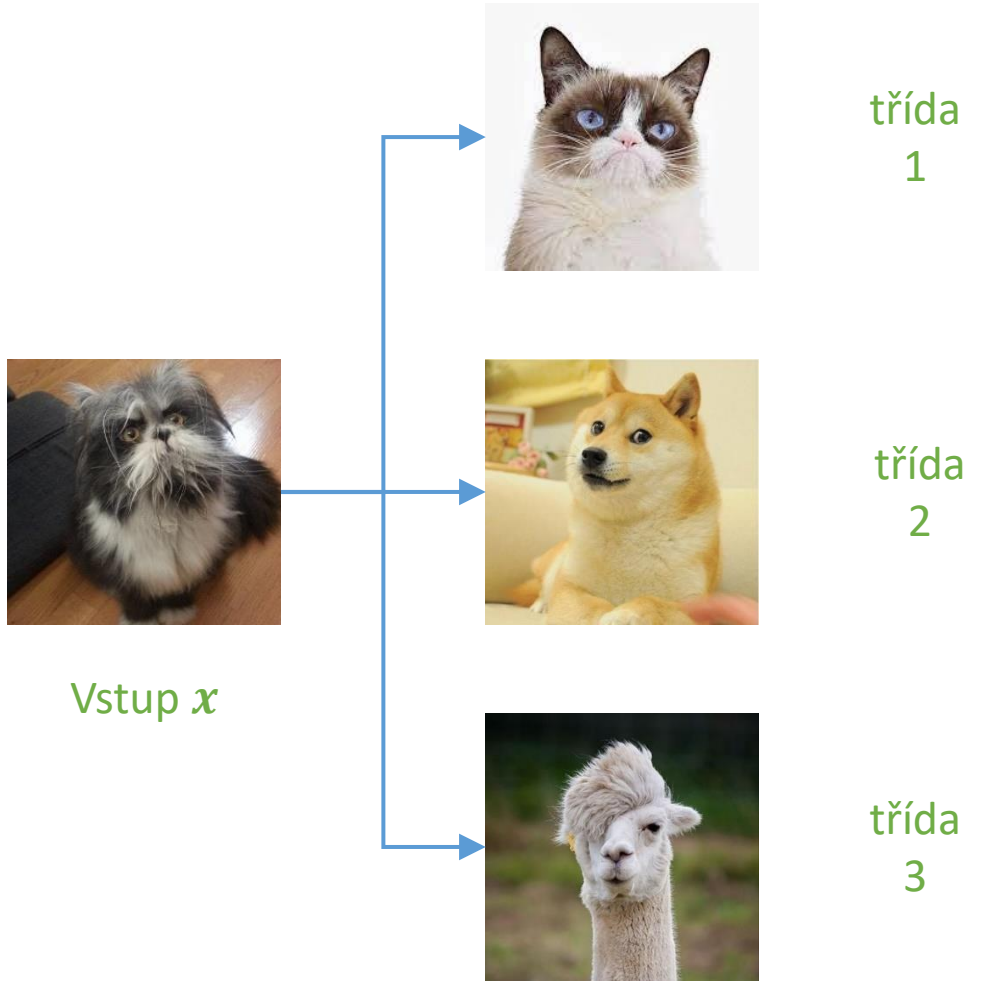
Počítačové vidění

Hluboké neuronové sítě, klasifikace obrázků

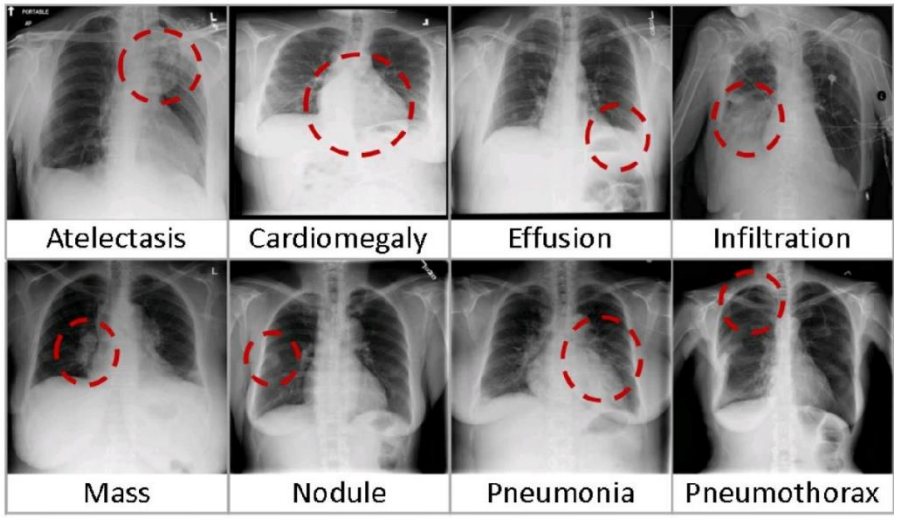
Úloha klasifikace obrázků

- Vstupem $\boldsymbol{x}_n$ je RGB obrázek
- Úkolem zařadit $\boldsymbol{x}_n$ do jedné ze tří předdefinovaných kategorií (tříd):
  1. “kočka”
  2. “pes”
  3. “*alpaka*”
- Počet tříd označíme jako K
- Výstupem bude jedno z následujících:
  - skóre jednotlivých tříd s_n,
  - pravděpodobnost jednotlivých tříd p_n,
  - nebo celé číslo $\hat{y}_n \in \{1, \dots, K\}$



Proč klasifikace?

