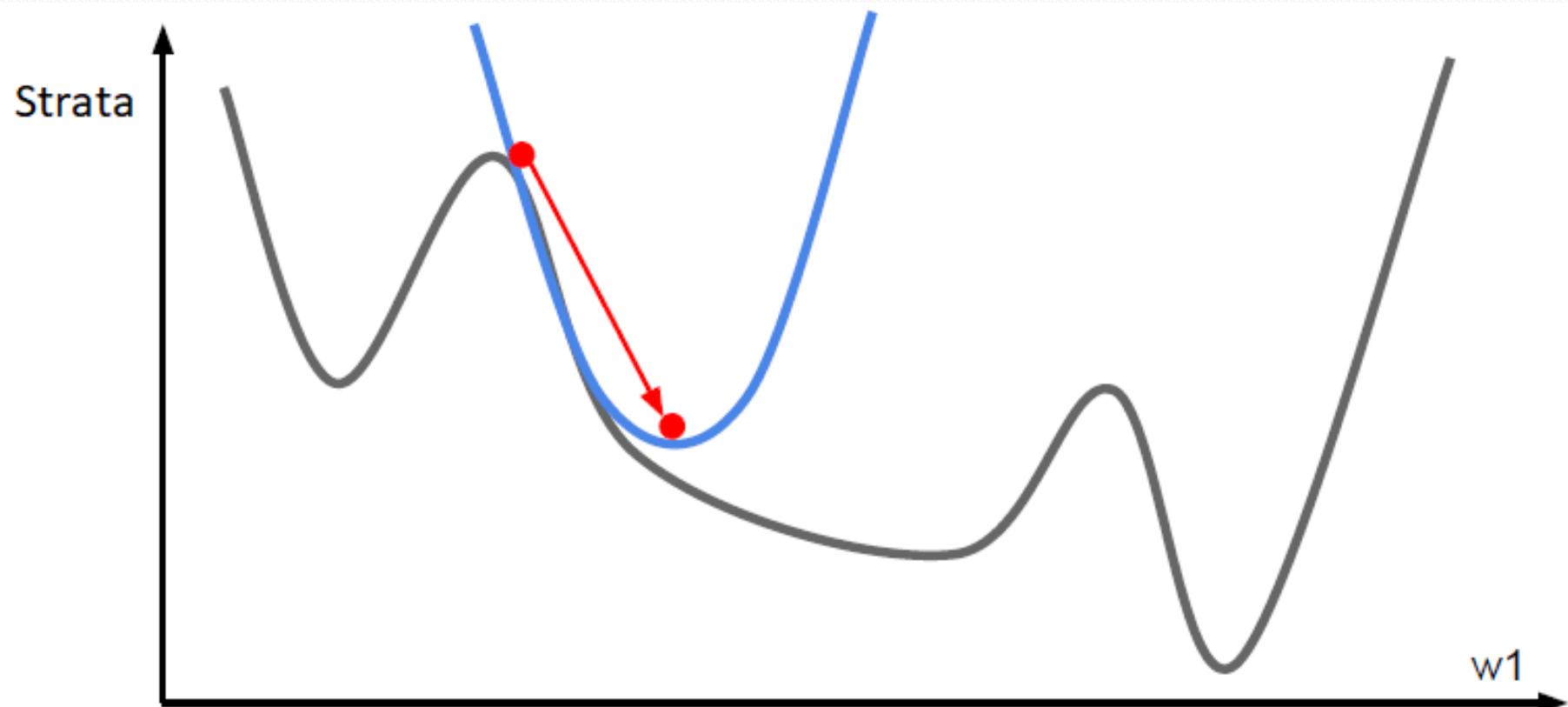
Optimalizácia prvého rádu

- 1) Použi gradient a vytvor lineárnu approximáciu
- 2) Urob krok na minimalizáciu approximácie



Optimalizácia druhého rádu

- 1) Použi gradient a Hessián na kvadratickú aproximáciu
- 2) Urob krok smerom k minimu aproximácie



Optimalizácia druhého rádu

- Taylorov rozvoj druhého rádu

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Pri riešení kritického bodu, dostaneme Newtonovu aktualizáciu parametrov:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Ot: Prečo je to zlé pre hlboké učenie?

Optimalizácia druhého rádu

- Taylorov rozvoj druhého rádu

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Pri riešení kritického bodu, dostaneme Newtonovu aktualizáciu parametrov:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Hessián má $O(N^2)$ prvkov
Invertovanie vezme $O(N^3)$
 N = až do stovky miliónov

Ot: Prečo je to zlé pre hlboké učenie?

Optimalizácia druhého rádu

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

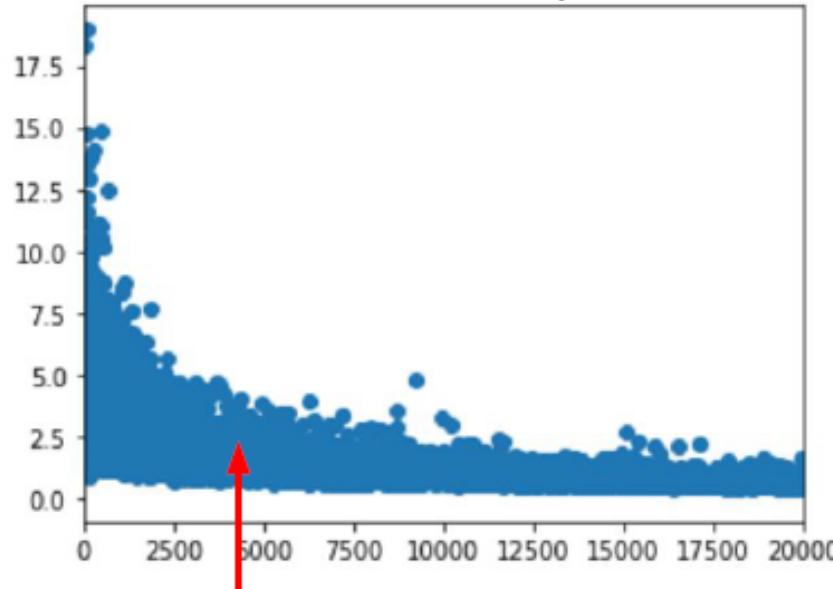
- Kvázi-newtonovské metódy (najpopulárnejšia je **BFGS**): *namiesto invertovania Hessiánu ($O(N^3)$), approximuj v čase inverzný Hessián aktualizáciami stupňa 1 ($O(N^2)$ každá)*
- **L-BFGS** (s obmedzenou pamäťou)
Nevytvára/neukladá celý inverzný Hessián

