Neurónové siete pre počítačové videnie

Trénovanie neurónových sietí I

RNDr. Zuzana Černeková, PhD.
Ing. Viktor Kocur, PhD.

Štruktúra dnešnej prednášky

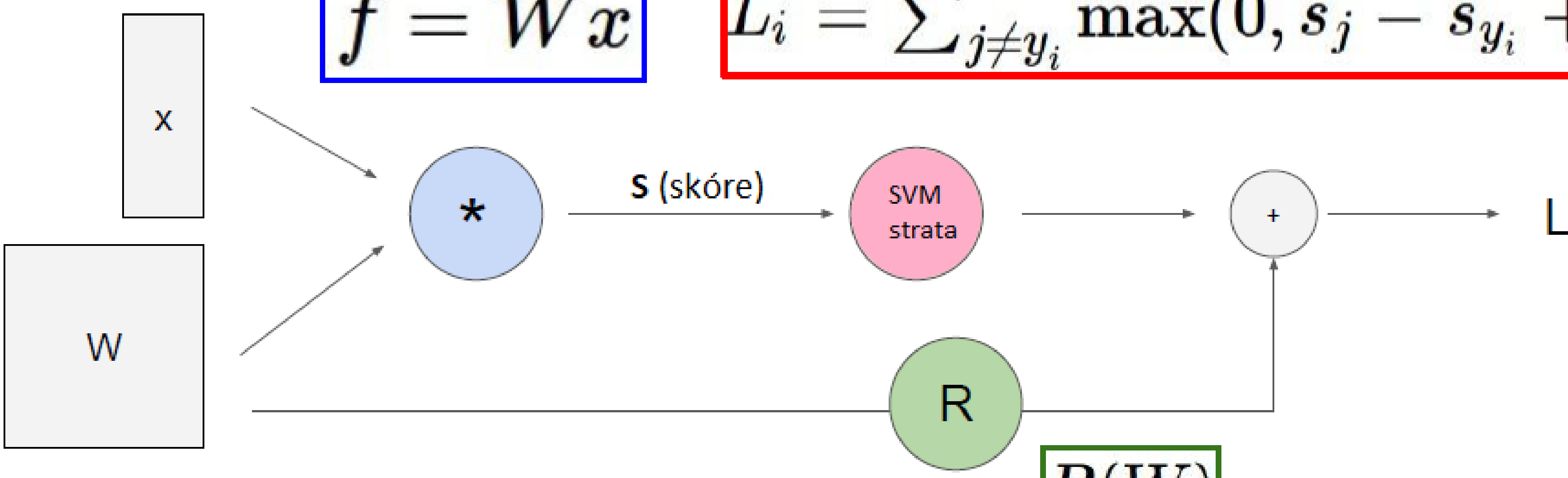
- Aktivačné funkcie
- Predspracovanie dát
- Inicializácia váh neurónovej siete
- Normalizácia dávky (batch normalisation)
  - pri trénovaní
  - pri testovaní

Opakovanie

- Výpočtové grafy a spätné šírenie chyby

$$f = Wx$$

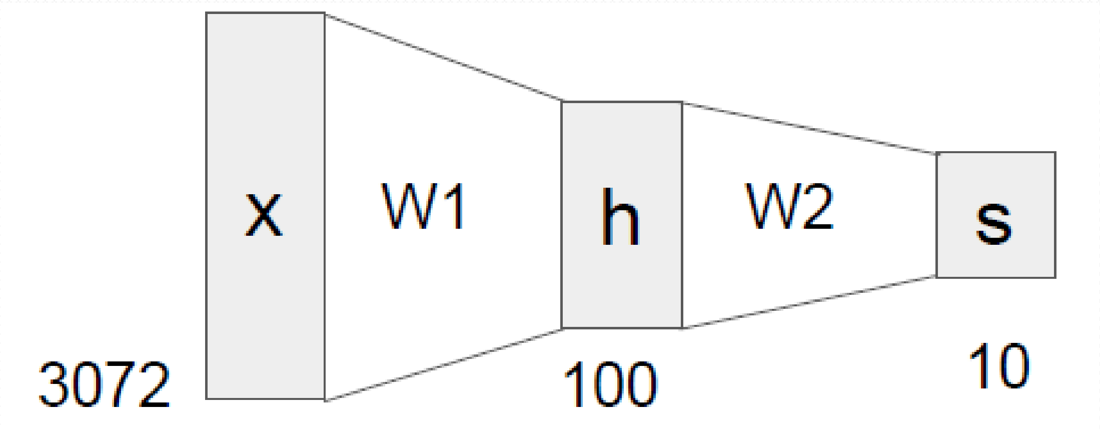
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

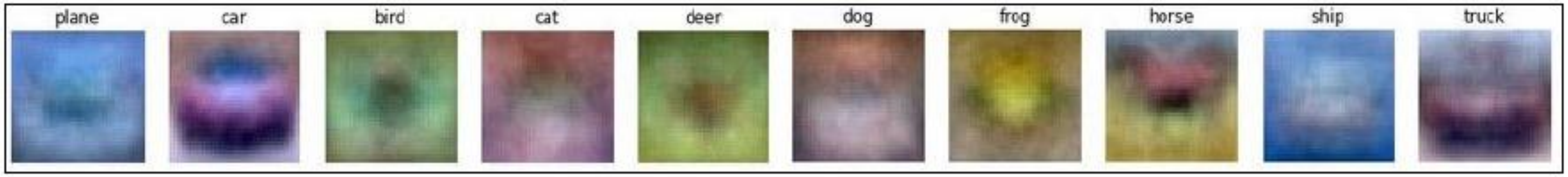


$$R(W)$$

Opakovanie 2

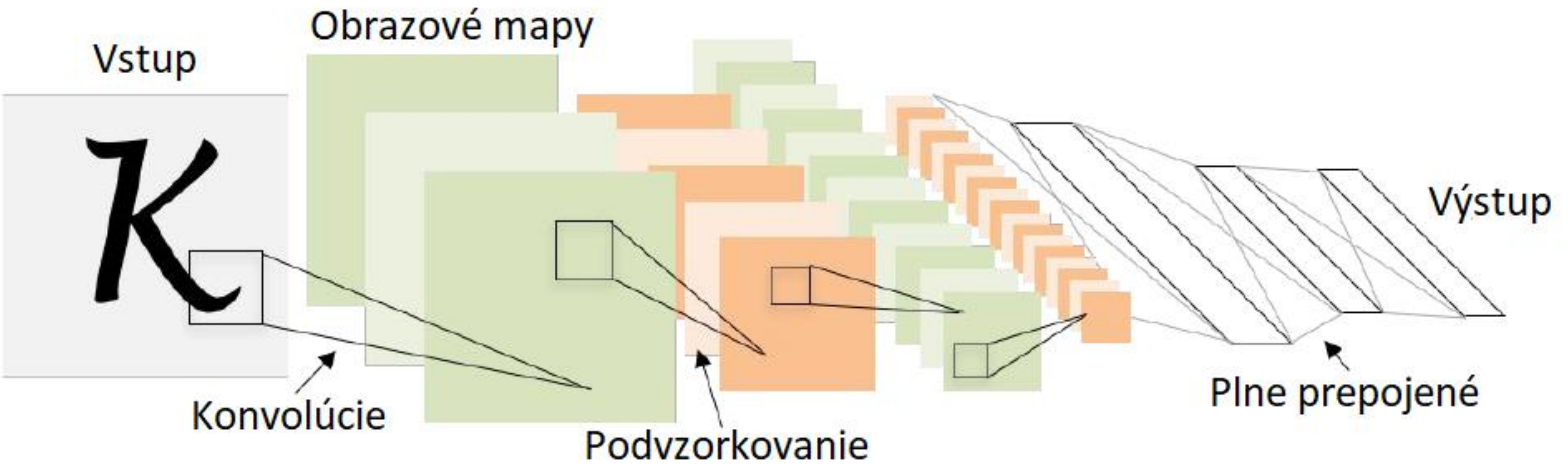
- Neurónové siete
- **Doteraz:** Lineárna fcia skóre $f = Wx$
- **Teraz:** 2-vrstvová NS $f = W_2 \max(0, W_1 x)$