Klasifikace je zajímavá a užitečná sama o sobě




Amanita_fulva1


Lepista_saeva48


Coprinopsis_atramentaria35


Morchella_esculenta 33


Amanita_fulva11


Auricularia_auricula-judae22

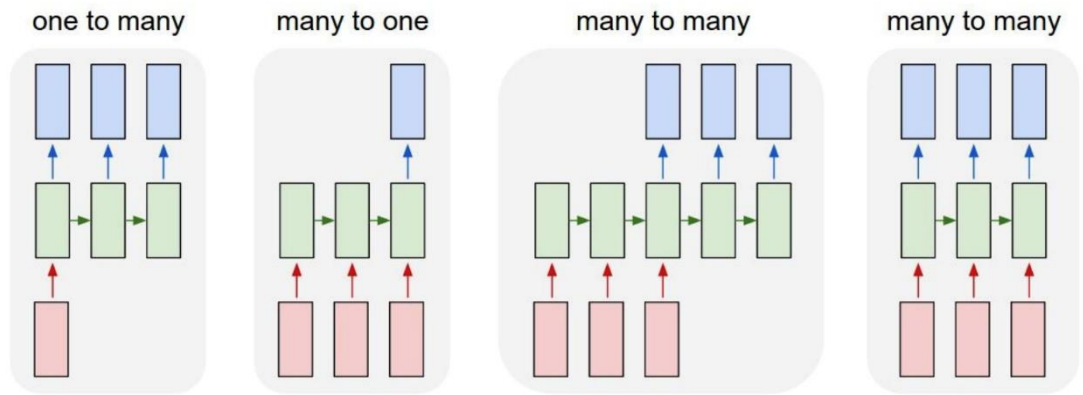

Pleurocybella_porrigens13


Galerina_sulciceps19


Amanita_arocheae37


Clavulinaceae15

zároveň ale tvoří základ mnoha dalších aplikací



tagování obrázku, generování hudby

analýza sentimentu, klasifikace videa

strojový překlad, rozpoznávání řeči

rozpoznávání fonémů, jazykový model

Návrh a trénování lineárního klasifkátoru

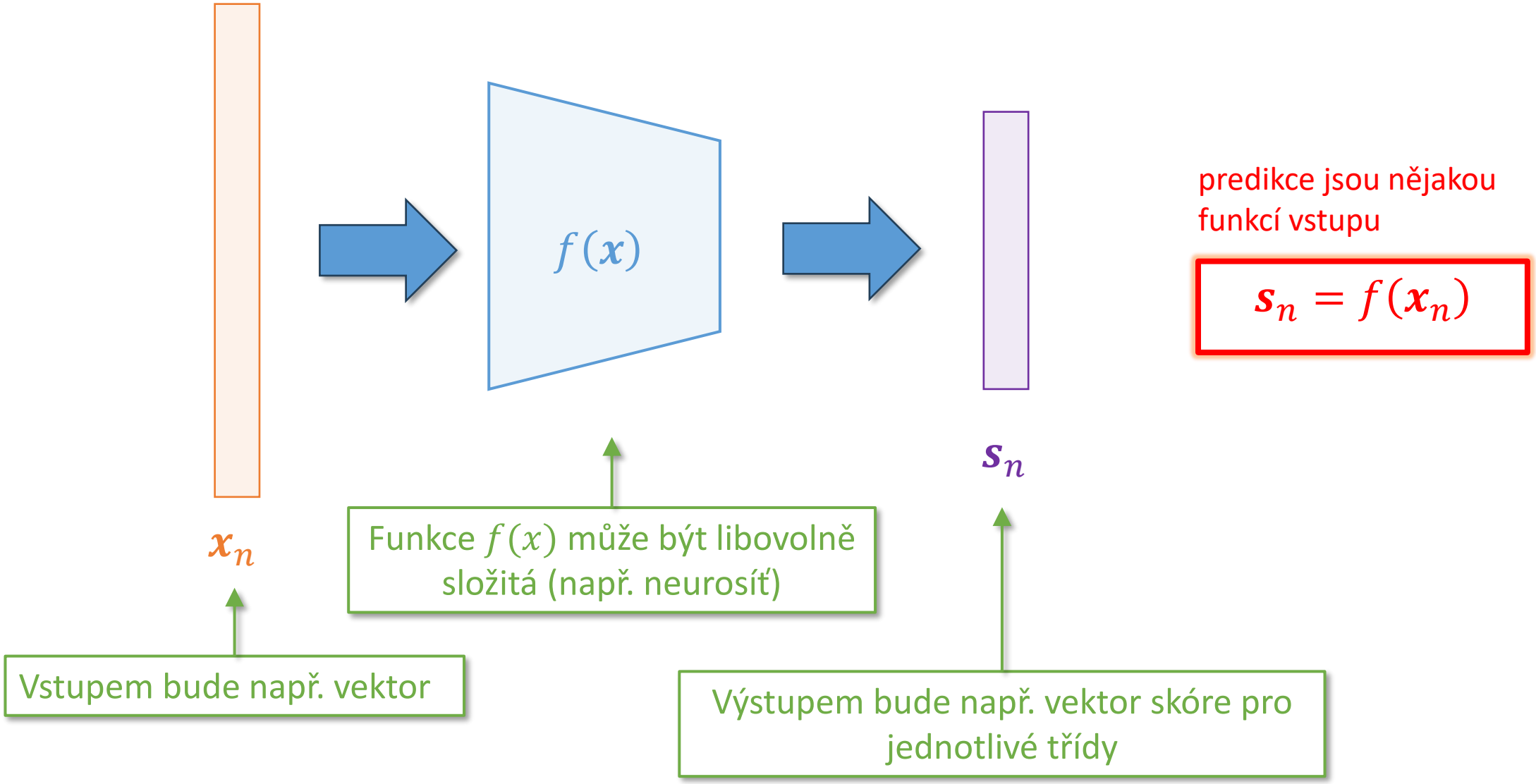
1. Navrhneme diskriminativní klasifikační funkci s upravitelnými parametry
2. Kvantifikujeme její úspěšnost klasifikace nějakým kritériem
3. Nastavíme parametry klasifikátoru tak, abychom optimalizovali zvolené kritérium

Návrh a trénování lineárního klasifkátoru

1. **Navrhneme diskriminativní klasifikační funkci s upravitelnými parametry**
2. Kvantifikujeme její úspěšnost klasifikace nějakým kritériem
3. Nastavíme parametry klasifikátoru tak, abychom optimalizovali zvolené kritérium