Praktické rady

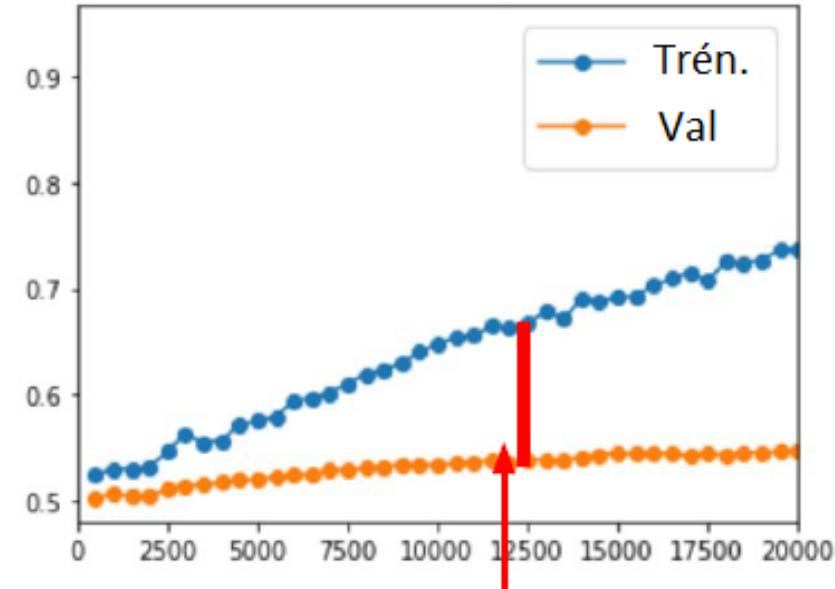
- **Adam** je veľmi dobrá prvotná voľba v mnohých prípadoch; funguje OK dokonca aj pri konštantnom kroku učenia
- **SGD+Momentum** môže prekonáť Adama, ale môže vyžadovať viac doladovania kroku učenia a schémy znižovania (použite kosínusovú schému, má veľmi málo hyperparametrov!)
- Ak si môžete dovoliť aktualizáciu na celej dávke, skúste **L-BFGS** (obmedzte všetky zdroje šumu)

Čo po trénovacej chybe?

Trénovacia chyba



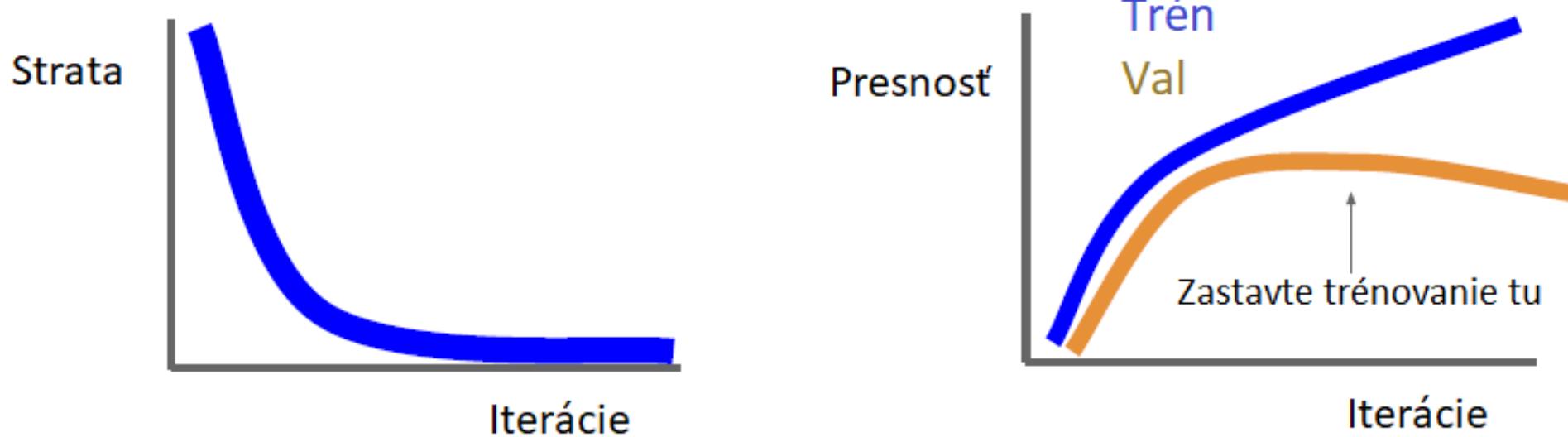
Presnosť



Lepšie optimalizačné algoritmy znížia trénovaciu chybu

Ale nás naozaj zaujíma chyba na nových dátach – ako znížiť ten rozdiel?