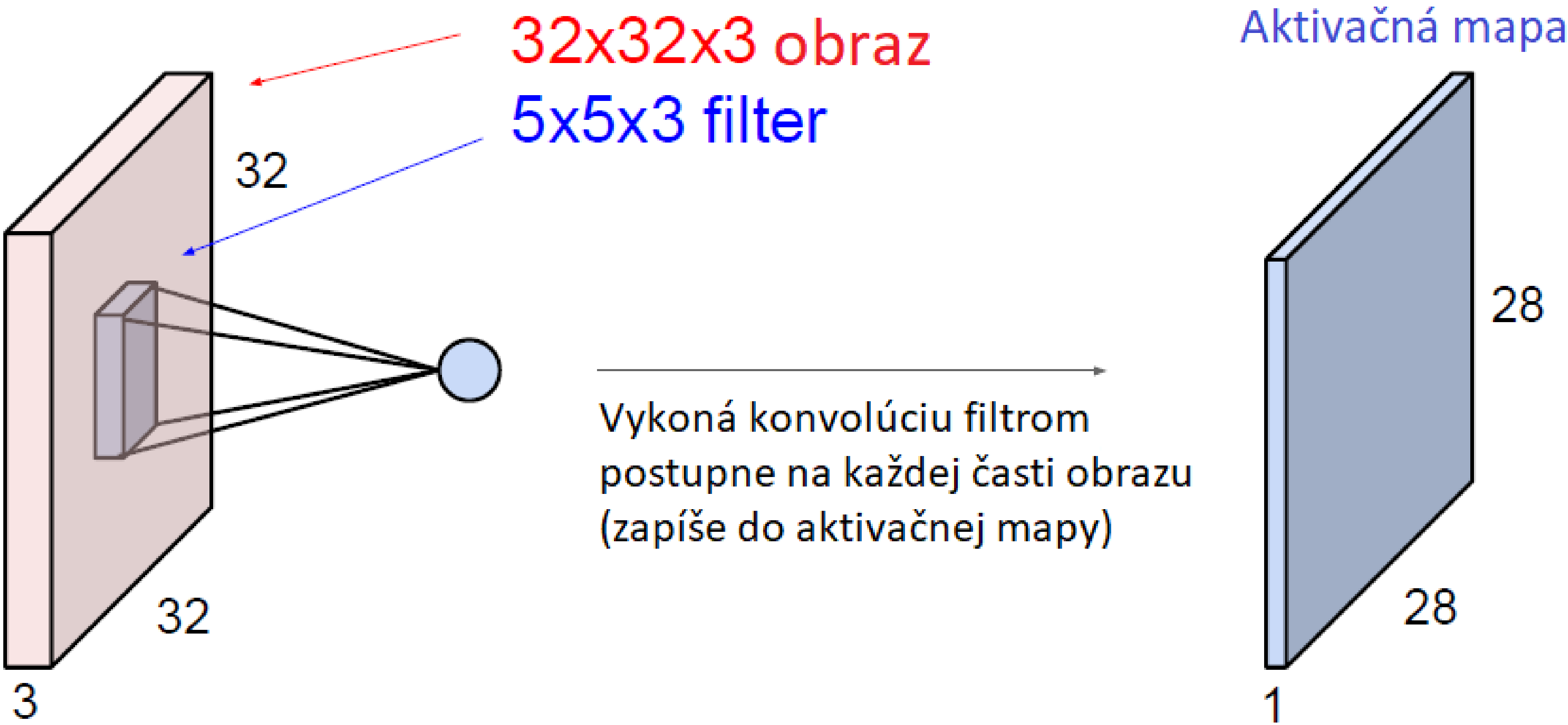
Opakovanie 3

- Konvolučné neurónové siete



Opakovanie 4

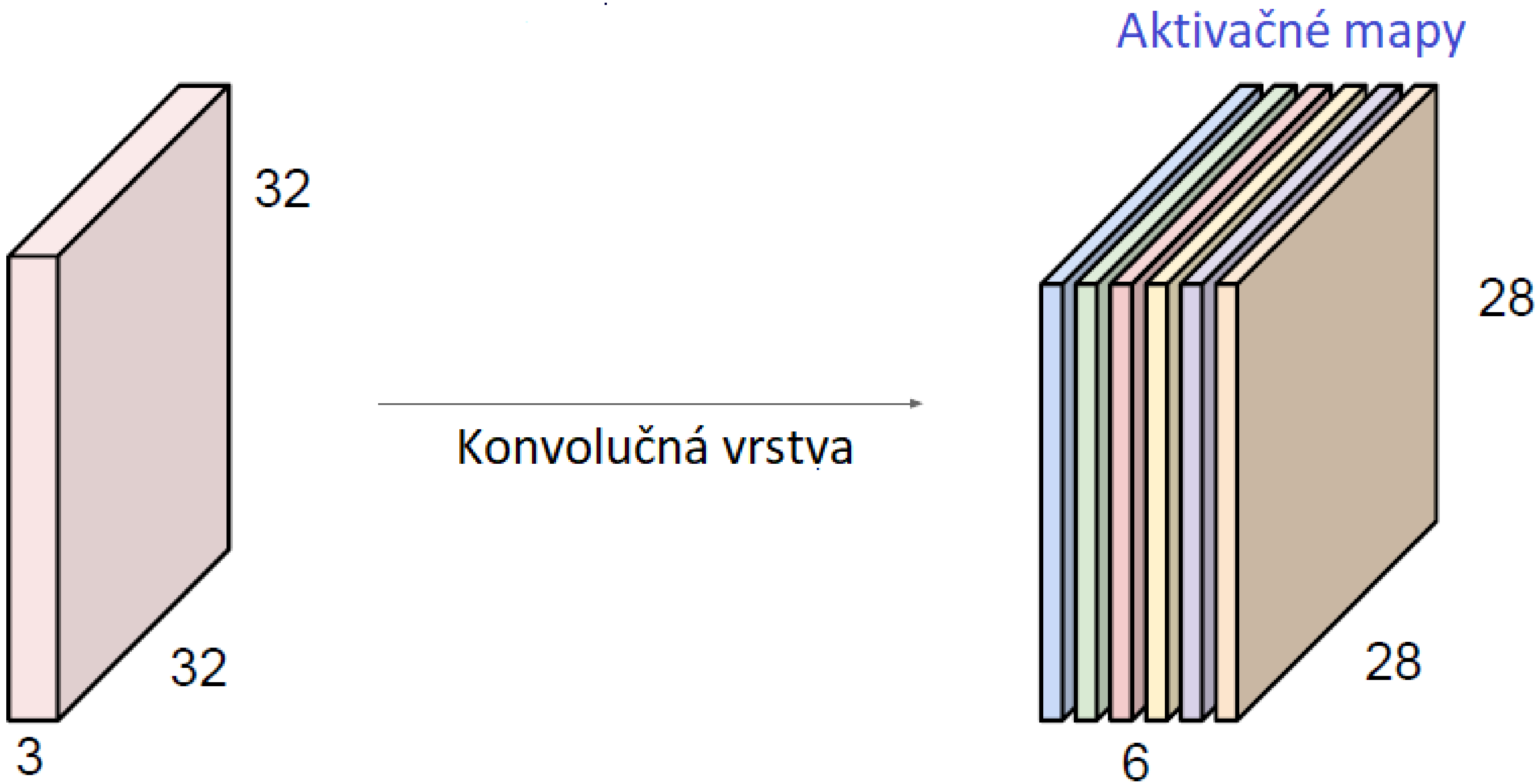
- Konvolučná vrstva



Opakovanie 5

- Konvolučná vrstva II

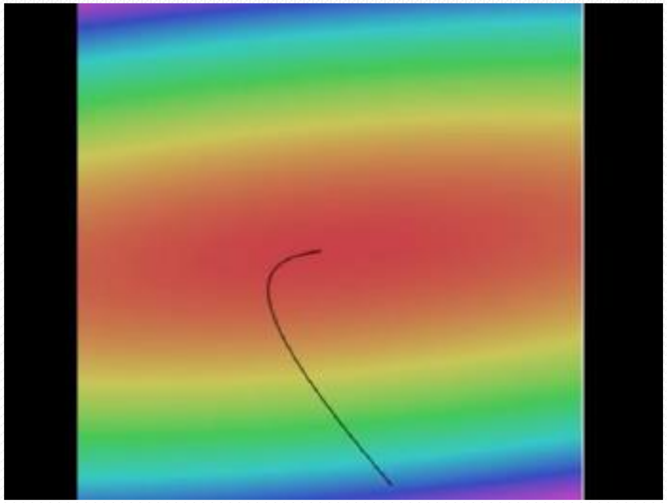
Ak máme 6 filtrov rozmeru 5x5, dostaneme 6 samostatných aktivačných máp



Pri tomto postupe dostaneme "nový obraz" rozmeru 28x2x6!