Diskriminativní klasifikace

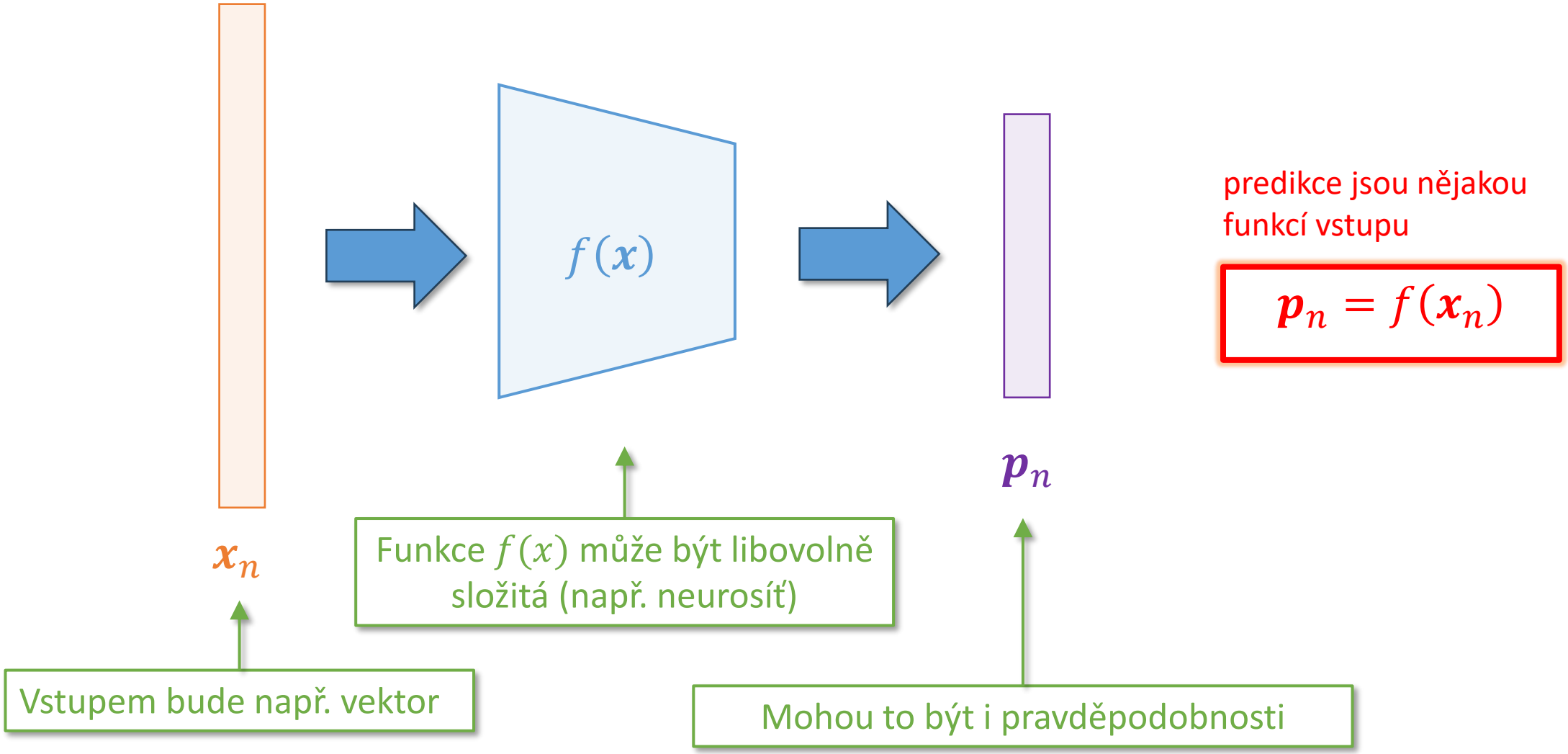
Diskriminativní klasifikátor



predikce jsou nějakou funkcí vstupu

$$\boldsymbol{s}_n = f(\boldsymbol{x}_n)$$

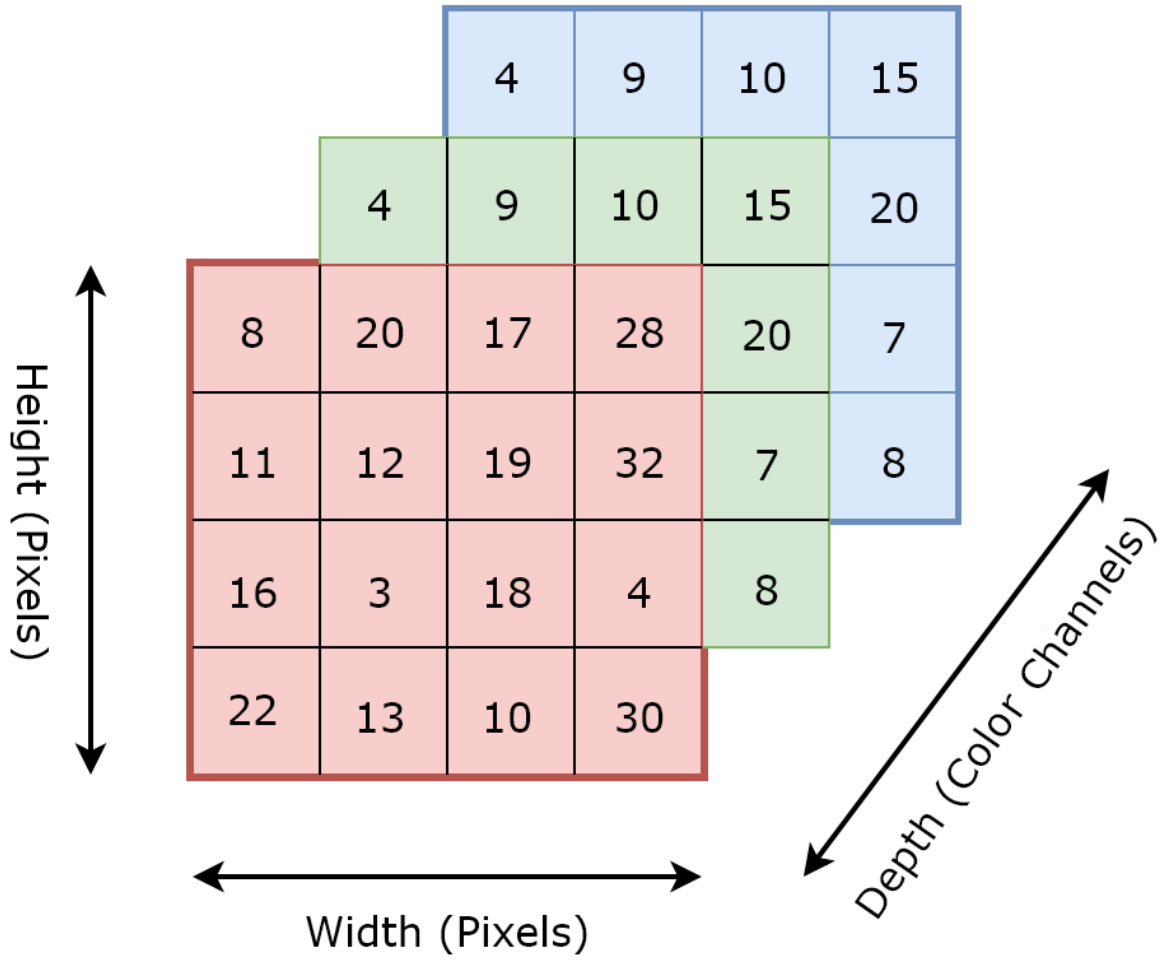
Diskriminativní klasifikátor

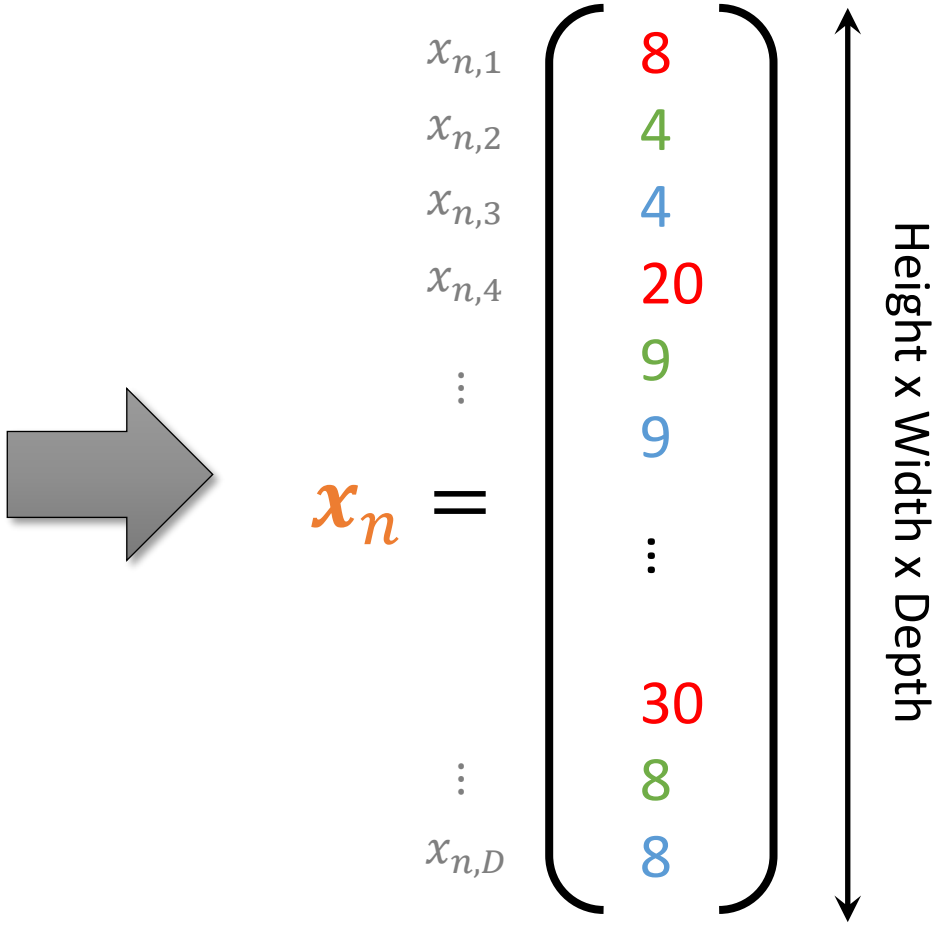


predikce jsou nějakou funkcí vstupu

$$\boldsymbol{p}_n = f(\boldsymbol{x}_n)$$

Reprezentace RGB obrázku jako vektoru





Tensor tvaru (výška, šířka, hloubka) (HWC formát)

Vektor délky D = výška x šířka x hloubka

obrázek: https://www.oreilly.com/library/view/machine-learning-with/9781789346565/6899a0ec-2cf9-4525-b3bf-56dfafa827e7.xhtml

Lineární klasifikátor

- Lineární model předpokládá afinní* vztah mezi skóre třídy $\boldsymbol{s}_n$ a vstupem $\boldsymbol{x}_n$

$$\boldsymbol{s}_n = \boldsymbol{w} \cdot \boldsymbol{x}_n + \boldsymbol{b}$$

kde

- $\boldsymbol{x}_n$ je sloupcový vektor o rozměru D
- $\boldsymbol{w}$ je matice vah klasifikátoru s rozměry $K \times D$
- $\boldsymbol{b}$ je sloupcový vektor biasů klasifikátoru s rozměrem K

} parametry klasifikátoru

Lineární predikce skóre: příklad

$$s_n = w \cdot x_n + b$$

x_n: 2×2

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0.0	0.25	0.2	-0.3

w: 3×4

$\cdot$

56
231
24
2

x_n: 4×1

$+$

2.0
0.0
-0.3

b: 3×1

$=$

-96.8
437.9
61.95

s_n: 3×1

příklad: https://cs231n.github.io/linear-classify/
https://web.eecs.umich.edu/~justincj/slides/eecs498/WI2022/598_WI2022_lecture03.pdf

Geometrická interpretace

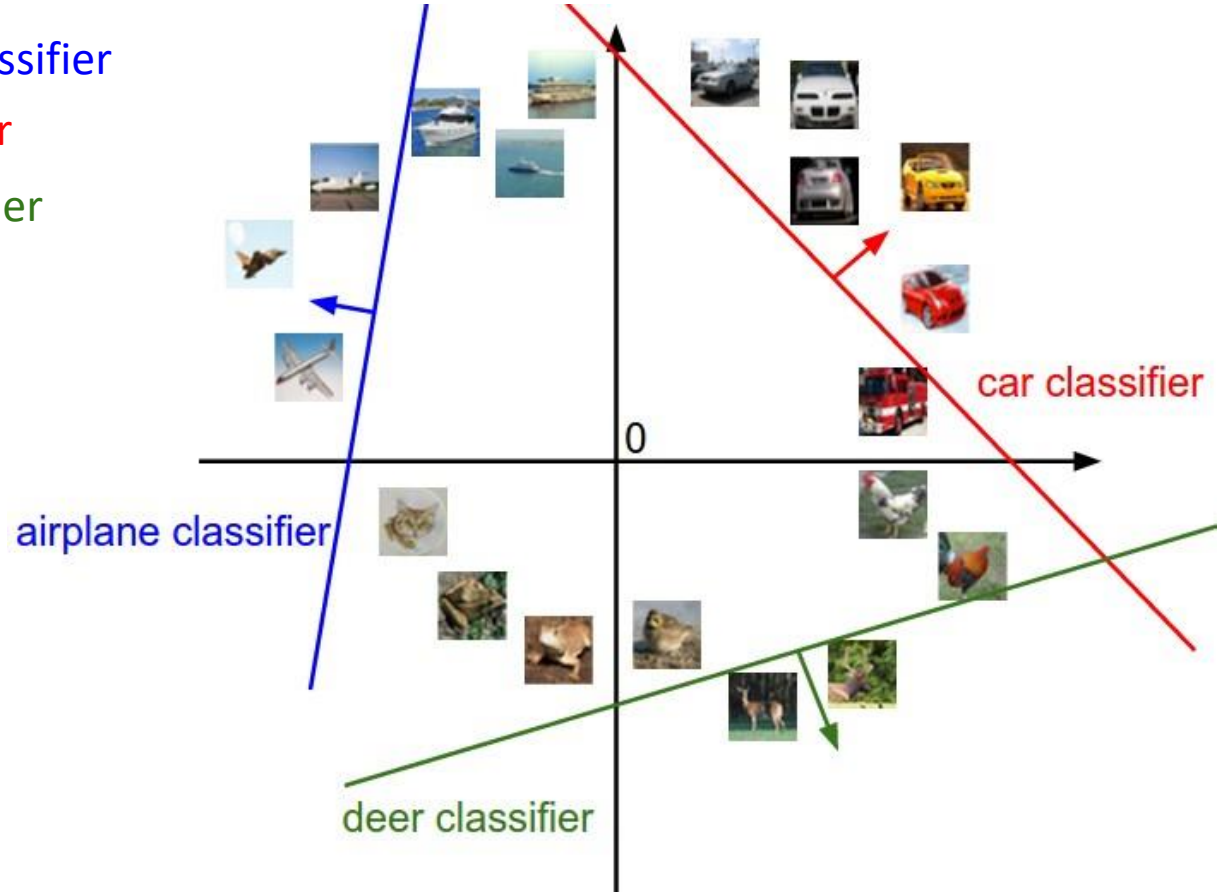
$$\boldsymbol{w} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,D} \\ w_{2,1} & w_{2,2} & \dots & w_{2,D} \\ w_{3,1} & w_{3,2} & \dots & w_{3,D} \end{bmatrix}$$