Treba vždy včas ukončiť



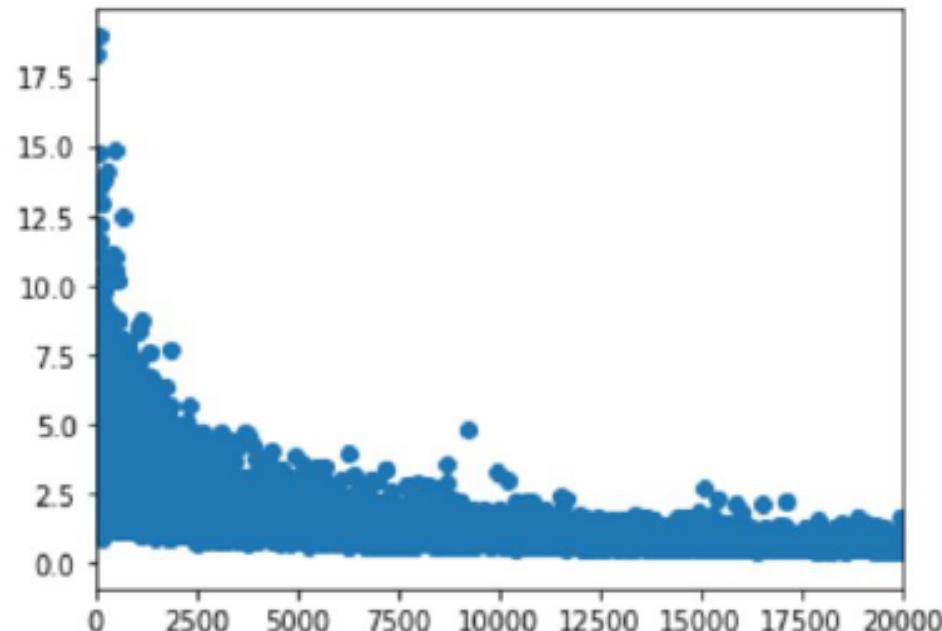
- Ukončite trénovanie modelu, keď sa presnosť na validačnej množine začne znižovať.
- Alebo trénujte dlhšie, ale vždy si pamätajte parametre modelu, ktorý mal na validačnej množine najlepšie výsledky

Skupiny modelov

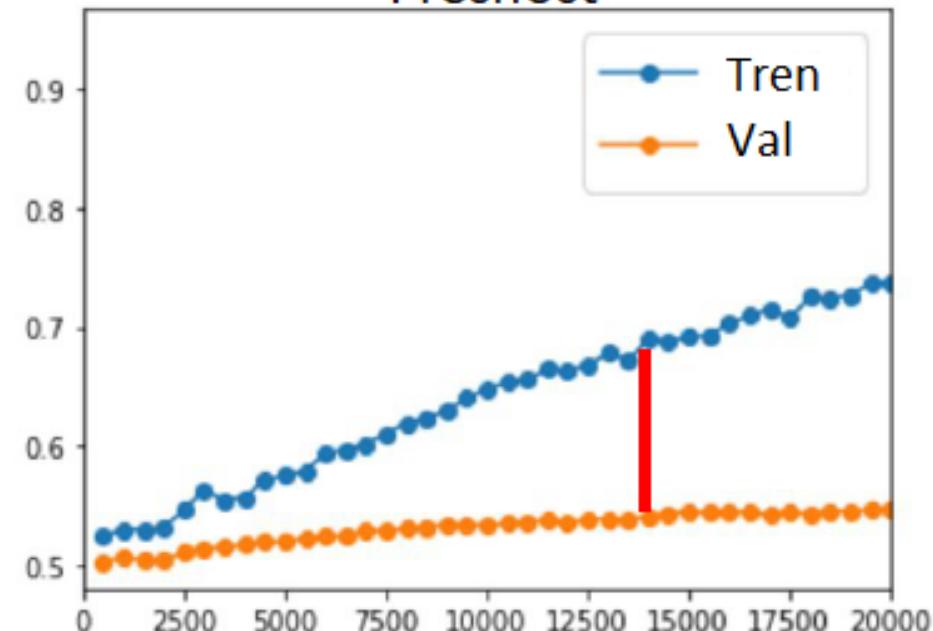
- Iná cesta môže byť trénovať viacero nezávislých modelov
- V čase testovania spriemerujte ich výsledky (vypočítajte priemery z predpovedaných rozdelení pravdepodobnosti a potom vyberte maximum)
- To umožní získať 2%-ný extra výkon, čo nie je veľa, ale môže to pomôcť

Ako zlepšiť výkon jedného modelu?

Trénovacia strata



Presnosť



Regularizácia

Regularizácia: pridať term k strate

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Bežne sa používa:

L2 regularizácia

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad (\text{Zníženie váh})$$

L1 regularizácia

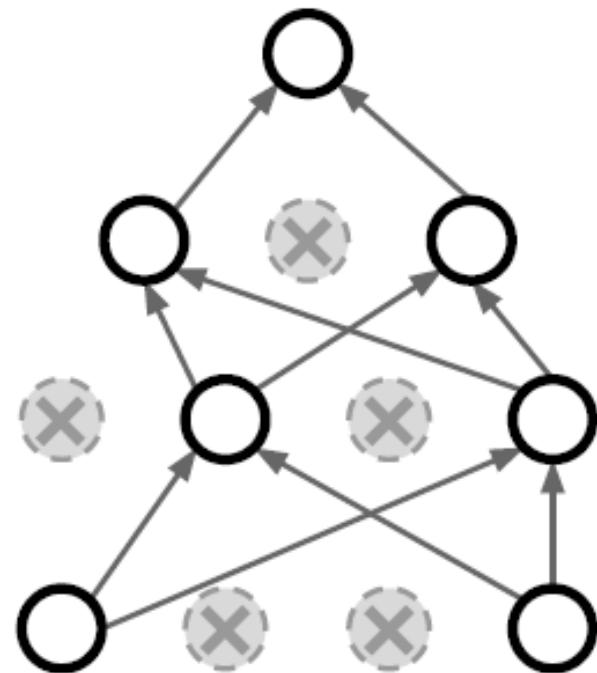
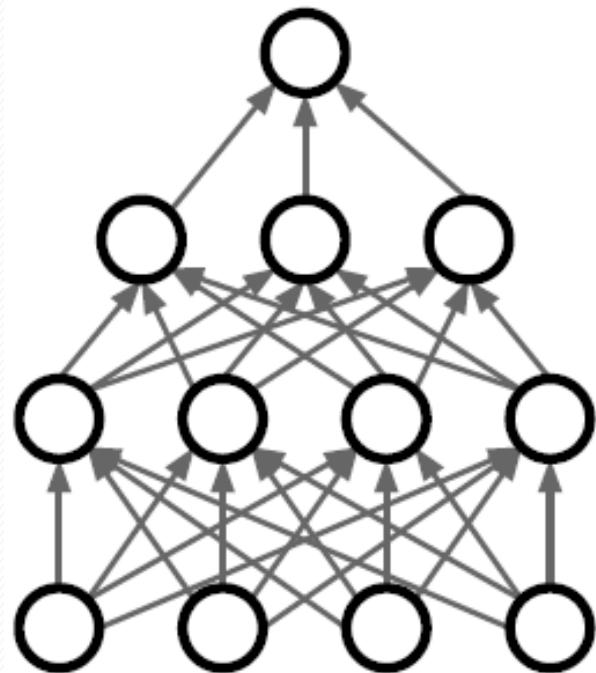
$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Elastická siet' (L1 + L2)

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Regularizácia: dropout

V každom doprednom chode, náhodne nastavte niektoré neuróny na nulu; pravdepodobnosť dropout je hyperparameter; obvyklá je 0,5



Regularizácia: dropout

Ako to vôbec môže byť dobrý nápad?