Opakovanie 6

- Naučiť sa parametre siete cez optimalizáciu

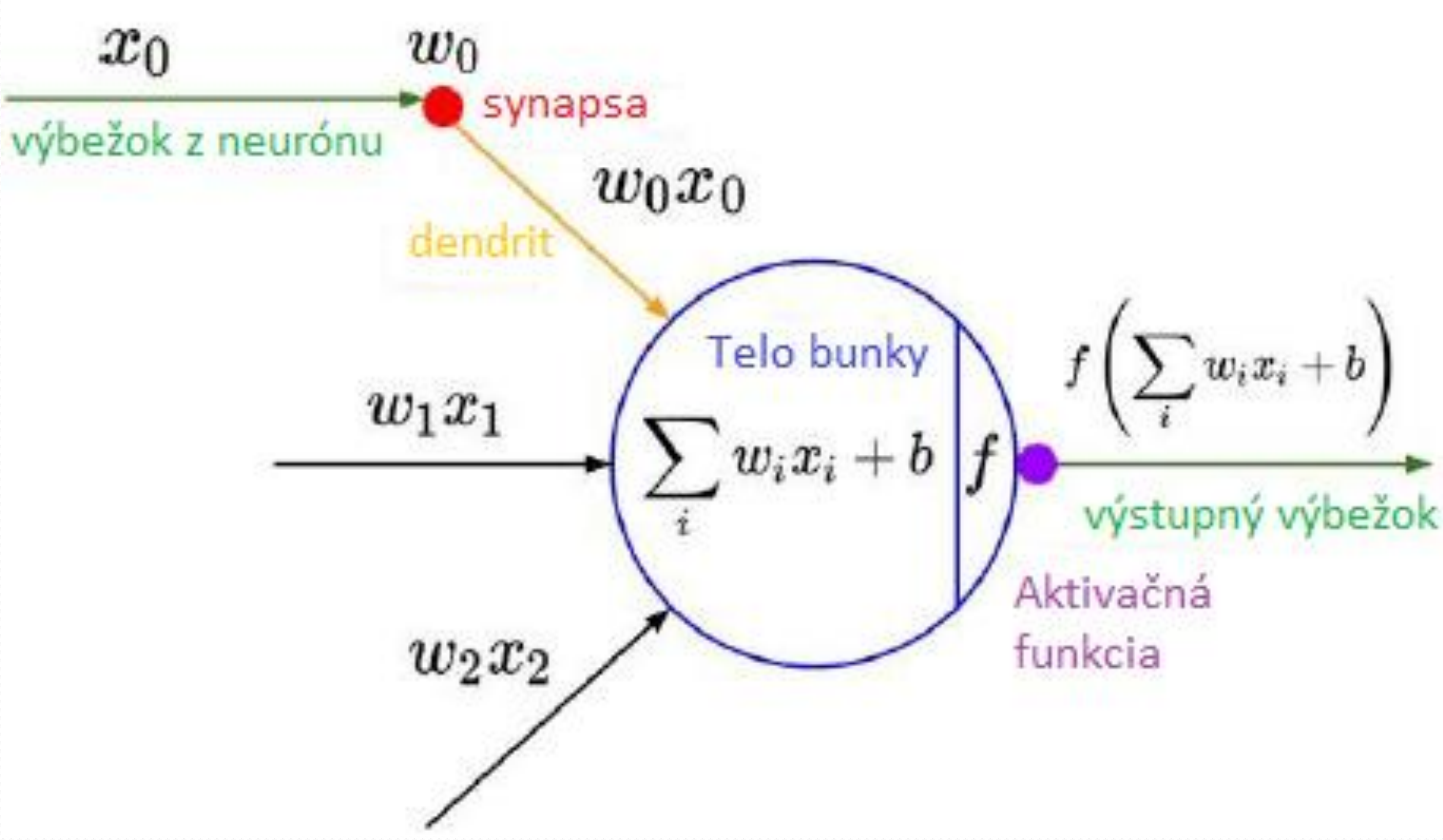


```
# Vanilla Gradient Descent

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

Trénovanie neurónových sietí

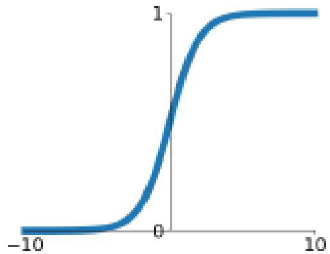
- Aktivačné funkcie



Aktivačné funkcie

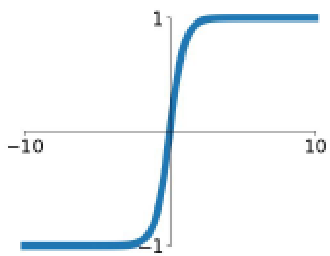
Sigmoid

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



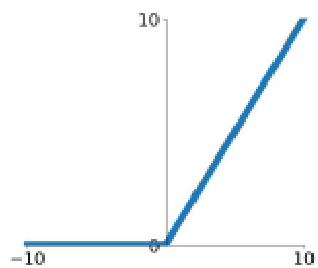
tanh

$\tanh(x)$



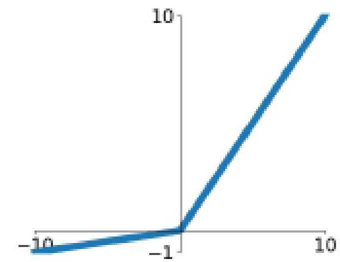
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

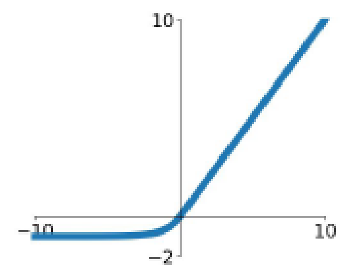


Maxout

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

ELU

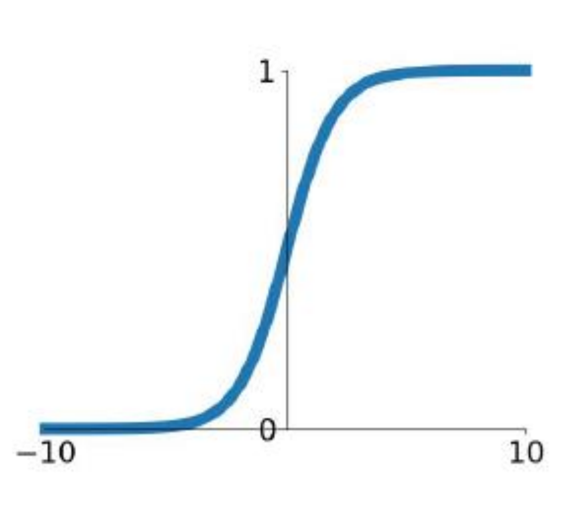
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Aktivačné funkcie

- **Sigmoid** stlačí čísla do intervalu [0,1]
- Historicky je populárny, pretože má peknú interpretáciu ako saturačná „miera odozvy“ neurónu

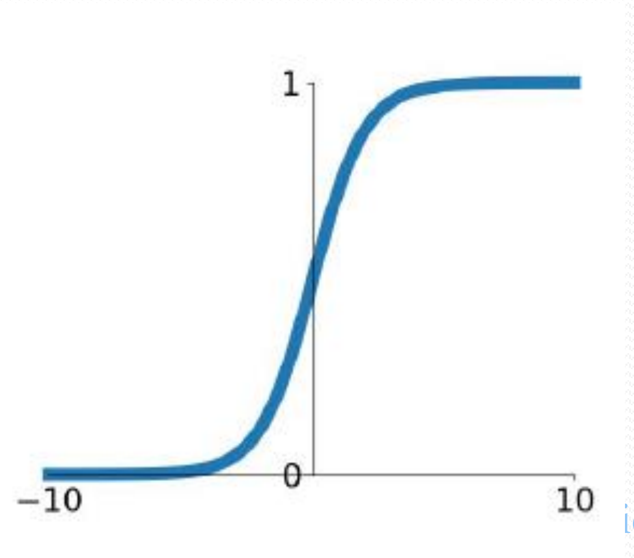
$$\sigma(x) = 1/(1 + e^{-x})$$



Aktivačné funkcie

- **Sigmoid** stlačí čísla do intervalu [0,1]
- Historicky je populárny, pre-tože má peknú interpretáciu ako saturačná „miera odozvy“ neurónu

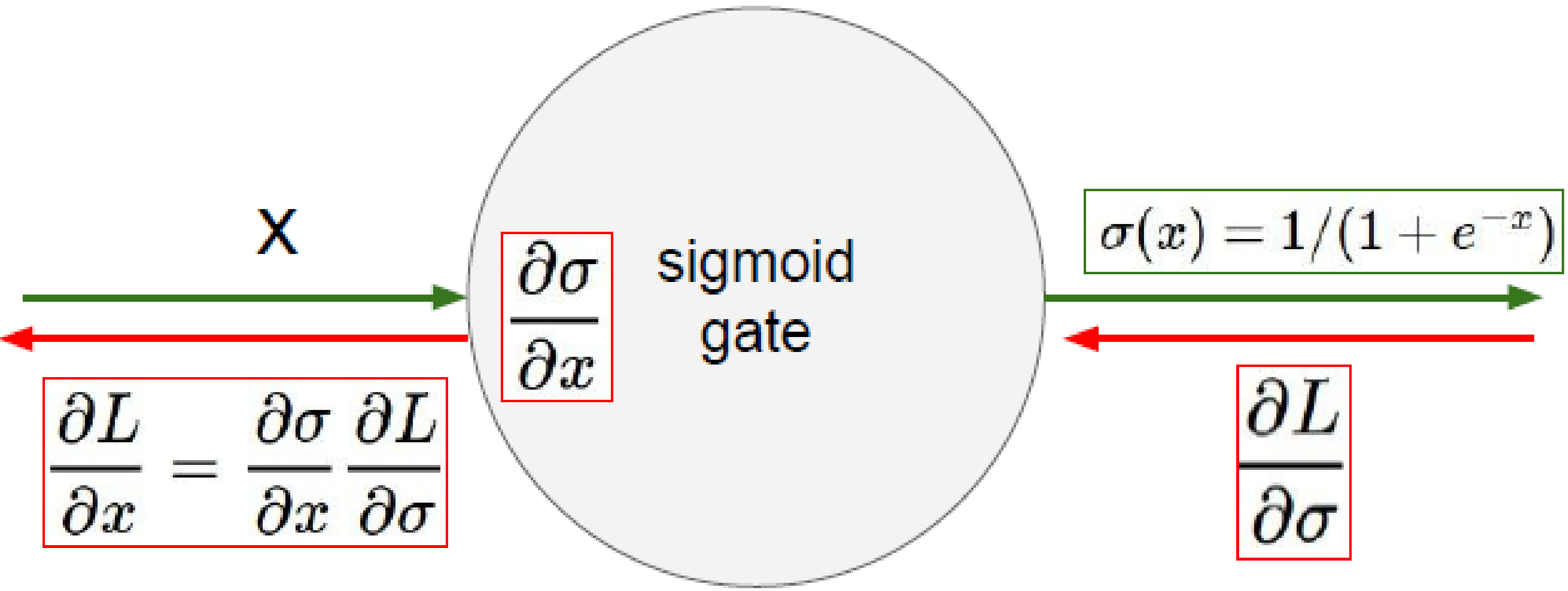
$$\sigma(x) = 1/(1+e^{-x})$$



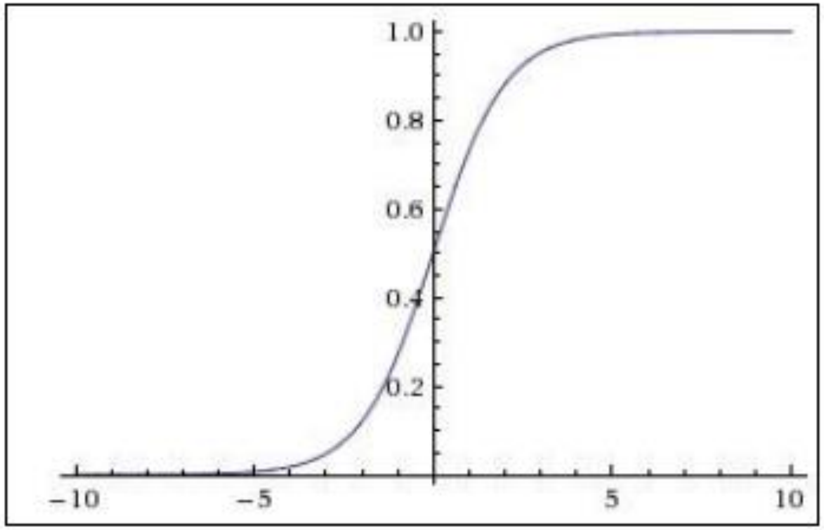
Tri problémy:

1. Saturované neuróny „zabíjajú“ gradienty

Aktivačné funkcie: sigmoid



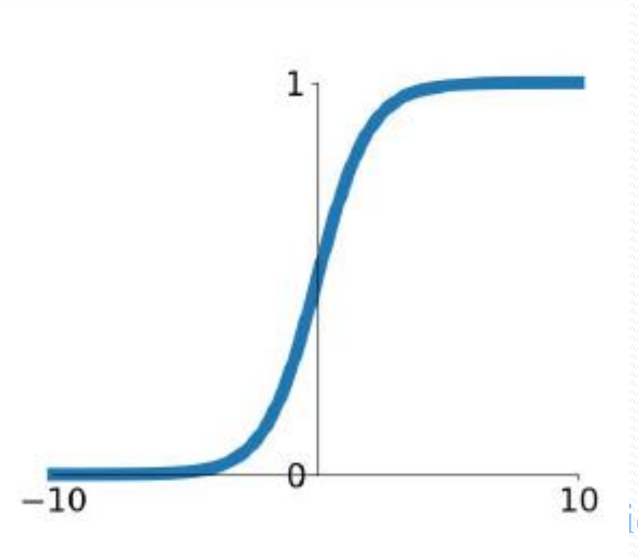
- Čo sa stane, ak x = -10?
- Čo sa stane, ak x = 0?
- Čo sa stane, ak x = 10?



Aktivačné funkcie: sigmoid

- **Sigmoid** stlačí čísla do intervalu [0,1]
- Historicky je populárny, pre-tože má peknú interpretáciu ako saturačná „miera odozvy“ neurónu

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



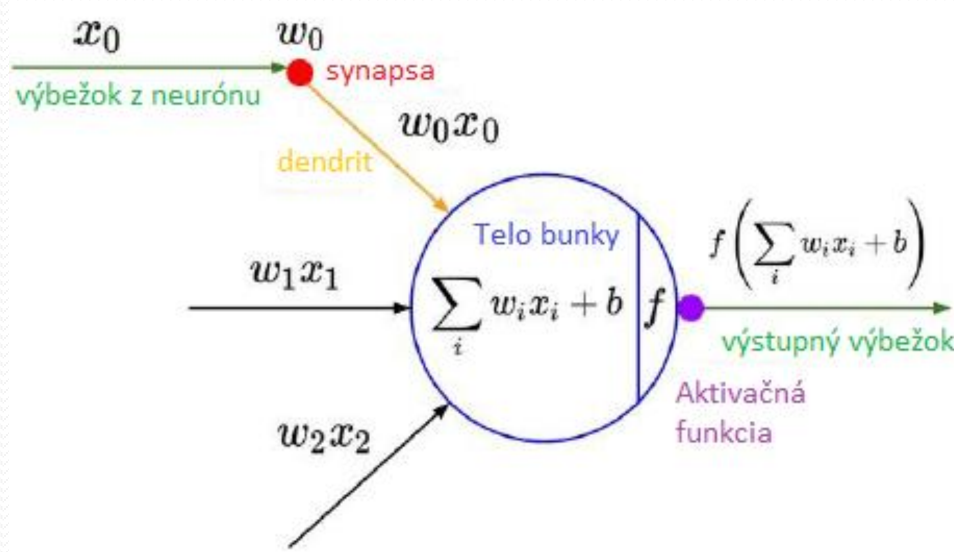
Tri problémy:

1. Saturované neuróny „zabíjajú“ gradienty
2. Výstupy sigmoidu nie sú centrované na nulu

Aktivačné funkcie: sigmoid

- Posúďte, čo sa stane, keď vstupy do neurónu budú vždy kladné ...

$$f\left(\sum_i w_i x_i + b\right)$$