airplane classifier

car classifier

deer classifier

Každý řádek matice $\boldsymbol{w}$ je binární klasifikátor diskriminující třídu k od ostatních



obrázek: https://cs231n.github.io/linear-classify/

Geometrická interpretace

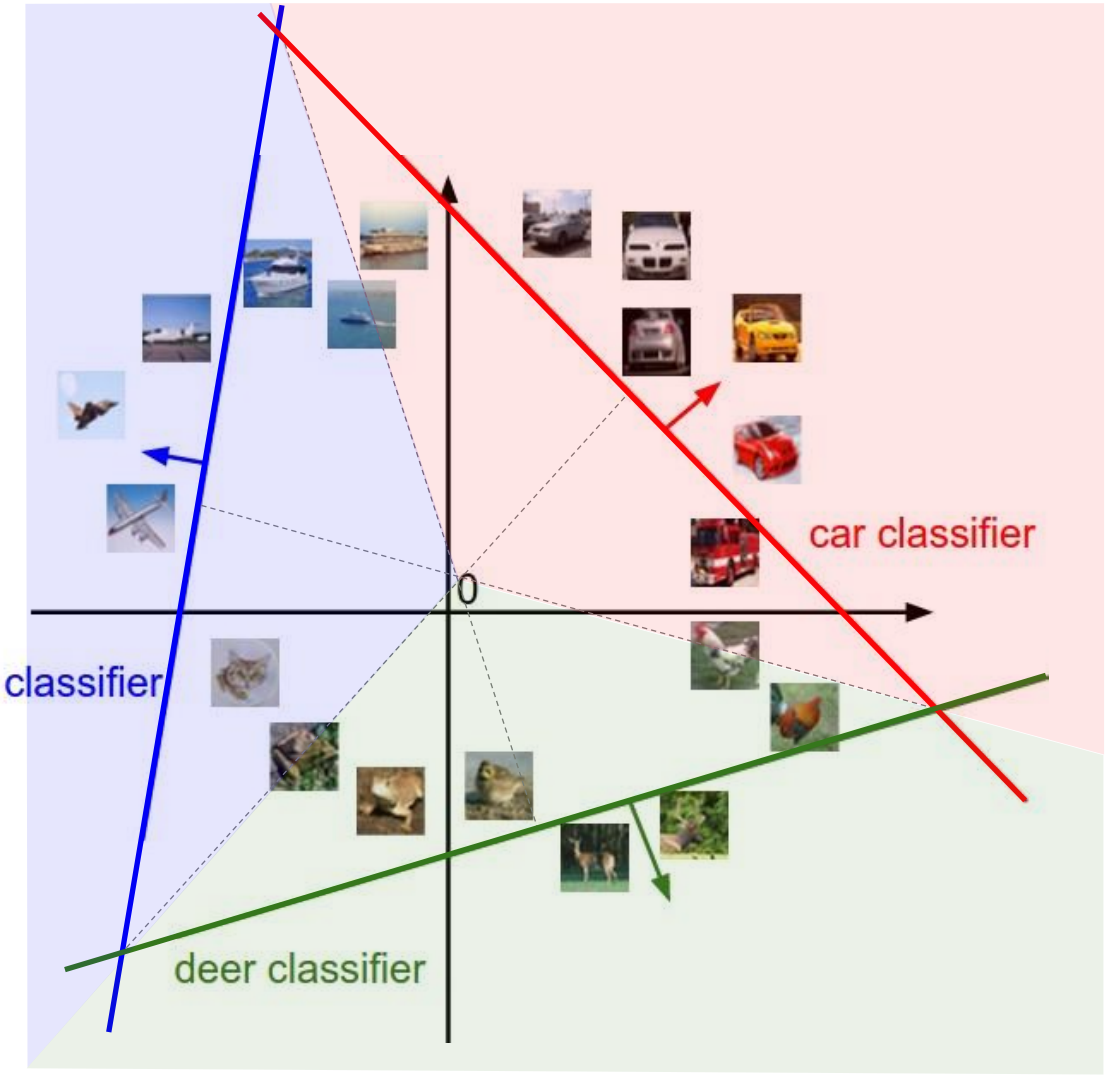
$$\boldsymbol{w} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,D} \\ w_{2,1} & w_{2,2} & \dots & w_{2,D} \\ w_{3,1} & w_{3,2} & \dots & w_{3,D} \end{bmatrix}$$

airplane classifier

car classifier

deer classifier

Každý řádek matice $\boldsymbol{w}$ je binární klasifikátor diskriminující třídu k od ostatních



obrázek: https://cs231n.github.io/linear-classify/

Návrh a trénování lineárního klasifkátoru

1. Navrhneme diskriminativní klasifikační funkci s upravitelnými parametry
2. **Kvantifikujeme její úspěšnost klasifikace nějakým kritériem**
3. Nastavíme parametry klasifikátoru tak, abychom optimalizovali zvolené kritérium

Klasifikační kritérium

Softmax cross entropy

Klasifikační kritérium

- Klasifikátor predikuje $\hat{y}_n \in \{1, \dots, K\}$

$$\hat{y}_n = \underset{k}{\operatorname{argmax}}\, \boldsymbol{s}_n$$

- Pro každý obrázek $\boldsymbol{x}_n$ přitom známe správnou odpověď $y_n \in \{1, \dots, K\}$ (target)
- Porovnáním $\hat{y}_n$ vs y_n (nebo vs $\boldsymbol{s}_n$) můžeme vyčíslit, jak dobrá/špatná predikce je
- Celkem máme **trénovací dataset** $\boldsymbol{X}$ s N obrázky a tedy i páry $(\boldsymbol{x}_n, y_n)$
- Nakonec tedy můžeme spočítat tzv. loss

$$L(\boldsymbol{X}) = \frac{1}{N} \sum_{n=1}^{N} L_n(\boldsymbol{s}_n, y_n)$$

Multiclass cross entropy

- Logistická regrese definuje loss jako tzv. **křížovou entropii**

$$l_n = -\sum_{k=1}^{K} p_{n,k} \cdot \log\left(\hat{p}_{n,k}\right)$$

← Pro jeden obrázek

- kde

$\boldsymbol{p}_n = \left[p_{n,1}, \ldots, p_{n,K}\right]^\top$ … cílové rozdělení (ground truth / target)

$\hat{\boldsymbol{p}}_n = \left[\hat{p}_{n,1}, \ldots, \hat{p}_{n,K}\right]^\top$ … výstupní pravd. (predikce) klasifikátoru

jsou vektory, na které nahlížíme jako na diskrétní pravděpodobnostní rozdělení

→ cross entropy = rozdíl mezi dvěma pravděpodobnostními rozděleními

Multiclass cross entropy

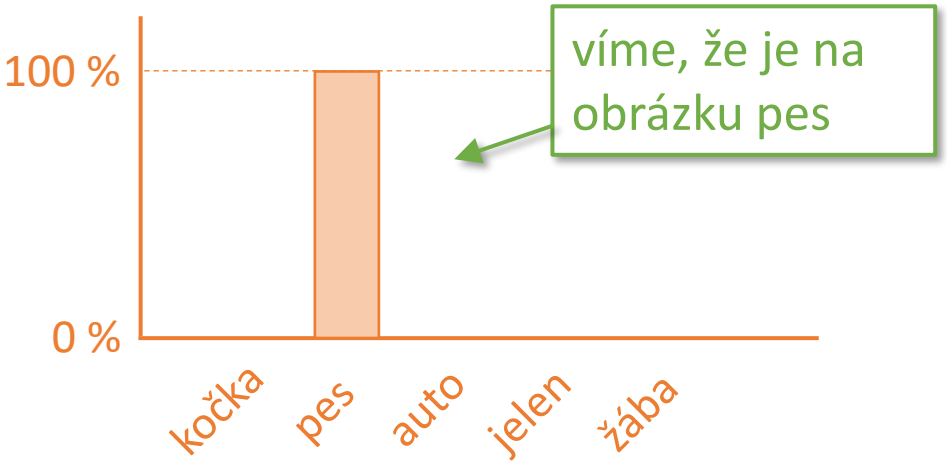
$$l_n = -\sum_{k=1}^{K} p_{n,k} \cdot \log(\hat{p}_{n,k})$$