Núti to siet' vytvoriť redundantnú reprezentáciu
Je to prevencia pred prispôsobením si príznakov



Regularizácia: dropout

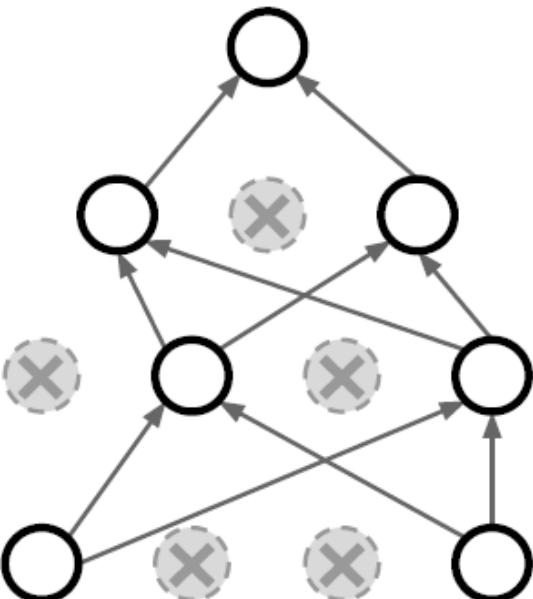
Ako to vôbec môže byť dobrý nápad?

Iná interpretácia:

Dropout je trénovanie veľkej skupiny modelov (ktoré zdieľajú parametre)

Každá binárna maska je jeden model

Každá FC vrstva so 4.096 neurónmi má $2^{4096} \sim 10^{1233}$ možných masiek!
Vo vesmíre je iba $\sim 10^{82}$ atómov



Dropout: v čase testovania

- Dropout robí náš výstup náhodným!

$$y = f_W(x, z)$$

Výstup (trieda) Vstup (obraz)
Náhodná maska

- Chceme „spriemerovať“ túto náhodnosť v čase testovania

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

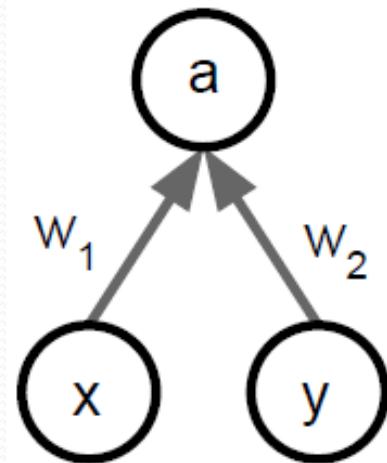
- Ale tento integrál vyzerá zložito ...

Dropout: v čase testovania

- Chceme approximovať ten integrál

$$y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz$$

Uvažujme jeden neurón

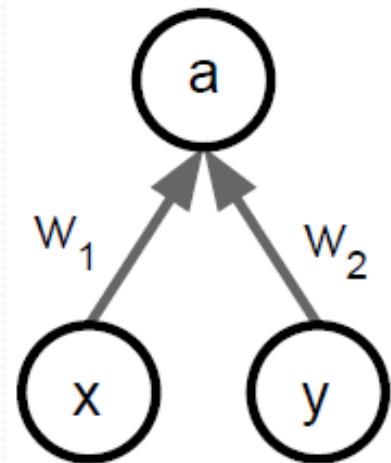


Dropout: v čase testovania

- Chceme aproximovať ten integrál

$$y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz$$

Uvažujme jeden neurón



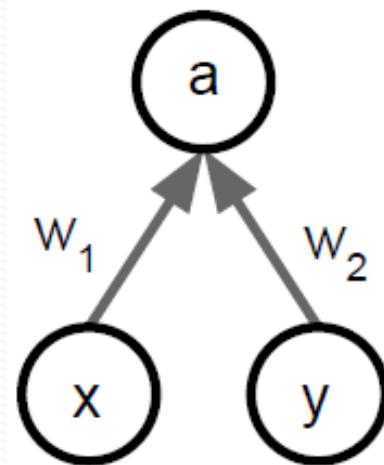
V čase testovania máme: $E[a] = w_1x + w_2y$

Dropout: v čase testovania

- Chceme aproximovať ten integrál

$$y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz$$

Uvažujme jeden neurón



V čase testovania máme: $E[a] = w_1x + w_2y$

Počas trénovania

máme:

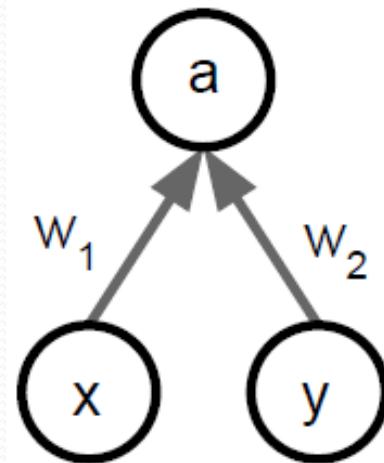
$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

Dropout: v čase testovania

- Chceme aproximovať ten integrál

$$y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz$$

Uvažujme jeden neurón



V čase testovania máme: $E[a] = w_1x + w_2y$

Počas trénovania

máme:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

Počas testovania, násobte
pravdepodobnosťou dropoutu

Dropout: v čase testovania

```
def predict(X):  
    # ensembled forward pass  
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations  
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations  
    out = np.dot(W3, H2) + b3
```

- V čase testovania sú všetky neuróny vždy aktívne, preto musíme škálovať aktivácie, aby pre každý neurón platilo:

výstup pri testovaní = očakávaný výstup pri trénovaní

Dropout: sumár

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

Dropout v
doprednom
chode

Škálovanie
pri testovaní

Obvyklejší: „invertovaný“ dropout

```
p = 0.5 # probability of keeping a unit active, higher = less dropout

def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
```

Delim p

Čas
testovania
je
nezmenený!