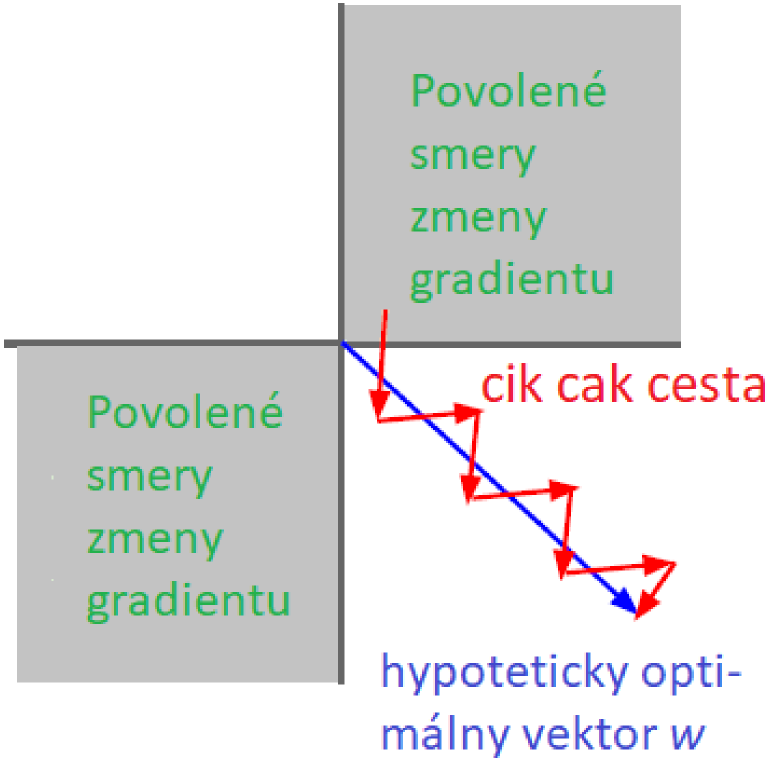


- Čo vieme povedať o gradientoch na ***W***?

Aktivačné funkcie: sigmoid

- Posúďte, čo sa stane, keď vstupy do neuró-nu budú vždy kladné ...

$$f\left(\sum_i w_i x_i + b\right)$$

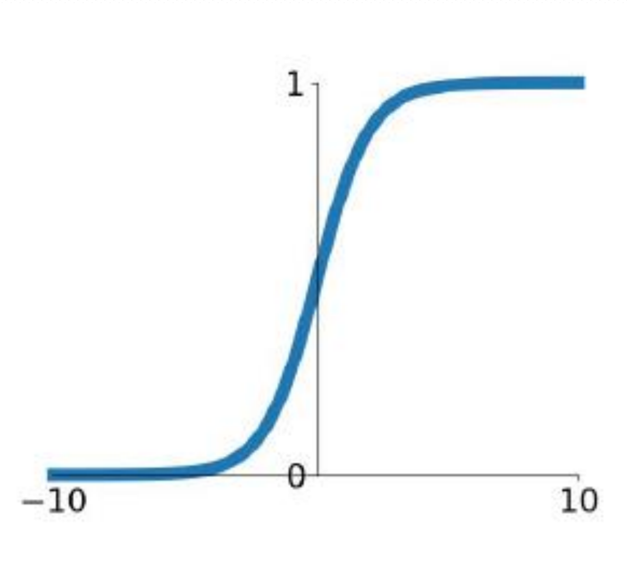


- Čo vieme povedať o gradientoch na ***W***?
- Vždy sú všetky kladné alebo záporné ☹

Aktivačné funkcie: sigmoid

- **Sigmoid** stlačí čísla do intervalu [0,1]
- Historicky je populárny, pre-tože má peknú interpretáciu ako saturačná „miera odozvy“ neurónu

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

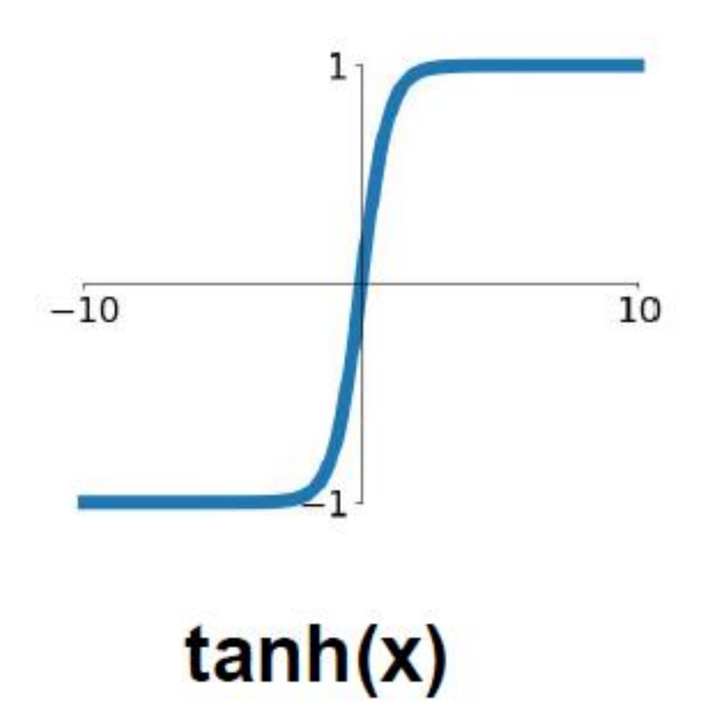


Tri problémy:

1. Saturované neuróny „zabíjajú“ gradienty
2. Výstupy sigmoidu nie sú centrované na nulu
3. Funkcia exp() je výpočtovo náročná

Aktivačné funkcie: tanh(x)

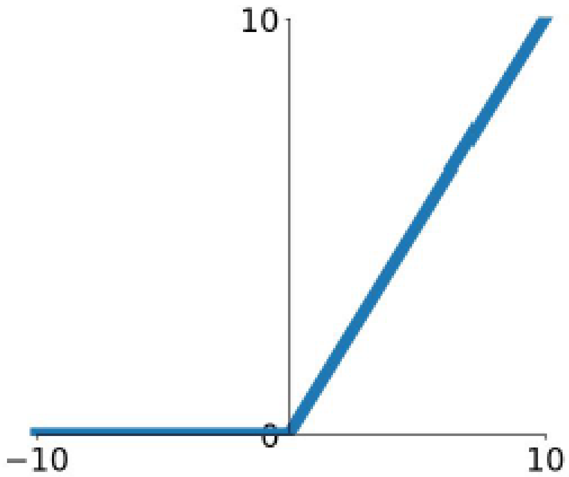
- **Hyperbolický tangens**



- Stlačí čísla do intervalu [-1,1]
- Je pekne centrovaný (na nulu)
- Stále zabíja gradienty, keď je saturovaný ☹

Aktivačné funkcie: ReLU

- **Rektifikovaná lineárna jednotka**



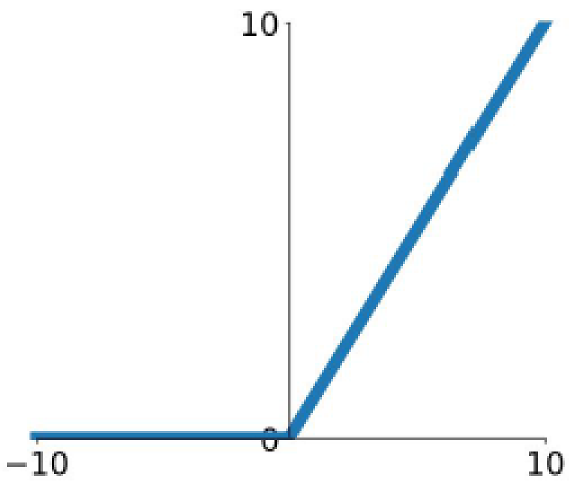
ReLU
(Rectified Linear Unit)

- Počíta **f(x) =max(0,x)**

- Nie je saturovaná (v + oblasti)
- Výpočtovo veľmi efektívna
- Konverguje omnoho rýchlejšie ako sigmoid/tanh (napr. 6x)

Aktivačné funkcie: ReLU

- **Rektifikovaná lineárna jednotka**

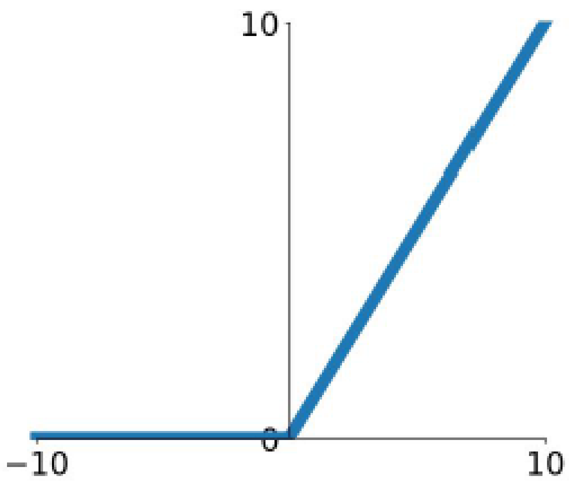


ReLU
(Rectified Linear Unit)

- Počíta **f(x) =max(0,x)**
- Nie je saturovaná (v + oblasti)
- Výpočtovo veľmi efektívna
- Konverguje omnoho rýchlejšie ako sigmoid/tanh (napr. 6x)
- Nemá výstup centrovaný na 0

Aktivačné funkcie: ReLU

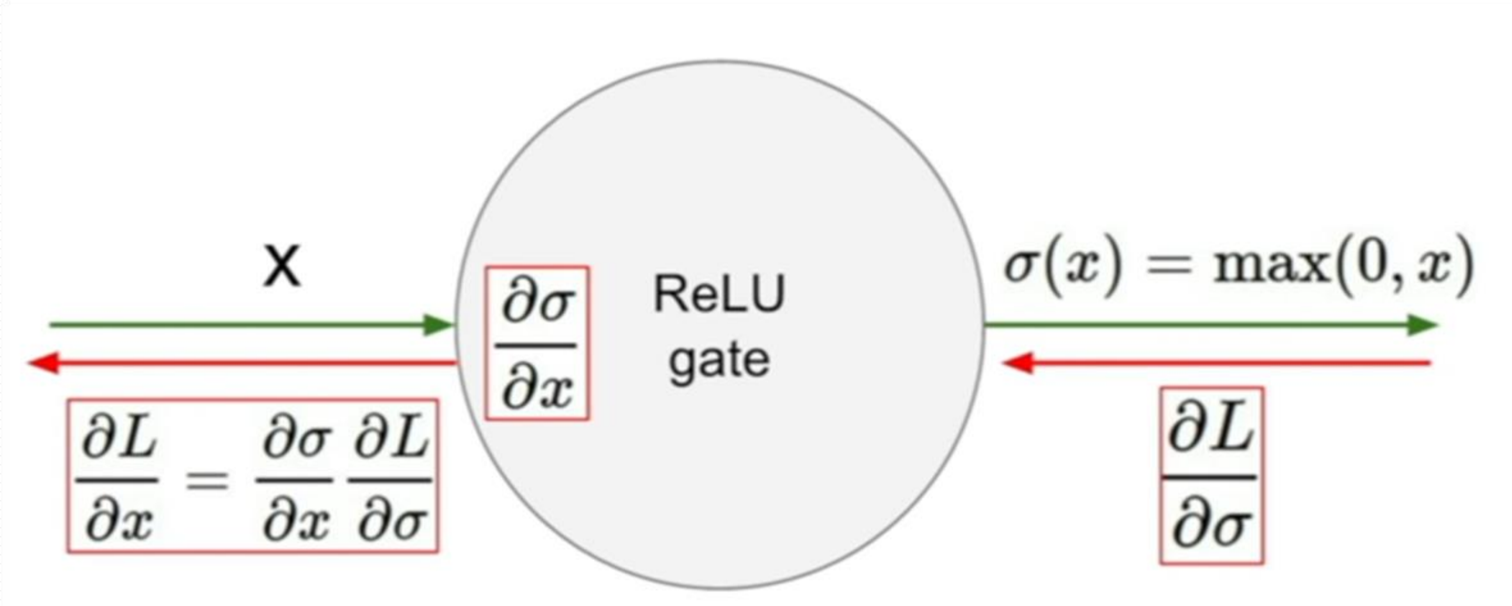
- **Rektifikovaná lineárna jednotka**



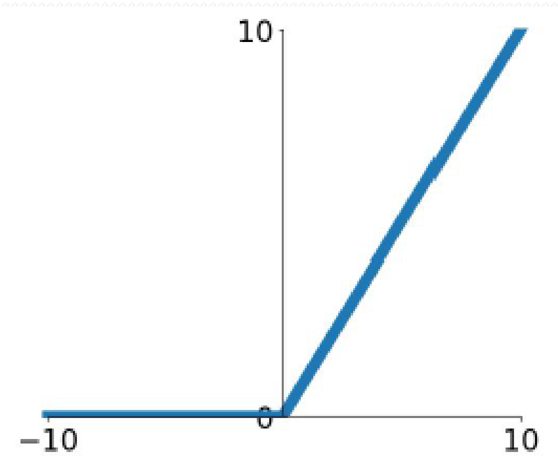
ReLU
(Rectified Linear Unit)

- Počíta **f(x) =max(0,x)**
- Nie je saturovaná (v + oblasti)
- Výpočtovo veľmi efektívna
- Konverguje omnoho rýchlejšie ako sigmoid/tanh (napr. 6x)
- Nemá výstup centrovaný na 0
- To je nepríjemné:
- Návod: aký bude gradient ak $x < 0$