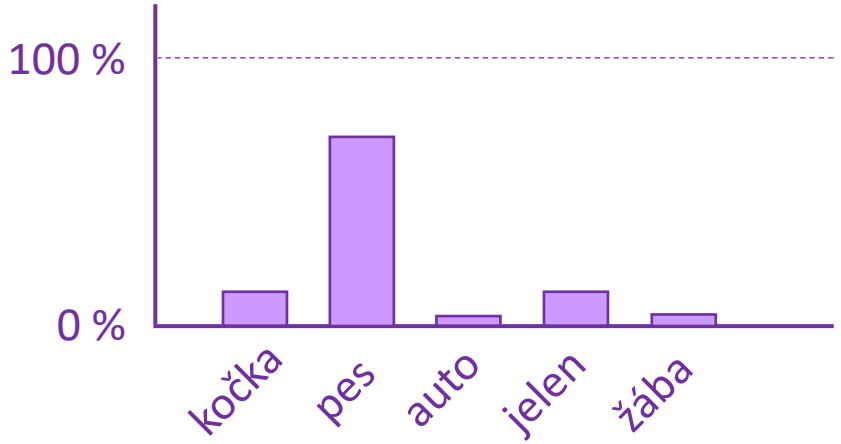
$\boldsymbol{p}_n = [p_{n,1}, \cdots, p_{n,K}]^\top$

cílové rozdělení (ground truth / target)

$\hat{\boldsymbol{p}}_n = [\hat{p}_{n,1}, \cdots, \hat{p}_{n,K}]^\top$

výstup (predikce) klasifikátoru





Převod číselného označení třídy na rozdělení: one hot encoding

- Značka y_n pro každý obrázek je celé číslo, tj. $y_n \in \{1, \dots, K\}$
- Pokud počet tříd $K = 5$ → požadované rozdělení je

$$y_n = 2 \quad \Rightarrow \quad \boldsymbol{p}_n = [0,1,0,0,0]^\top$$

$$y_n = 5 \quad \Rightarrow \quad \boldsymbol{p}_n = [0,0,0,0,1]^\top$$

Převod výstupních skóre modelu na rozdělení: **softmax**

- Normalizuje vektor skóre $\boldsymbol{s}_n$ tak, že výstup lze interpretovat jako pravděpodobnosti
- Pravděpodobnost, že na obrázku $\boldsymbol{x}_n$ je objekt třídy k definuje jako

$$\hat{p}_{n,k} = P(\text{třída } k|\boldsymbol{x}_n) = \frac{e^{s_{n,k}}}{\sum_{i=1}^{K} e^{s_{n,i}}}$$

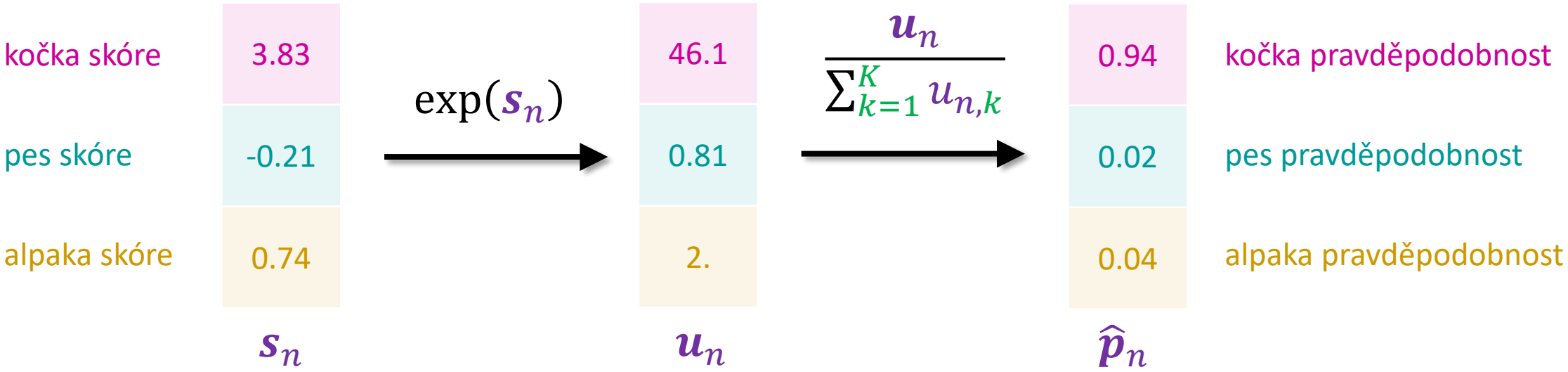
- Výstupem K-dimezionální vektor $\hat{\boldsymbol{p}}_n$ pravděpodobností jednotlivých tříd

$$\hat{\boldsymbol{p}}_n = \left[\hat{p}_{n,1}, \ldots, \hat{p}_{n,K}\right]^{\mathsf{T}}, \qquad 0 \leq \hat{p}_{n,k} \leq 1, \qquad \sum_{k=1}^{K} \hat{p}_{n,k} = 1$$

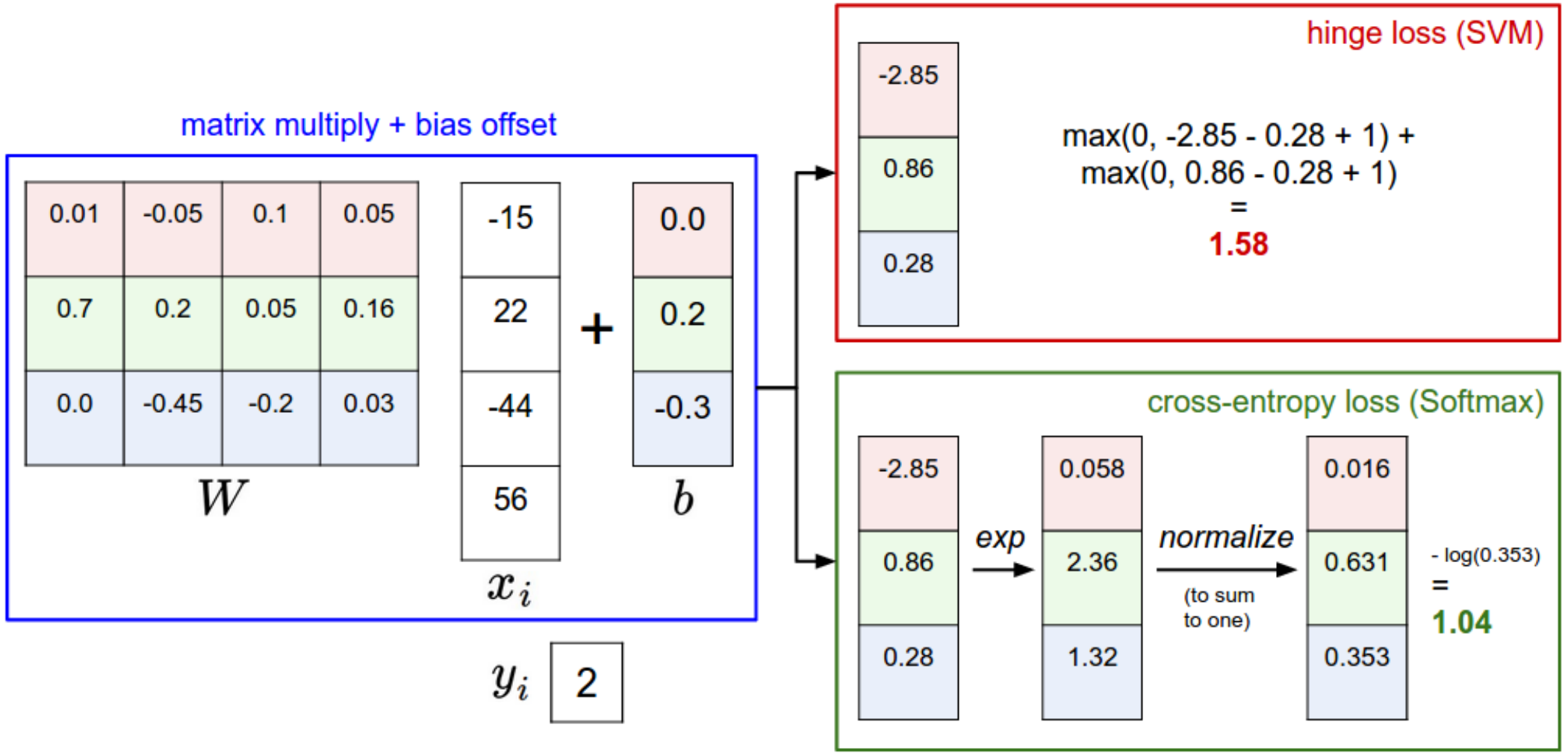
- Chová se jako “měkké” maximum: exponenciováním se zvýrazní rozdíly (nejvyšší hodnota vynikne), až teprve pak se normalizuje (ostatní jsou staženy k nule)

Softmax: příklad

$$\hat{p}_{n,k} = \frac{e^{s_{n,k}}}{\sum_{i=1}^{K} e^{s_{n,i}}}$$



Lineární model a klasifikační loss



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

Návrh a trénování lineárního klasifkátoru

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

Celkový loss

- Chybovost klasifikátoru (loss)

$$L(\boldsymbol{X}) = \frac{1}{N}\sum_{n=1}^{N} L_n(\boldsymbol{s}_n, y_n)$$

- Je funkce, která kromě dat závisí na parametrech kasifikátoru, protože

$$\boldsymbol{s}_n = \boldsymbol{w} \cdot \boldsymbol{x}_n + \boldsymbol{b}$$

- Celkově tedy vyhodnocujeme funkci $L(\boldsymbol{X}, \boldsymbol{\theta})$ závislou na datech $\boldsymbol{X}$ a parametrech $\boldsymbol{\theta}$

Strojové učení je optimalizace

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{X}, \boldsymbol{\theta})$$

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.