Regularizácia: spoločný vzor

- **Trénovanie**

- pridáva nejaký druh náhodnosti

$$y = f_W(x, z)$$

- **Testovanie**

- priemeruje náhodnosť (niekedy approximuje)

$$y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz$$

Regularizácia: spoločný vzor

- **Trénovanie**

- pridáva nejaký druh náhodnosti

$$y = f_W(x, z)$$

- **Testovanie**

- priemeruje náhodnosť (niekedy approximuje)

$$y = f(x) = E_z [f(x, z)] = \int p(z)f(x, z)dz$$

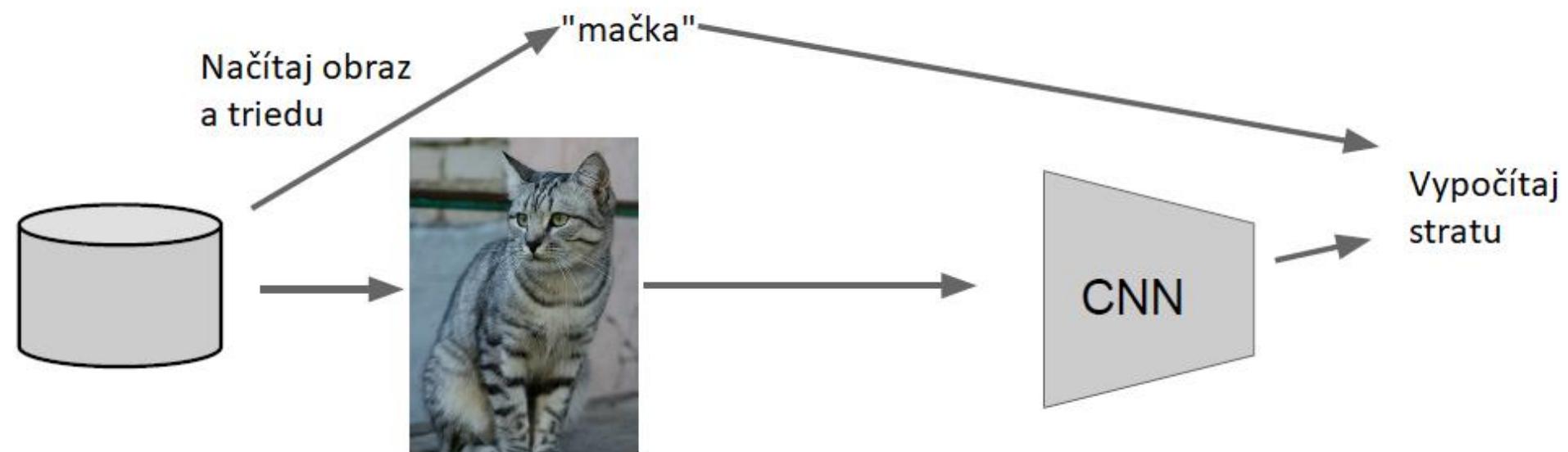
Príklad: BN

Trénovanie:

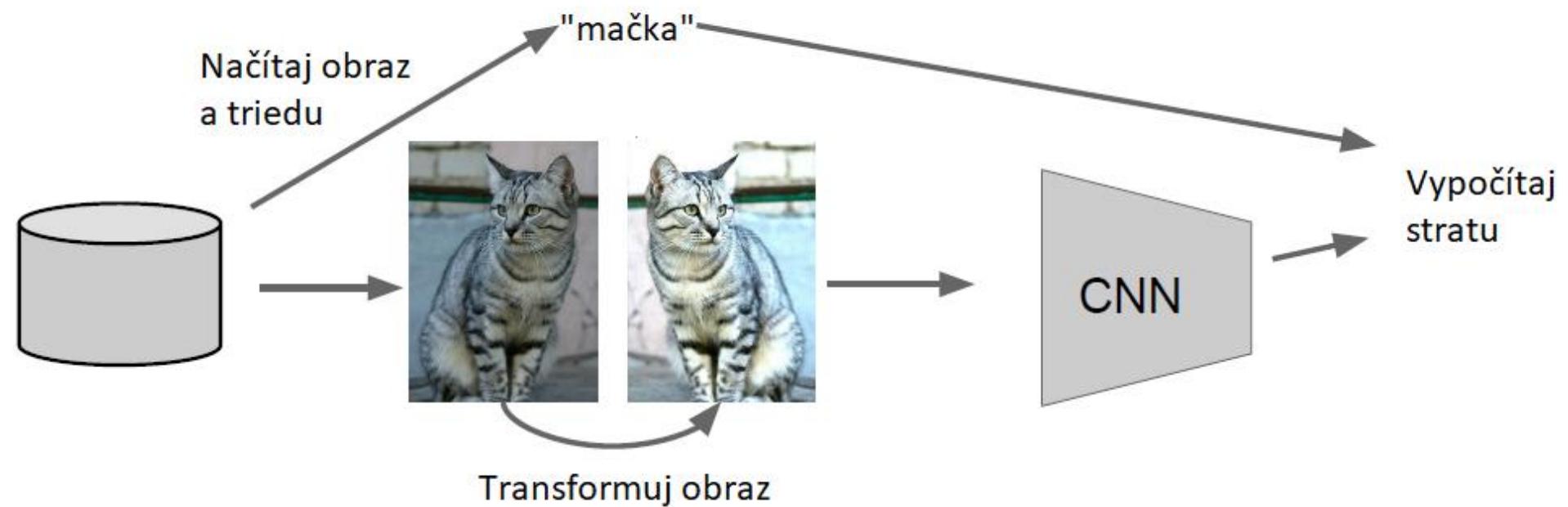
Normalizuj pomocou štatistik z náhodných minidávok