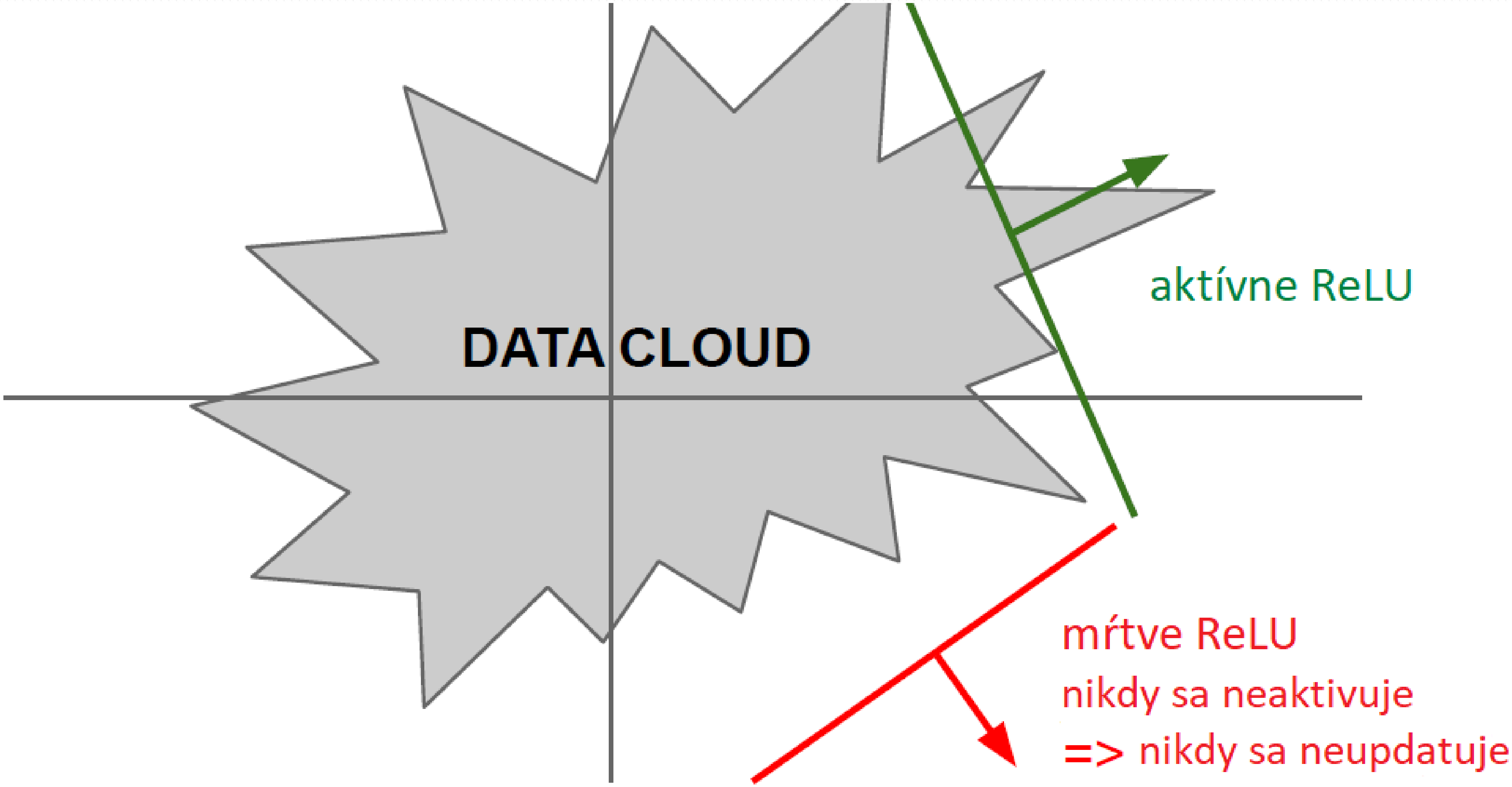
Aktivačné funkcie: ReLU



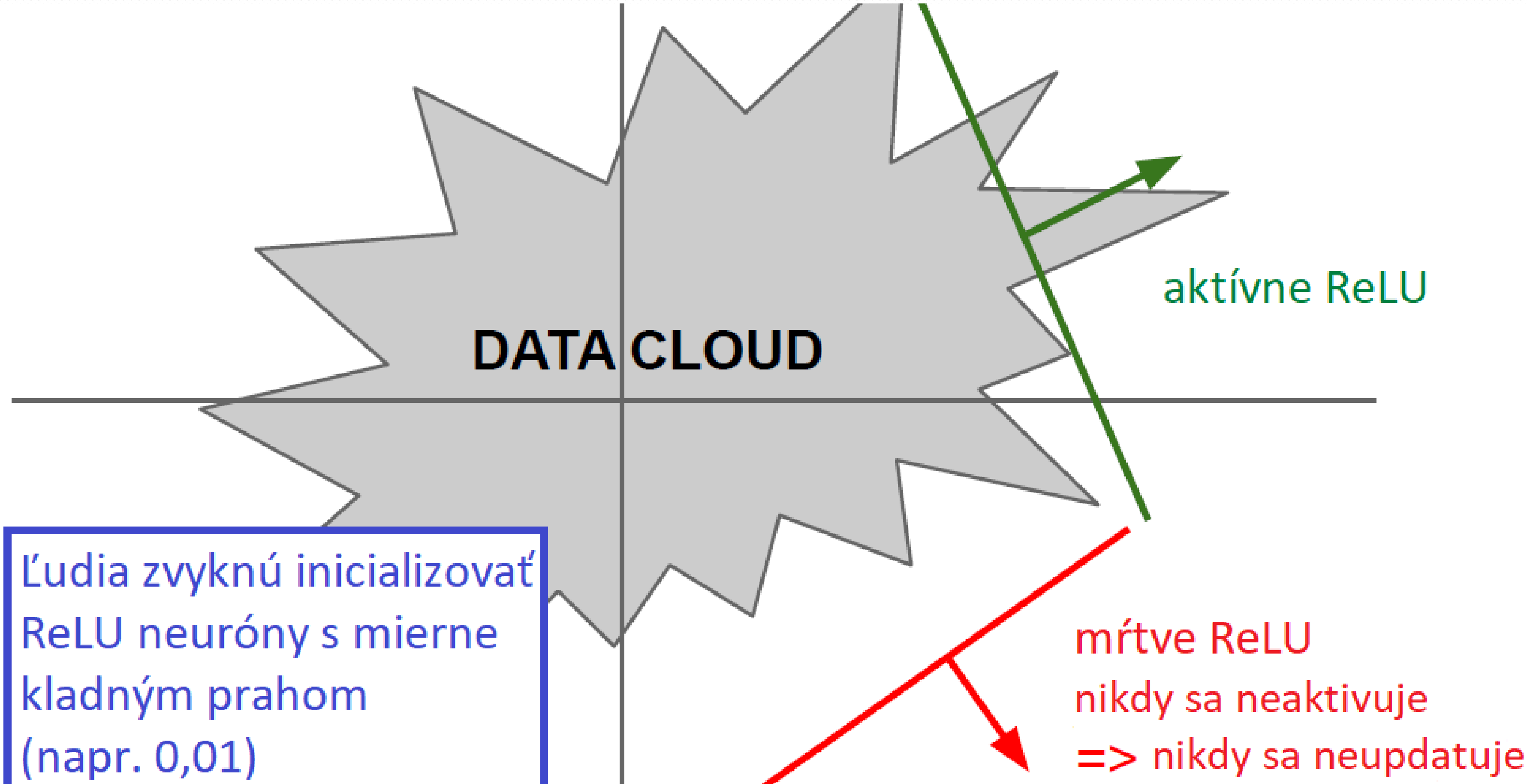
- Čo sa stane, ak *x* = -10?
- Čo sa stane, ak *x* = 0?
- Čo sa stane, ak *x* = 10?



Aktivačné funkcie: ReLU



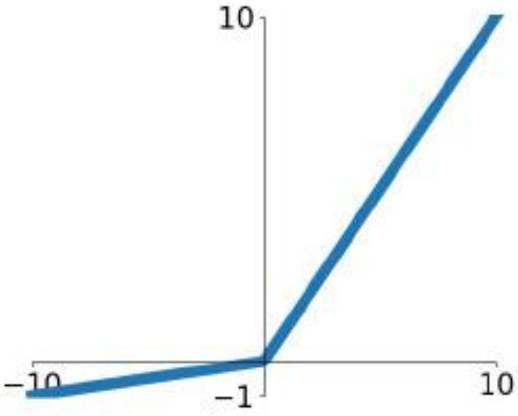
Aktivačné funkcie: ReLU



Ľudia zvyknú inicializovať ReLU neuróny s mierne kladným prahom (napr. 0,01)

Aktivačné funkcie: Leaky ReLU

- **Tečúce ReLU**



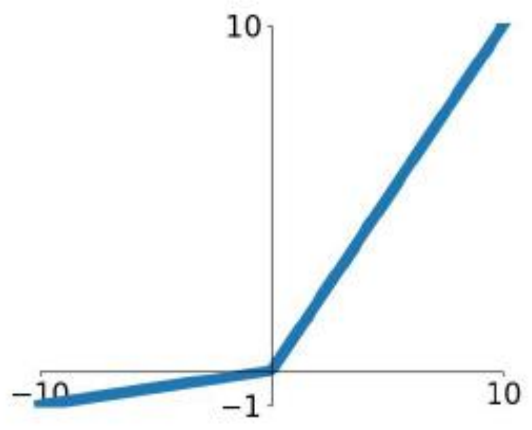
Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Nie je saturovaná
- Je výpočtovo efektívna
- Konverguje omnoho rýchlejšie ako sigmoid/tanh (napr. 6x)
- **„Neumrie“**

Aktivačné funkcie: Leaky ReLU

- **Tečúce ReLU**



Leaky ReLU

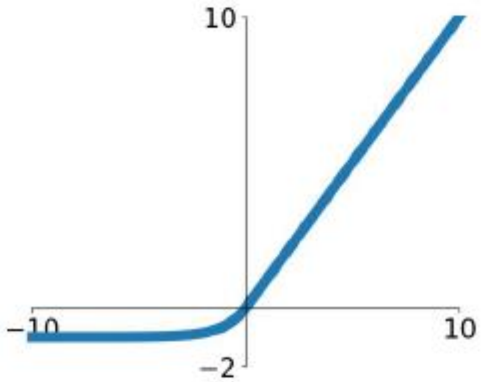
$$f(x) = \max(0.01x, x)$$

- Nie je saturovaná
- Je výpočtovo efektívna
- Konverguje omnoho rýchlejšie ako sigmoid/tanh (napr. 6x)
- **„Neumrie"**

- **Parametrický rektifikátor (PReLU)**
- $f(x) = max(\alpha x, x)$
- α je parameter na backprop

Aktivačné funkcie: ELU

- **Exponenciálne lineárne jednotky (Exponential linear units)**



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Má všetky výhody ReLU
- Je bližšie k výstupom s nulovou strednou hodnotou
- Negatívny saturačný režim v porovnaní s Leaky ReLU pridáva určitú robustnosť voči šumu
- Výpočty vyžadujú počítať exp(x)

Aktivačné funkcie: maxout

- **Maxout „neurón“**
- Nemá základnú formu násobenia po zložkách ➡ nelinearita
- Zovšeobecňuje ReLU a Leaky ReLU
- Lineárny režim! Nemá problém so saturáciou! Neumrie!!

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

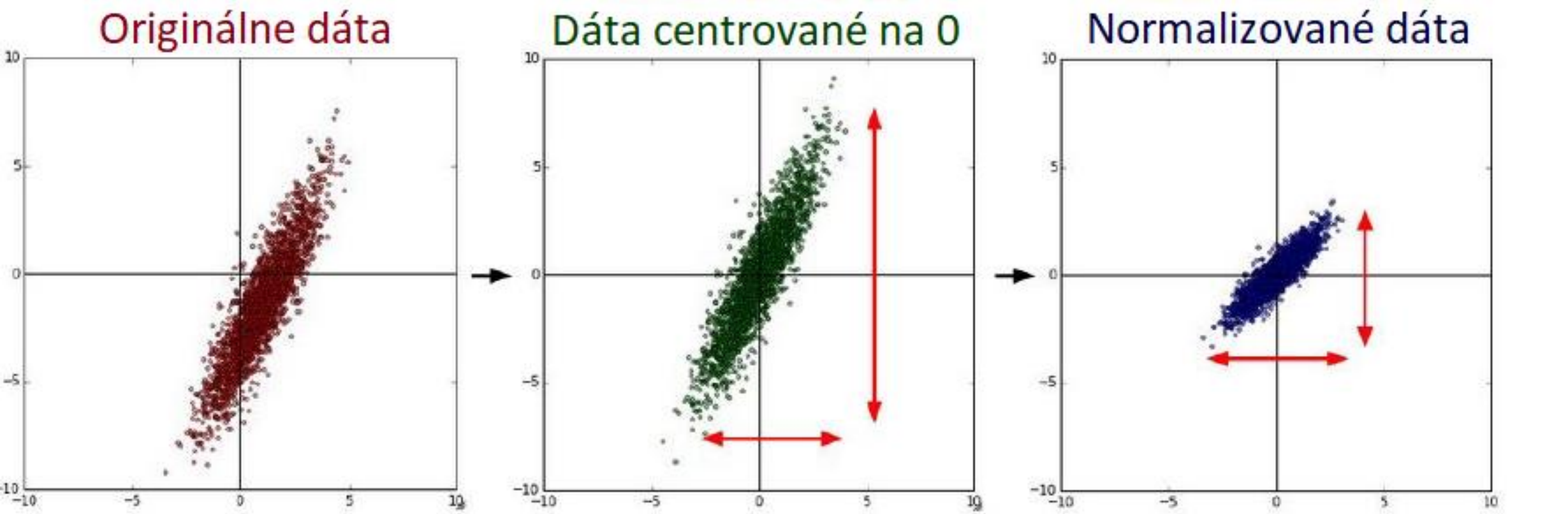
- Problém: zdvojnásobuje počet parametrov neurónu ☹

Dobré rady pre prax

- Používajte ReLU. Buďte opatrní pri voľbe krokov učenia
- Skúste Leaky ReLU/Maxout/ELU
- Skúste tanh, ale nečakajte príliš veľa
- Nepoužívajte sigmoid

Predspracovanie dát

Predpokladajme, že X rozmeru [NxD] je dátová matica a každý príklad rozmeru D je v jednom riadku



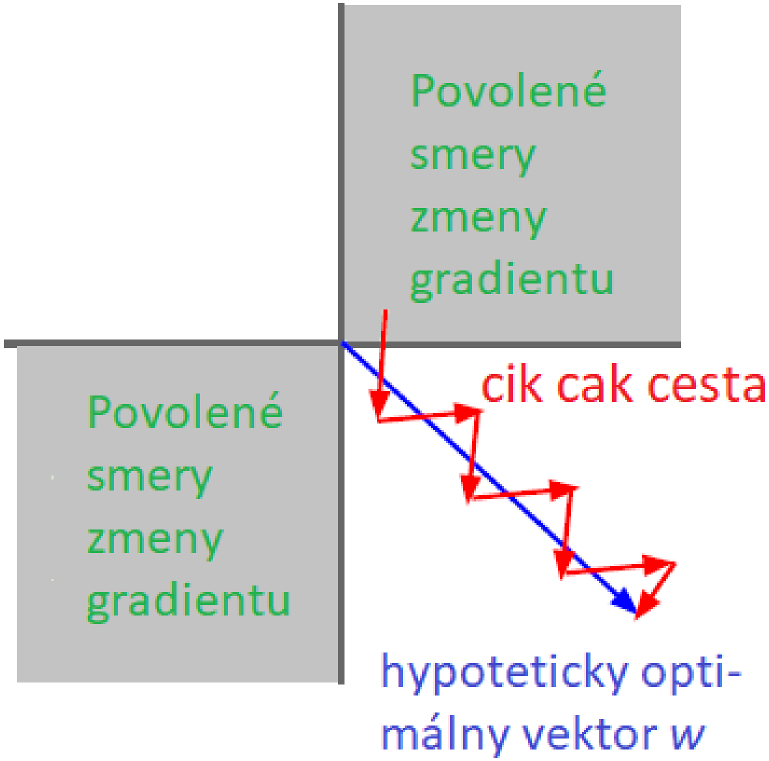