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

Metoda největšího spádu (Gradient Descent, GD)

- Metoda největšího spádu využívá derivaci pro zjištění, kterým směrem funkce roste

1. V aktuální pozici x spočítá derivaci $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$ (exaktně nebo numericky)

2. Vylepší aktuální odhad x posunutím se ve směru opačném ke směru růstu, formálně

$$x' := x - \gamma \cdot \frac{\mathrm{d}f(x)}{\mathrm{d}x}$$

   Výsledná velikost rozdílu mezi původním odhadem a novým odhadem je ovlivněna
   a) velikostí kroku γ, kterou se výsledek derivace škáluje
   b) absolutní hodnotou derivace $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$, 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

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

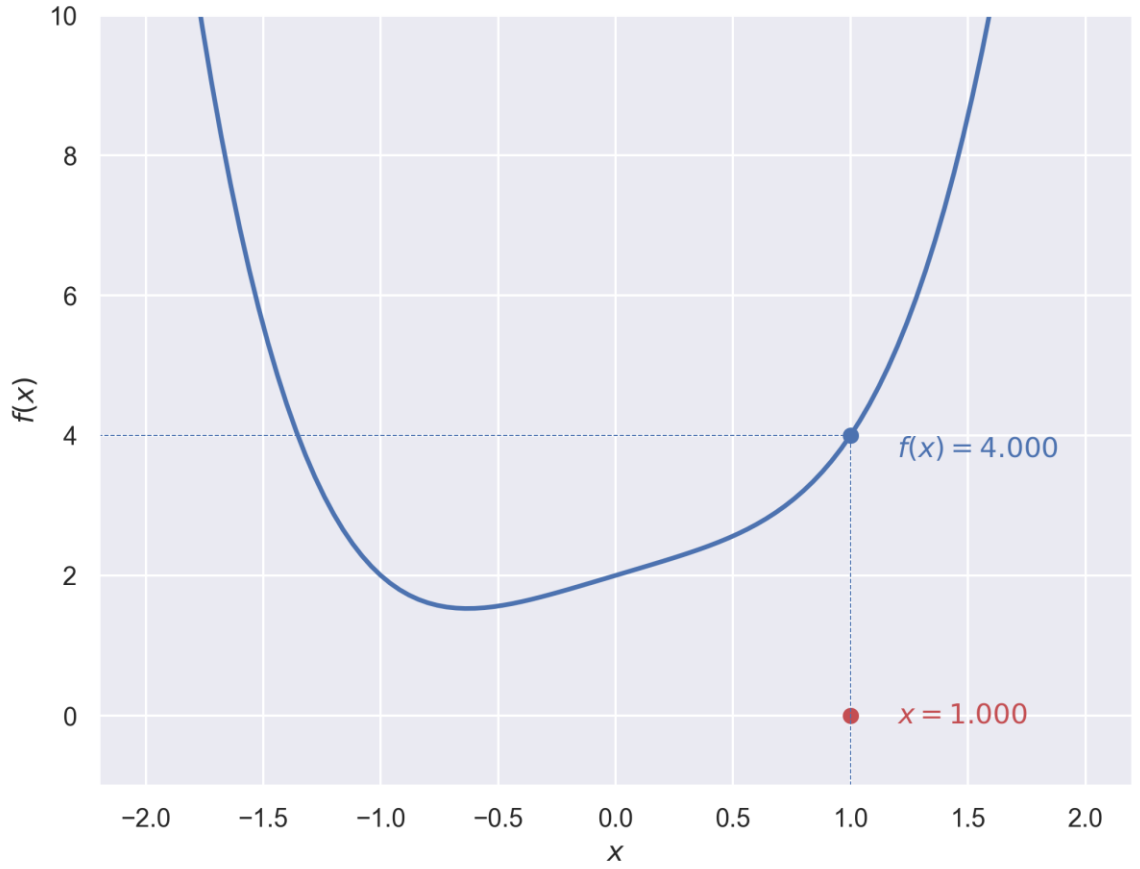
- Hledáme pozici x^* jejího minima, tj.