Testovanie: Použi fixné globálne štatistiky na normalizáciu

Regularizácia: Augmentácia dát



Regularizácia: Augmentácia dát



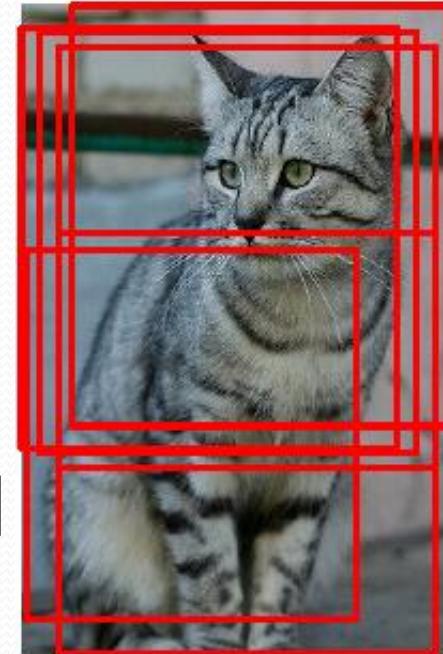
Augmentácia dát

- **Horizontálne otočenia**



Augmentácia dát

- Náhodné výseky a škály
- Trénovanie (ResNet):
 1. Vyber náhodné L v rozsahu [256,480]
 2. Zmeň veľkosť obrazu, kratšia strana je L
 3. Preškáluj náhodne časť obrazu [224 x 224]

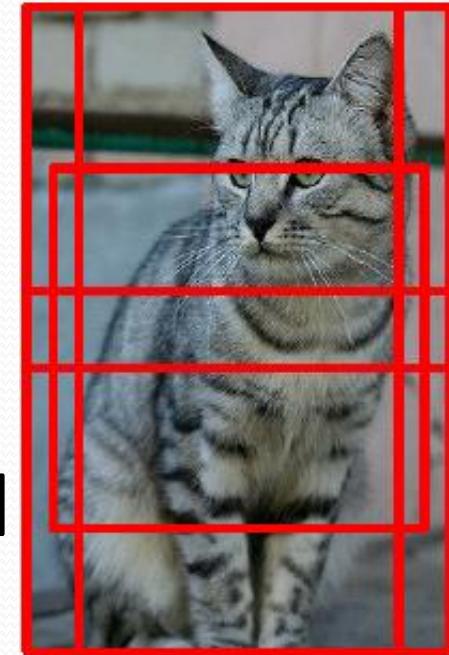


Augmentácia dát

- **Náhodné výseky a škály**

- **Trénovanie (ResNet):**

1. Vyber náhodné L v rozsahu [256,480]
2. Zmeň veľkosť obrazu, kratšia strana je L
3. Preškáluj náhodne časť obrazu [224 x 224]



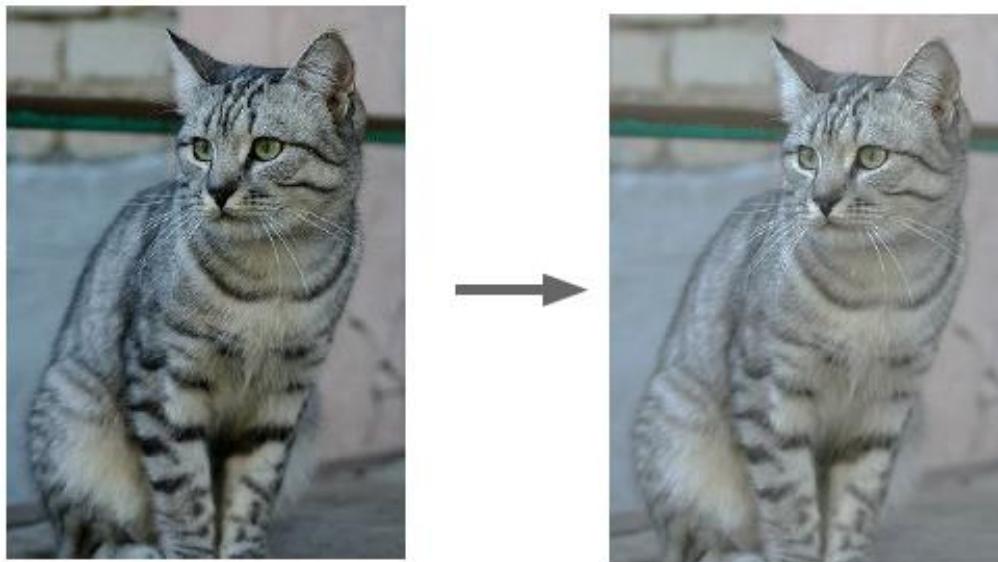
- **Testovanie (ResNet):**

- Spriemeruje fixnú množinu výsekov

1. Zmeň veľkosť obrazu na 5 škál: {224, 256, 384, 460, 640}
2. Pre každú veľkosť použi 10 224 x 224 výsekov: 4 rohy + stred + otočenia