```
X -= np.mean(X, axis = 0)
```

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

Predspracovanie dát

- Pripomenieme: Zvážte, čo sa stane, keď vstup do neurónu bude vždy kladný

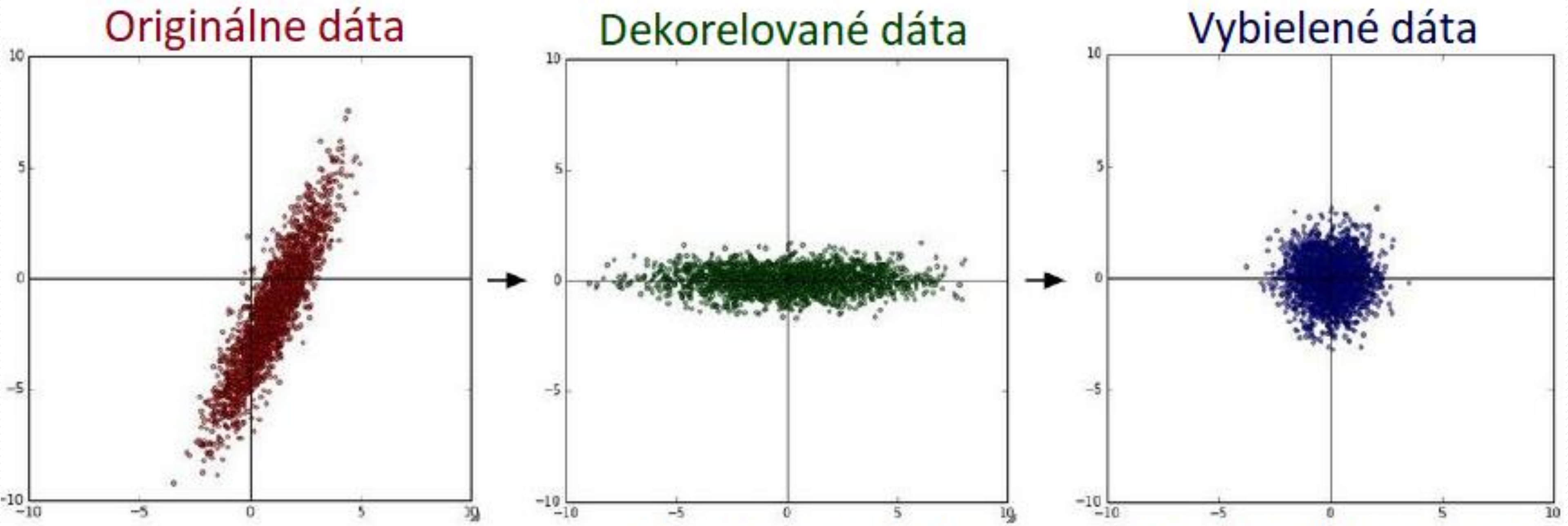
$$f\left(\sum_i w_i x_i + b\right)$$



- Čo vieme povedať o gradientoch na ***W***?
- Vždy sú všetky kladné alebo záporné ☹
- Aj preto chceme mať dáta centrované na nulu!

Predspracovanie dát

V praxi môžeme stretnúť ešte **PCA** a **Bielenie** dát

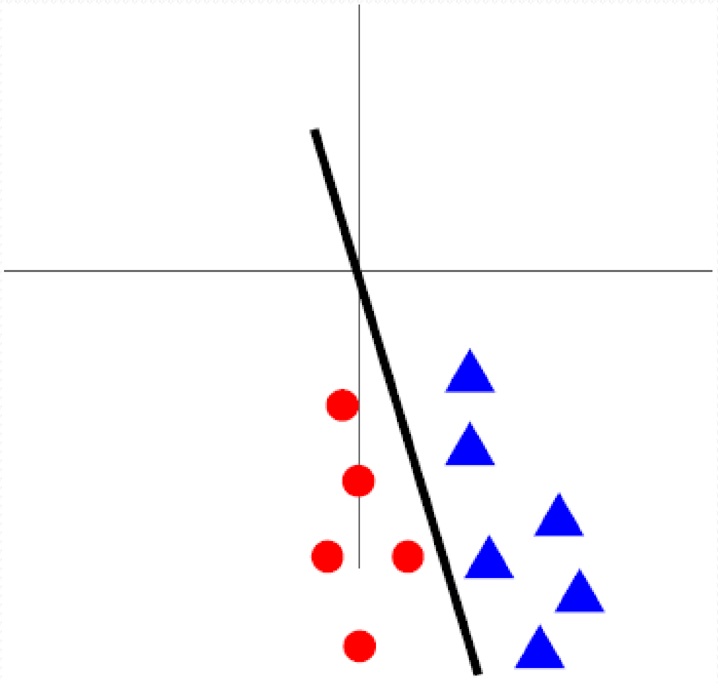


Dáta majú diagonálnu kovariančnú maticu

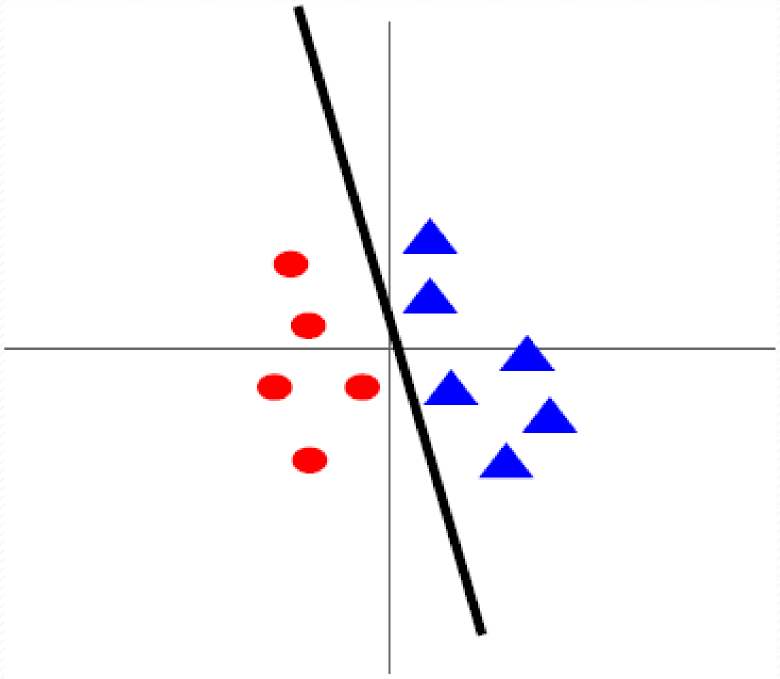
Kovariančná matica je identická matica

Predspracovanie dát

Pred normalizáciou: klasifikačná strata je veľmi senzitívna na zmeny v matici váh; ťažko sa optimalizuje



Po normalizácii: menej senzitívne na malé zmeny v dátach; ľahšie sa optimalizuje

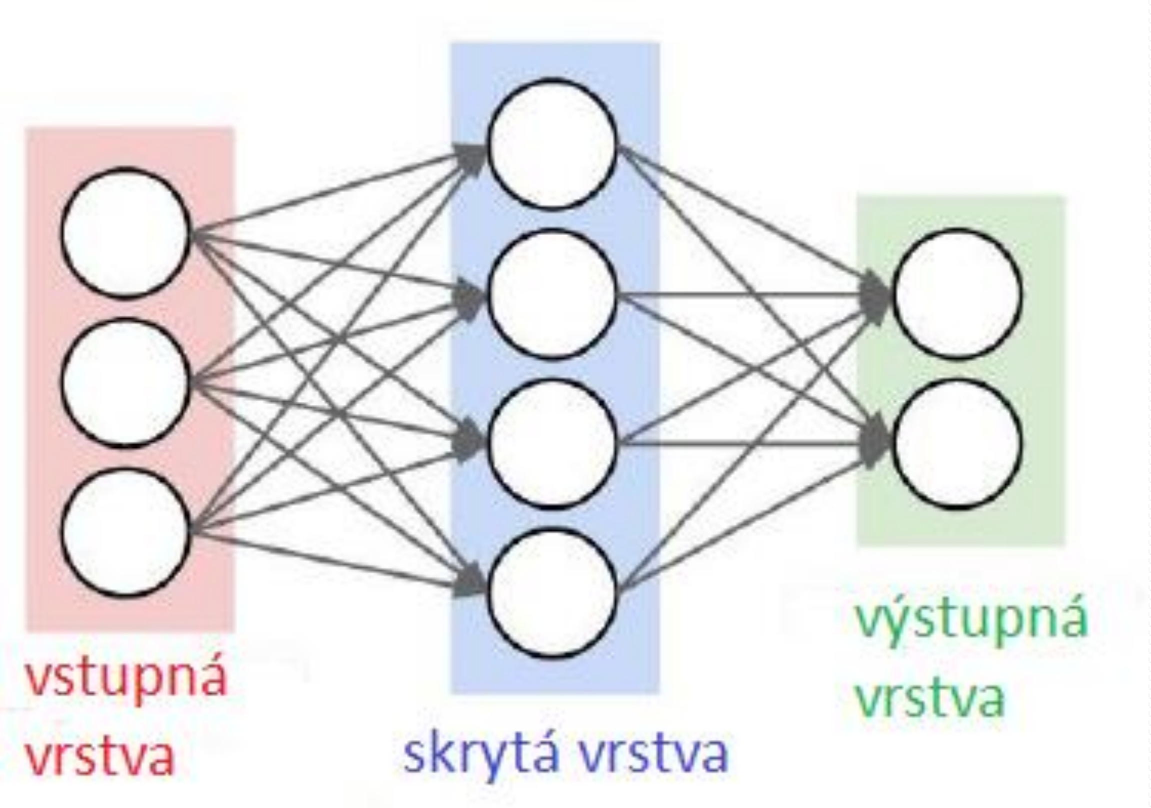


Dobré rady pre obrazy: iba centrujte

- Napr. zoberte CIFAR10 s obrazmi [32x32x3]
- Odčítajte priemerný obraz (napr. AlexNet)
  Priemerný obraz = pole [32x32x3]
- Odčítajte strednú hodnotu po kanáloch (napr. VGG Net) Stredné hodnoty = 3 čísla
- Odčítajte strednú hodnotu po kanáloch a deľte strednou hodnotou po kanáloch (napr. ResNet)
  Stredné hodnoty = 3 čísla   Nerobte PCA ani bielenie

Inicializácia váh

- Otázka: čo sa stane, ak zvolíme za *W* konštant-nú inicializáciu?



Inicializácia váh

- Prvá myšlienka: **Malé náhodné čísla**
- Gaussián s nulovou strenou hodnotou a odchýlkou 10^{-2}