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

- Jako počáteční odhad minima zkusíme

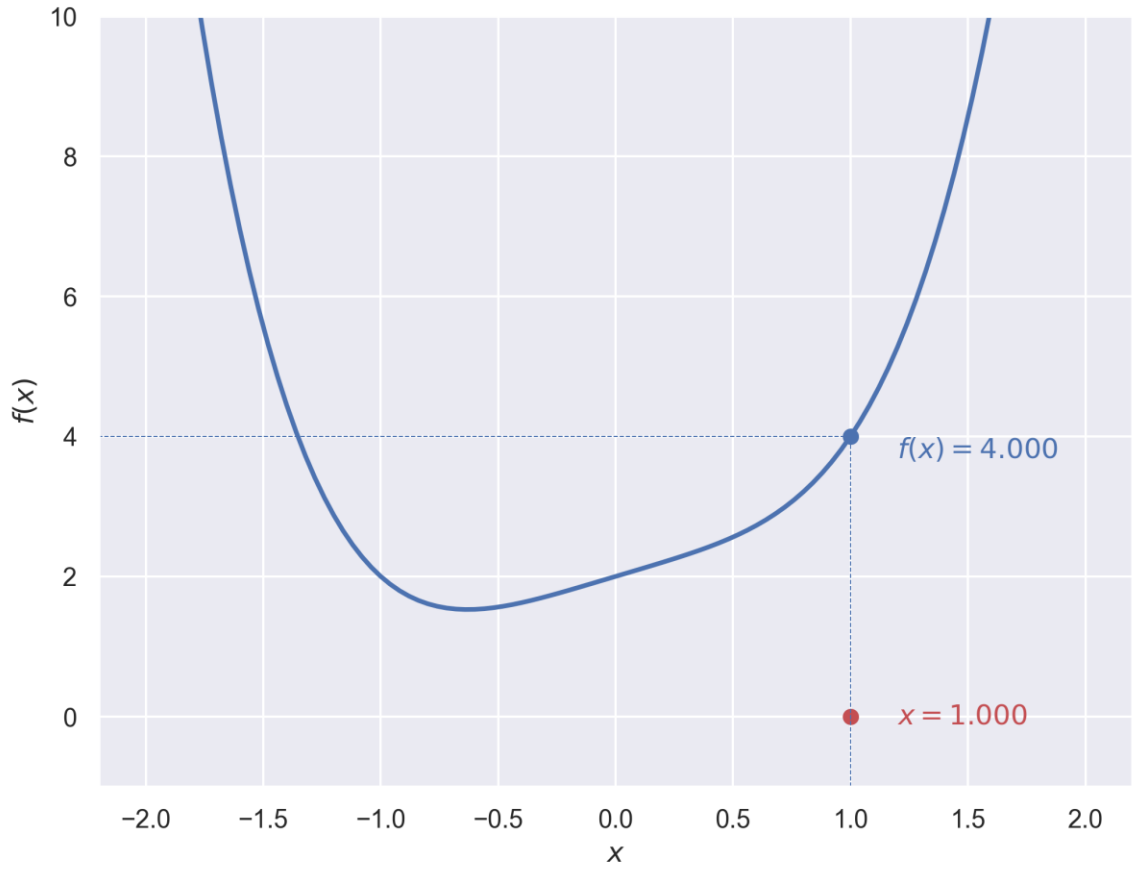
$$x^{(0)} = 1$$

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



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

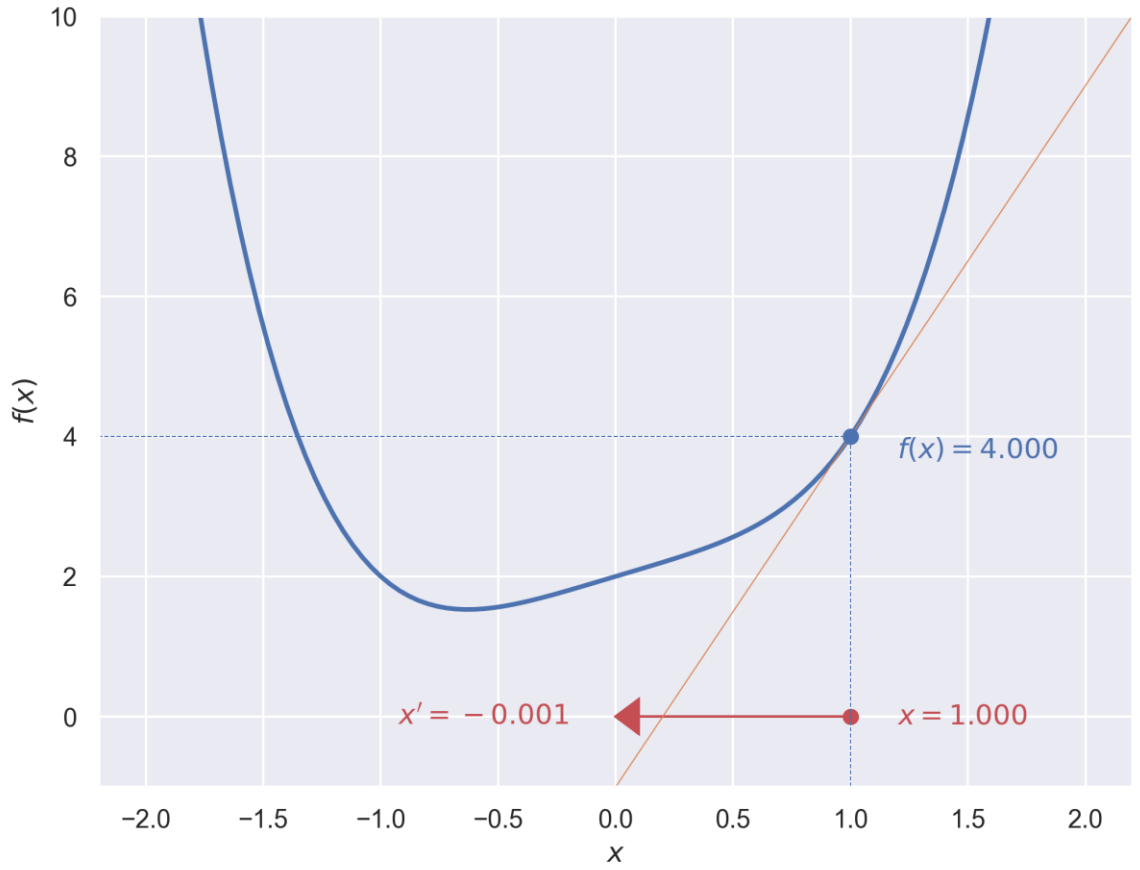
t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
0	1.000	4.000	5.006

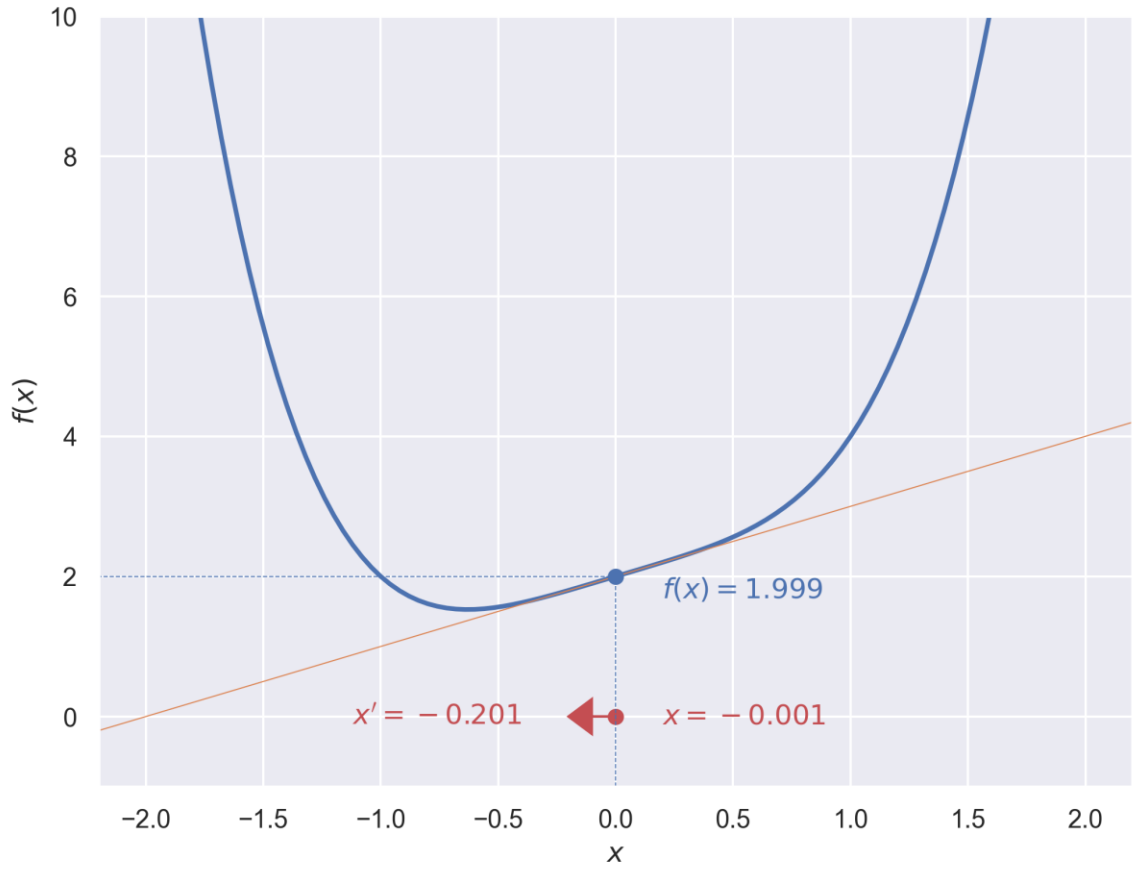
$$
\begin{aligned}
x &= 1.000 \\
f(x) &= 1.000^4 + 1.000 + 2 = 4.000 \\
\frac{\mathrm{d}f(x)}{\mathrm{d}x} &= \frac{f(1.000 + 0.001) - f(1.000)}{0.001} \\
&= \frac{4.005 - 4.000}{0.001} \\
&= 5.006 \\
x' &= 1.000 - 0.2 \cdot 5.006 \\
&= -0.001
\end{aligned}
$$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
0	1.000	4.000	5.006
1	-0.001	1.999	1.000

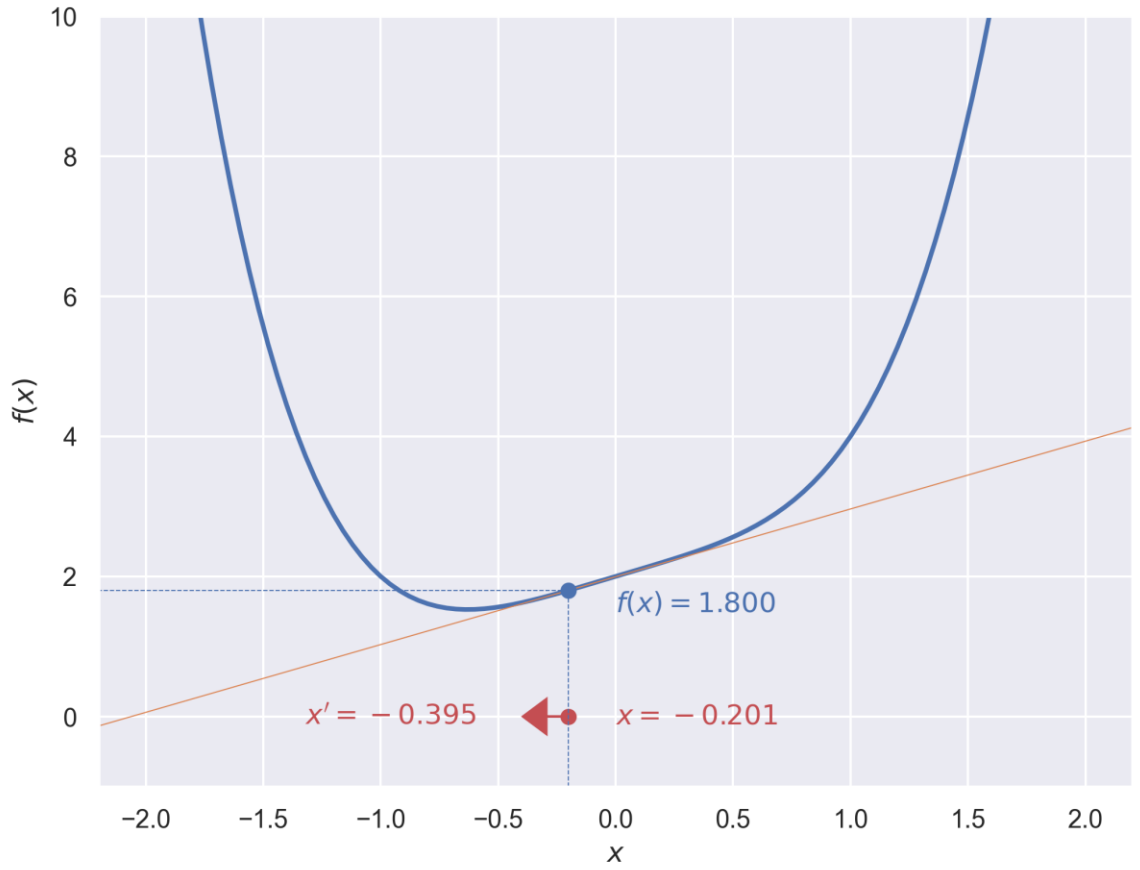
$$
\begin{aligned}
x &= -0.001 \\
f(x) &= (-0.001)^4 + (-0.001) + 2 = 1.999 \\
\frac{\mathrm{d}f(x)}{\mathrm{d}x} &= \frac{f(-0.001 + 0.001) - f(-0.001)}{0.001} \\
&= \frac{2.000 - 1.999}{0.001} \\
&= 1.000 \\
x' &= -0.001 - 0.2 \cdot 1.000 \\
&= -0.201
\end{aligned}
$$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
0	1.000	4.000	5.006
1	-0.001	1.999	1.00
2	-0.201	1.800	0.968