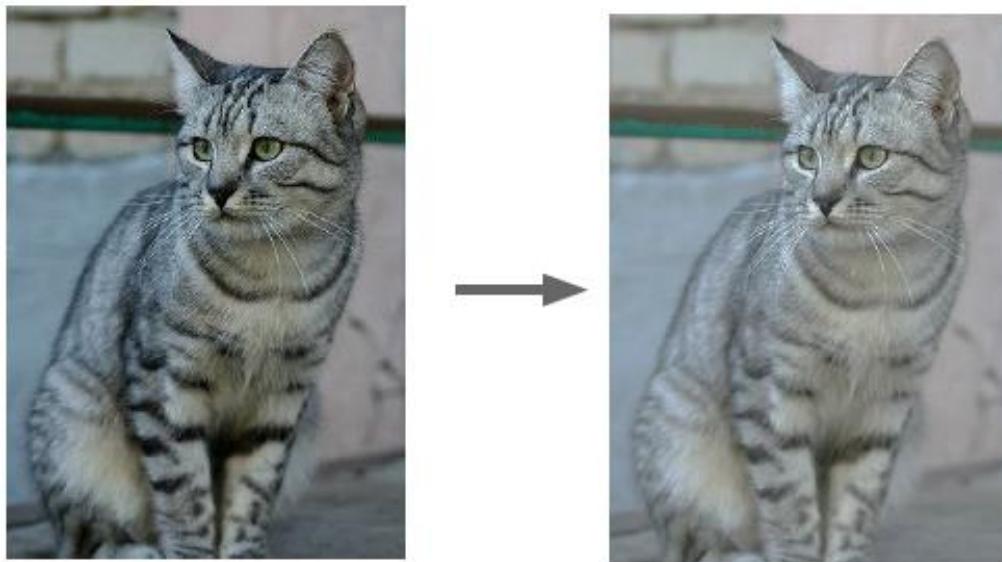
Augmentácia dát

- Vibrácia farieb
- Jednoducho: použite náhodný kontrast a jas



Augmentácia dát

- Vibrácia farieb
- Jednoducho: použite náhodný kontrast a jas



- Trochu zložitejšie:
 1. Požite PCA na všetky [R,G,B] pixle v trénovacej množine
 2. Navzorkujte „posun farby“ pozdĺž smerov hlavných komponentov
 3. Pridajte posun ku všetkým pixlom trénovacieho obrazu

Augmentácia dát

- Budťte kreatívni pri Vašom probléme
- Náhodná kombinácia:
 - Translácie
 - Rotácie
 - Strečingu
 - Šmýkania
 - Distorcia šošoviek, ... (môžete sa vyblázniť)

Automatická augmentácia dát

- Článok z roku 2019

	Original	Sub-policy 1	Sub-policy 2	Sub-policy 3	Sub-policy 4	Sub-policy 5
Batch 1						
Batch 2						
Batch 3						
	ShearX, 0.9, 7 Invert, 0.2, 3	ShearY, 0.7, 6 Solarize, 0.4, 8	ShearX, 0.9, 4 AutoContrast, 0.8, 3	Invert, 0.9, 3 Equalize, 0.6, 3	ShearY, 0.8, 5 AutoContrast, 0.7, 3	

Regularizácia: spoločný vzor

- **Trénovanie:** Pridaj náhodný šum
- **Testovanie:** Zníž šum na minimum
- **Príklady:**

Dropout

BN

Augmentácia dát

Regularizácia: DropConnect

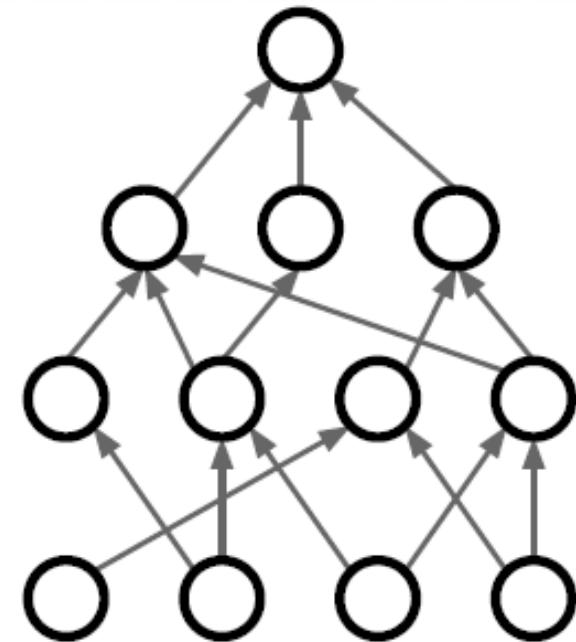
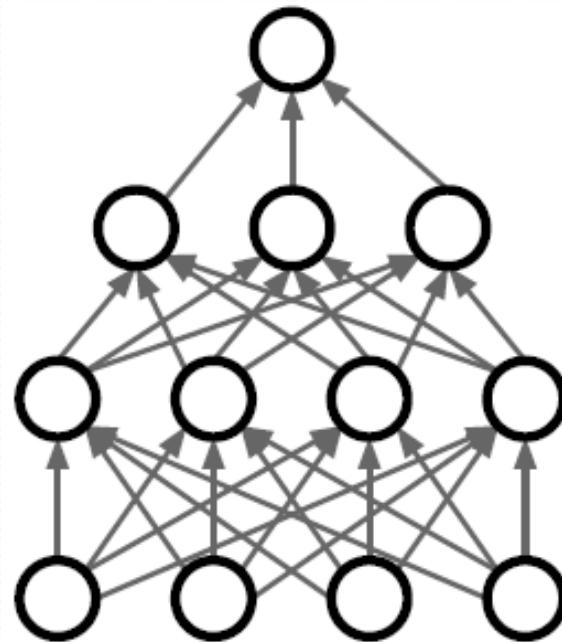
- **Trénovanie:** Zruš spojenia medzi neurónmi ($w = 0$)
- **Testovanie:** Použi všetky spojenia
- **Príklady:**

Dropout

BN

Augmentácia dát

DropConnect



Regularizácia: čiastkový pooling

- **Trénovanie:** Použi náhodné poolovacie oblasti
- **Testovanie:** Priemeruj predikcie z viacerých oblastí
- **Príklady:**

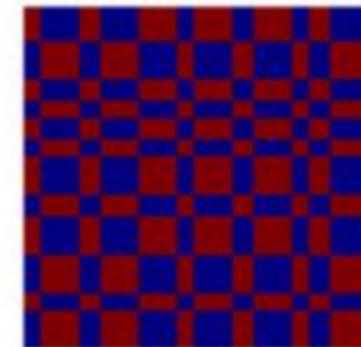
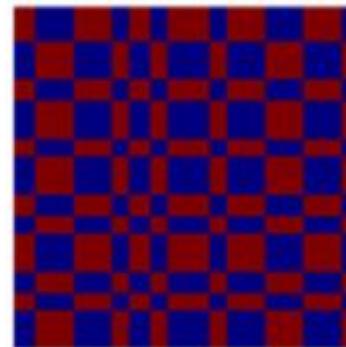
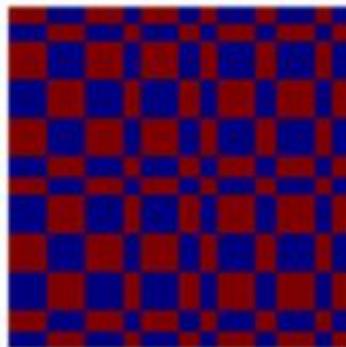
Dropout