```
W = 0.01 * np.random.randn(Din, Dout)
```

- Funguje relatívne dobre na malých sieťach, ale má problémy na hlbokých sieťach

Aktivačné štatistiky

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

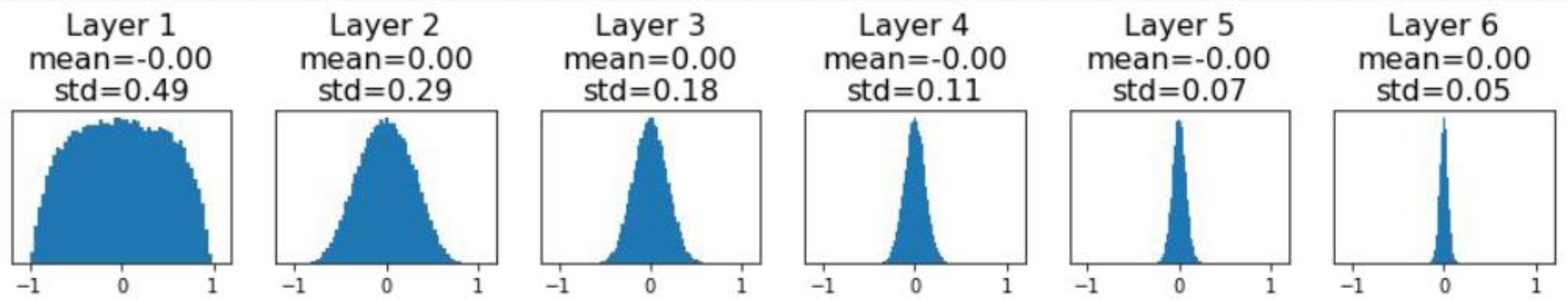
Forward pass for a 6-layer net with hidden size 4096

Aktivačné štatistiky

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

- Všetky aktivácie smerujú k nule pre hlbšie vrstvy siete
- **Ot:** Ako budú vyzerať gradienty dL/dW?

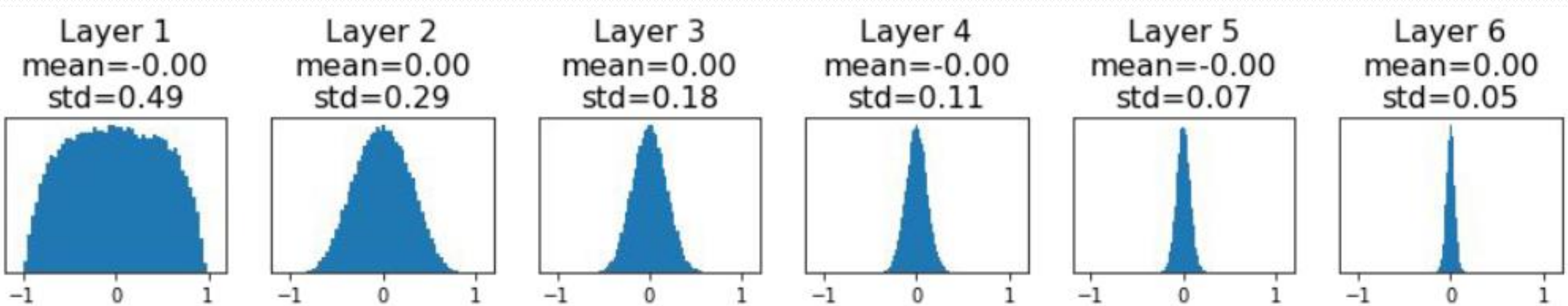


Aktivačné štatistiky

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

- Všetky aktivácie smerujú k nule pre hlbšie vrstvy siete
- **Ot:** Ako budú vyzerať gradienty dL/dW?
- **Od:** Všetky budú 0, t.j. žiadne učenie ☹



Aktivačné štatistiky

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Increase std of initial weights from 0.01 to 0.05

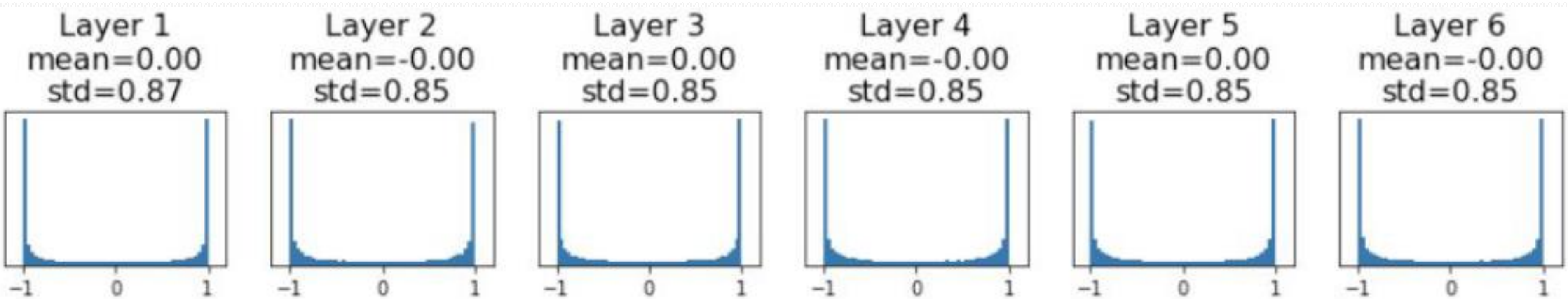
- Všetky aktivácie sú saturované
- **Ot:** Ako vyzerajú gradienty?

Aktivačné štatistiky

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Increase std of initial weights from 0.01 to 0.05

- Všetky aktivácie sú saturované
- **Ot:** Ako vyzerajú gradienty?
- **Od:** Lokálne gradienty sú všetky nula, žiadne učenie ☹



„Xavierova“ inicializácia

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

„Xavierova“ inicializácia

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

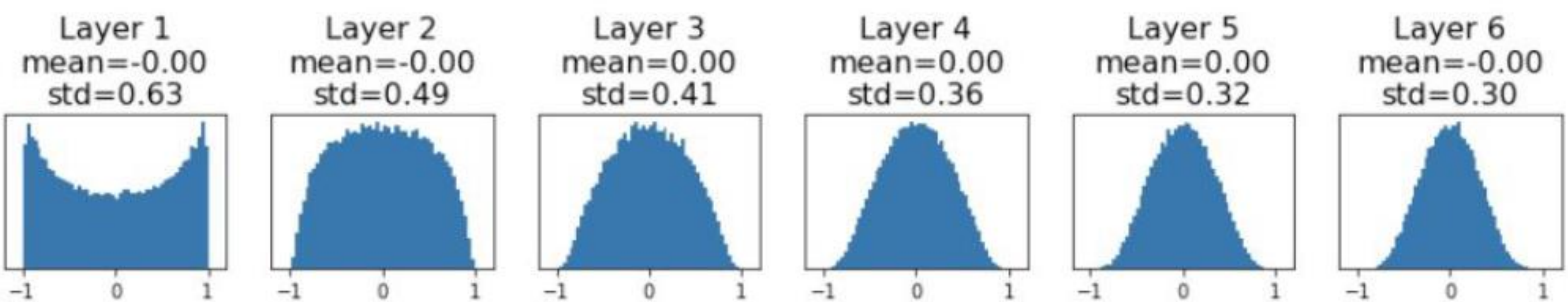
- Správne. Aktivácie sú pekne škálované vo všetkých vrstvách!

„Xavierova“ inicializácia

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

- Správne. Aktivácie sú pekne škálované vo všetkých vrstvách!
- Pre konv. vrstvy, Din je $\text{kernel\_size}^2 * \text{input\_channels}$



Inicializácia váh s ReLU

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

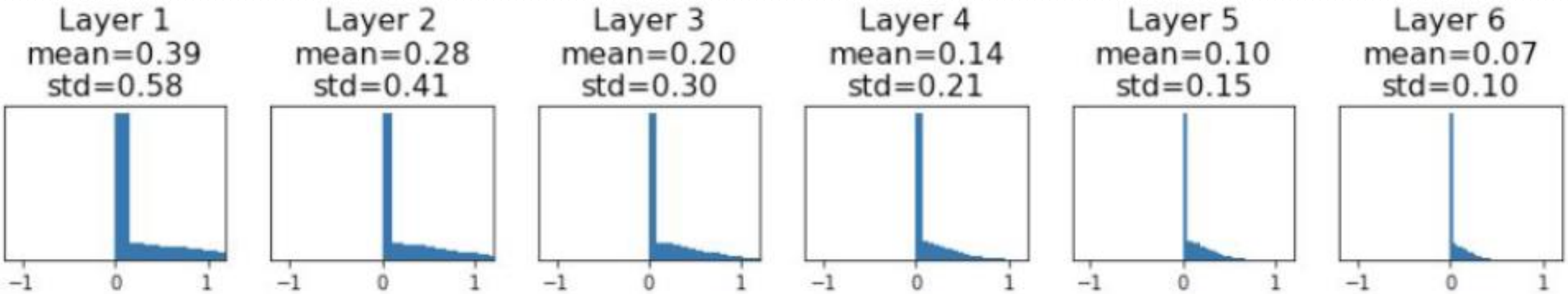
Change from tanh to ReLU

Inicializácia váh s ReLU

Change from tanh to ReLU

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

- Xavier predpokladá aktivačnú funkciu centrovanú na nulu
- Aktivácie opäť skolabujú k nule, žiadne učenie ☹

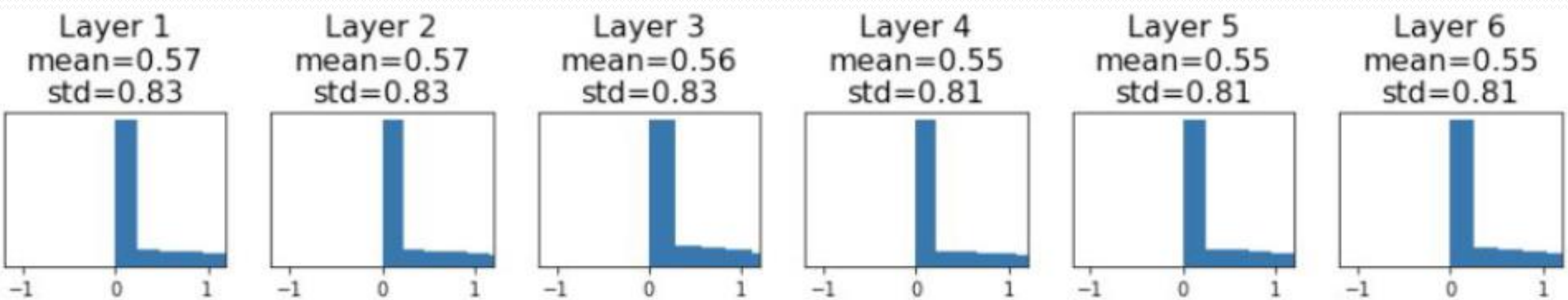


Kaimingova/MSRA inicializácia

ReLU correction: std = sqrt(2 / Din)

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

- Správne. Aktivácie sú pekne škálované vo všetkých vrstvách!



Dobrá inicializácia

- Je stále predmetom aktívneho výskumu ...

Understanding the difficulty of training deep feedforward neural networks
by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

Fixup Initialization: Residual Learning Without Normalization, Zhang et al, 2019

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, Frankle and Carbin, 2019

Normalizácia balíka (BN)

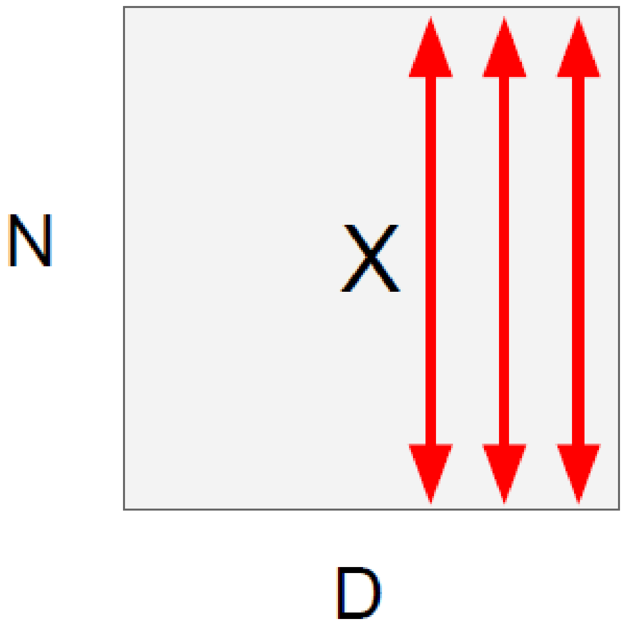
- Batch normalisation [Ioffe a Szegedy, 2015]
- „Chcete mať aktivácie s rozdelením *(0,1)*, t.j. s nulovou strednou hodnotou a jednotkovou odchýlkou? Tak ich také urobte.“
- Uvažujme balík aktivácií v nejakej vrstve. Na získanie rozdelenia *(0,1)* použite

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

To je jednoducho dife-rencovateľná funkcia ...

Normalizácia balíka (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*

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

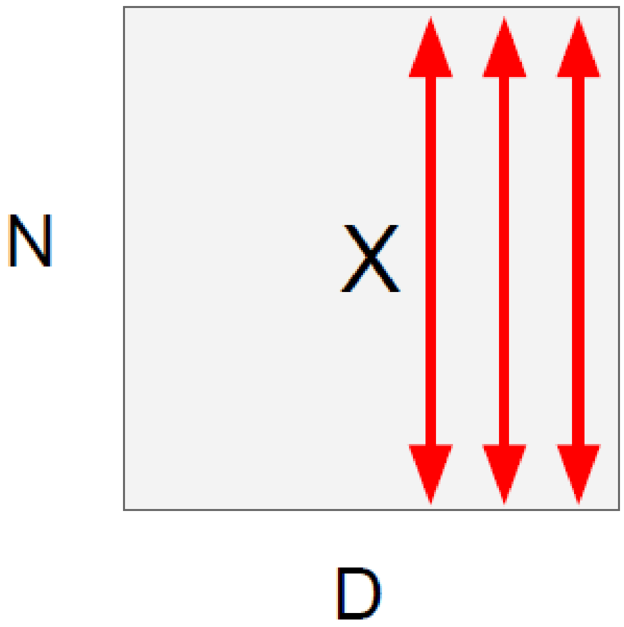
Odchýlka po kanáloch, rozmer je *D*

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

Normalizované *x*, rozmer je *N x D*

Normalizácia balíka (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 x D*

Problém: čo je keď je podmienka *(0,1)* príliš ťažká

Normalizácia balíka (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*

Parametre škály a posunu nastavovaně učením

$$\gamma, \beta: D$$

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

Normalizované *x*, rozmer je *N x D*

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

Výstup, rozmer je *D*

Výsledok učenia $\gamma = \sigma, \beta = \mu$ znamená identickú funkciu = žiadnu BN

BN pri testovaní

Odhady závisia na minibalíku; to sa nedá robiť pri testovaní

Vstup: $x: N \times D$

Parametre škály a posunu nastavovanê učením

$$\gamma, \beta : D$$

Výsledok učenia $\gamma = \sigma, \beta = \mu$ vedie na identickú funkciu

$$\boxed{\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j} \qquad \text{Stredná hodnota po kanáloch, rozmer je } D \qquad \sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2 \qquad \text{Odchýlka po kanáloch, rozmer je } D}$$

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

Normalizované *x*, rozmer je *N x D*

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

Výstup, rozmer je *D*

BN pri testovaní

Vstup: $x: N \times D$

Parametre škály a posunu nastavova-né učením

$$\gamma, \beta: D$$

Počas testovania sa BN stáva lineárnym operá-torom. Môže sa spojiť s predošlou FC alebo konvolučnou vrstvou

$\mu_j =$ (Pri behu) Priemer hodnôt videných počas trénovania — Stredná hodnota po kanáloch, rozmer je *D*

$\sigma_j^2 =$ (Pri behu) Priemer hodnôt videných počas trénovania — 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 x D*

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

Výstup, rozmer je *D*

BN pri testovaní

Vstup: $x: N \times D$

Parametre škály a posunu nastavova-né učením

$$\gamma, \beta: D$$

Výsledok učenia $\gamma = \sigma, \beta = \mu$ vedie na identickú funkciu

$$\mu_j = \text{(Pri behu) Priemer hodnôt videných počas trénovania}$$

Stredná hodnota po kanáloch, rozmer je *D*

$$\sigma_j^2 = \text{(Pri behu) Priemer hodnôt videných počas trénovania}$$

Odchýlka po kanáloch, rozmer je *D*

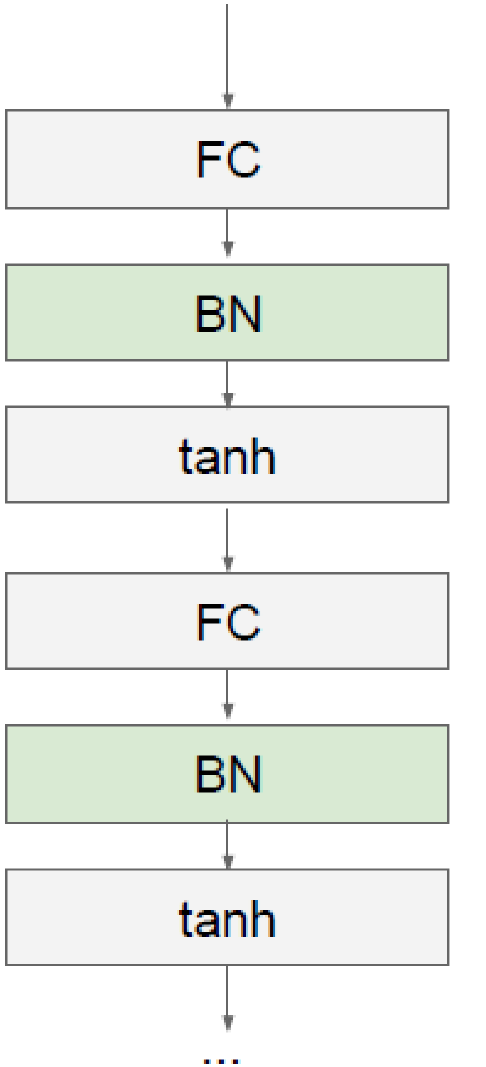
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalizované *x*, rozmer je *N x D*

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

Výstup, rozmer je *D*

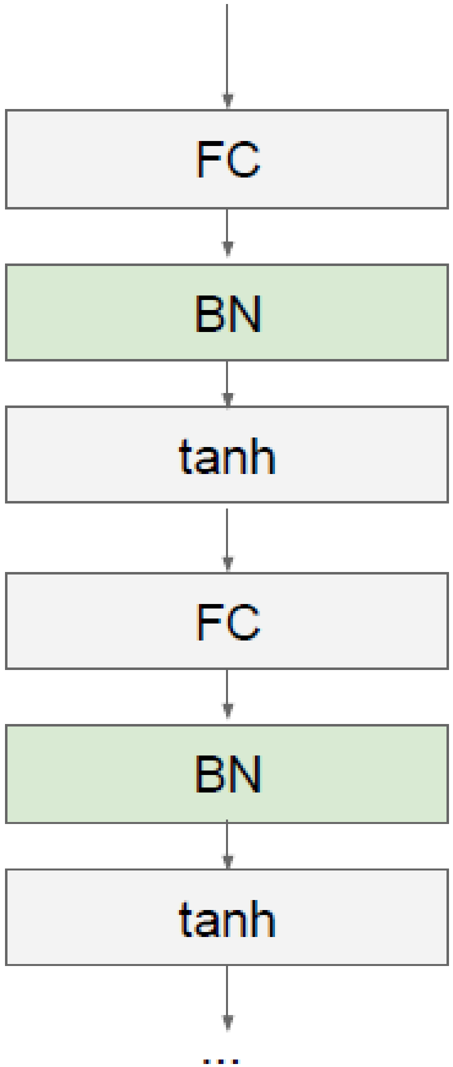
Normalizácia balíka (BN)



Obyčajne sú vrstvy normalizácie vložené po plne prepojených alebo konvolučných vrstvách a pred nelinearitou

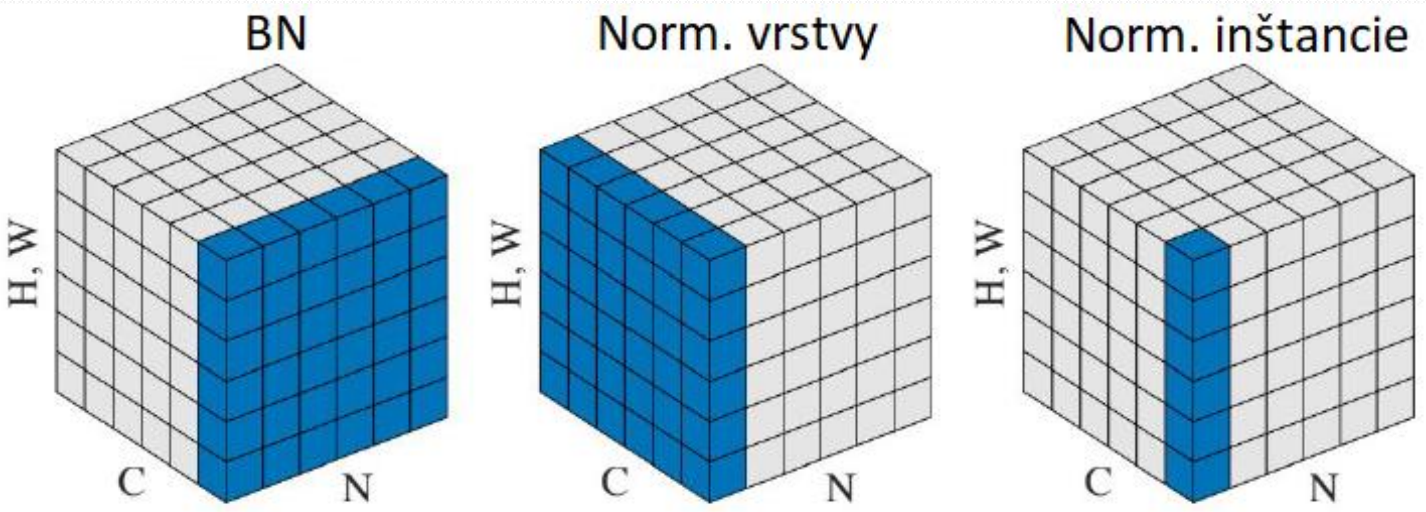
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Normalizácia balíka (BN)

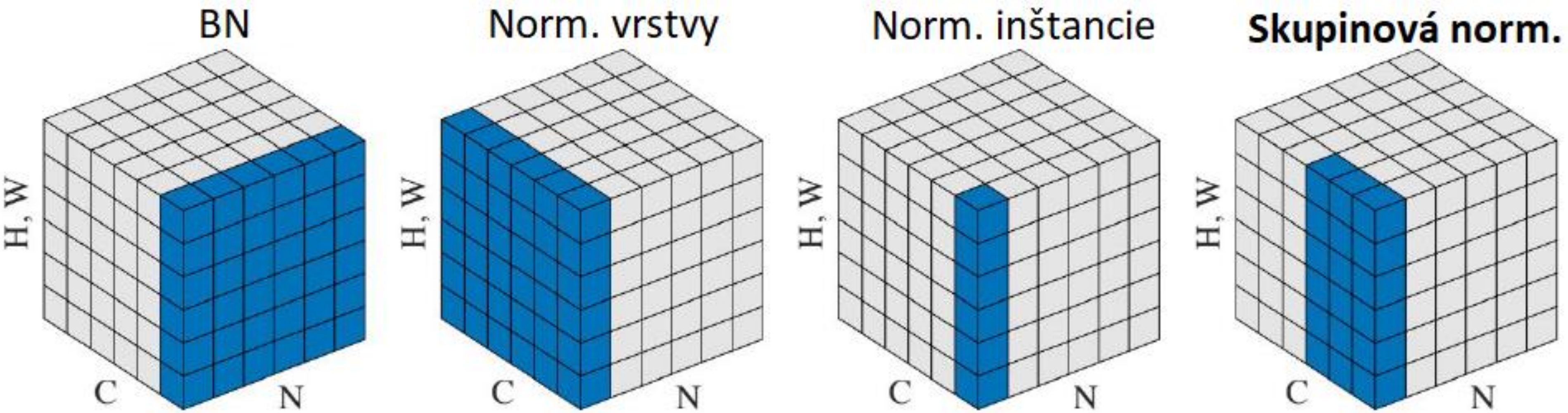


- Vďaka BN sa hlboké siete **omnoho** ľahšie trénujú!
- Zvyšuje tok gradientu
- Umožňuje väčšie kroky učenia a rýchlejšiu konvergenciu
- Siete sa stávajú robustnejšie voči inicializácii
- Počas trénovania funguje ako regularizácia
- Nulové zdržanie pri testovaní, dá sa spojiť s konvolučnou vrstvou
- Správa sa rozdielne počas trénovania a testovania, to je najčastejší zdroj chýb!!

Porovnanie normalizačných vrstiev



Porovnanie normalizačných vrstiev



Zhrnutie

- Detailne sme preskúmali štyri okruhy:
- Aktivačné funkcie (používajte ReLU)
- Predspracovanie dát (pri obrazoch odčítajte priemer)
- Inicializácia váh (použite Xavier/MSRA inicializáciu)
- Normalizácia balíka (používajte)