BN

Augmentácia dát

DropConnect

Čiastkový max pooling



Regularizácia: stochastická híbka

- **Trénovanie:** Preskoč niektoré vrstvy v sieti
- **Testovanie:** Použi všetky vrstvy
- **Príklady:**

Dropout

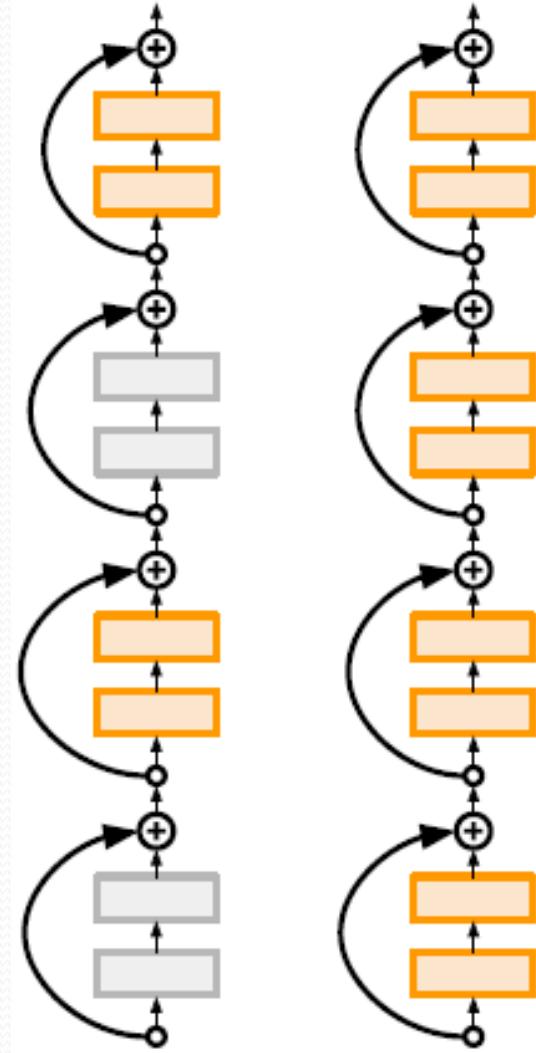
BN

Augmentácia dát

DropConnect

Čiastkový max pooling

Stochastická híbka



Regularizácia: výsek

- **Trénovanie:** Nastav náhodne oblasti na nulu
- **Testovanie:** Použi celý obraz
- **Príklady:**

Dropout

BN

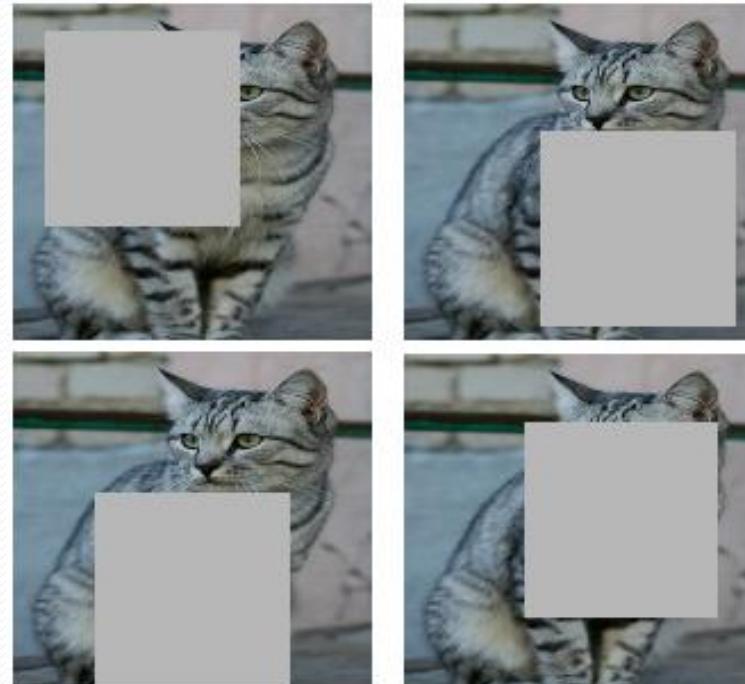
Augmentácia dát

DropConnect

Čiastkový max pooling

Stochastická hĺbka

Výsek (Cutout)



Funguje veľmi dobre pre malé databázy ako CIFAR10, menej pre veľké ako ImageNet

Regularizácia: zmiešanie

- **Trénovanie:** Na náhodnom zmiešaní obrazov
- **Testovanie:** Použi pôvodné obrazy
- **Príklady:**

Dropout



BN

Augmentácia dát



DropConnect

Čiastkový max pool



Stochastická hĺbka

Výsek (Cutout)

Zmiešanie (Mixup)



Výsledné označenie:
mačka 0,4
pes 0,6

Náhodne zmieša pixle
páru trénovacích obrazov
napr. 40% mačka, 60% pes

Regularizácia v praxi

- **Trénovanie:** Pridaj náhodný šum
- **Testovanie:** Zníž šum na minimum
- **Príklady:**

Dropout

- Zvažujte dropout pre veľké plne prepojené vrstvy

BN

Augmentácia dát

- Normalizácia dávky a augmentácia dát je skoro vždy dobrý nápad

DropConnect

- Vyskúšajte Výsek a Zmiešanie obrazov najmä pre malé datasety

Čiastkový max pooling

Stochastická hĺbka

Výsek (Cutout)

Zmiešanie (Mixup)