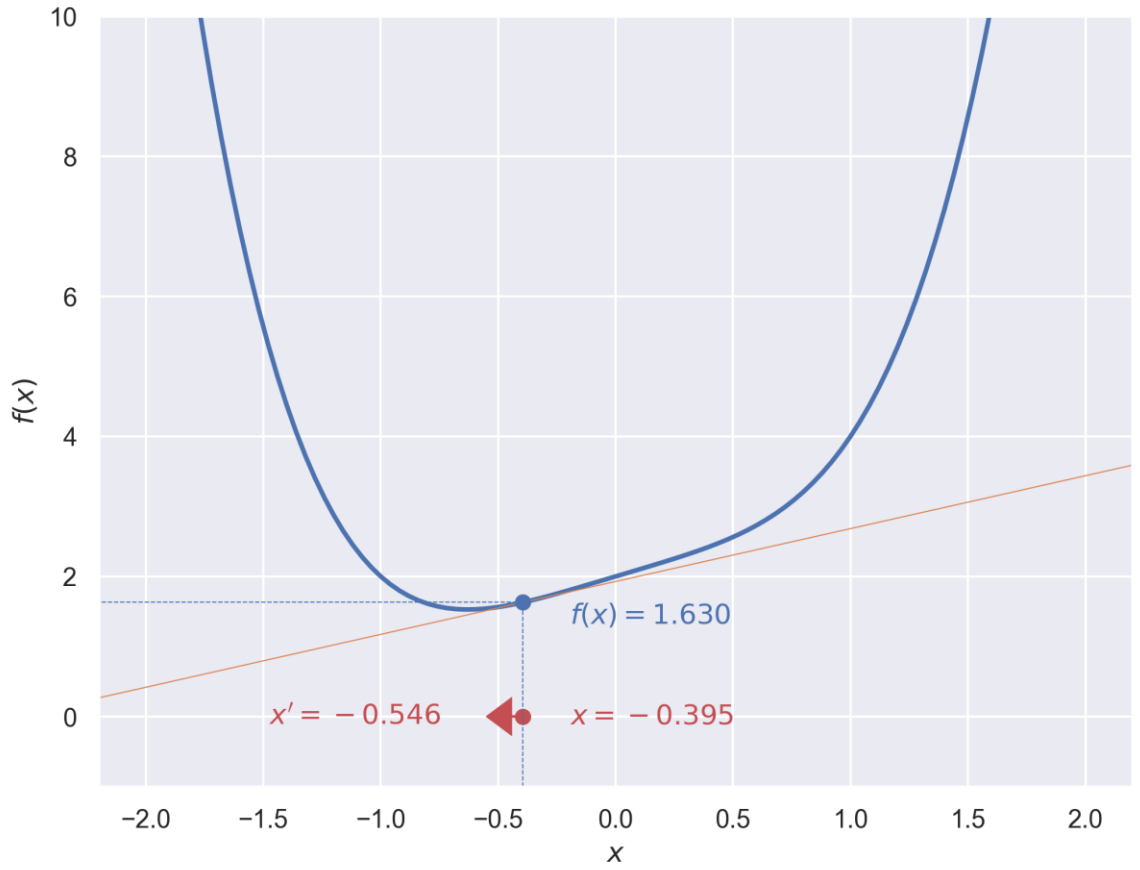
$$
\begin{aligned}
x &= -0.201 \\
f(x) &= (-0.201)^4 + (-0.201) + 2 = 1.800 \\
\frac{\mathrm{d}f(x)}{\mathrm{d}x} &= \frac{f(-0.201 + 0.001) - f(-0.201)}{0.001} \\
&= \frac{1.801 - 1.800}{0.001} \\
&= 0.968 \\
x' &= -0.201 - 0.2 \cdot 0.968 \\
&= -0.395
\end{aligned}
$$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
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

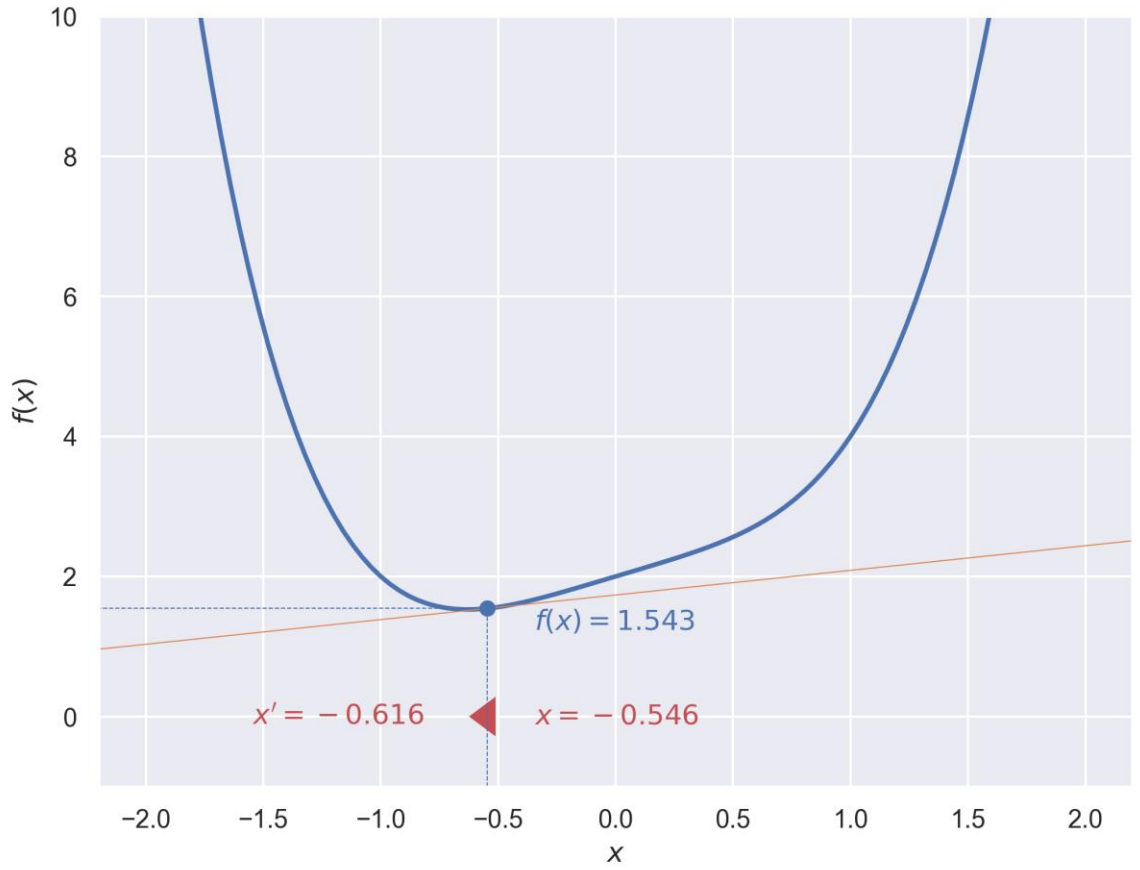
$$x = -0.395$$
$$f(x) = (-0.395)^4 + (-0.395) + 2 = 1.630$$
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \frac{f(-0.395 + 0.001) - f(-0.395)}{0.001}$$
$$= \frac{1.630 - 1.630}{0.001}$$
$$= 0.755$$
$$x' = -0.395 - 0.2 \cdot 0.755$$
$$= -0.546$$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
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

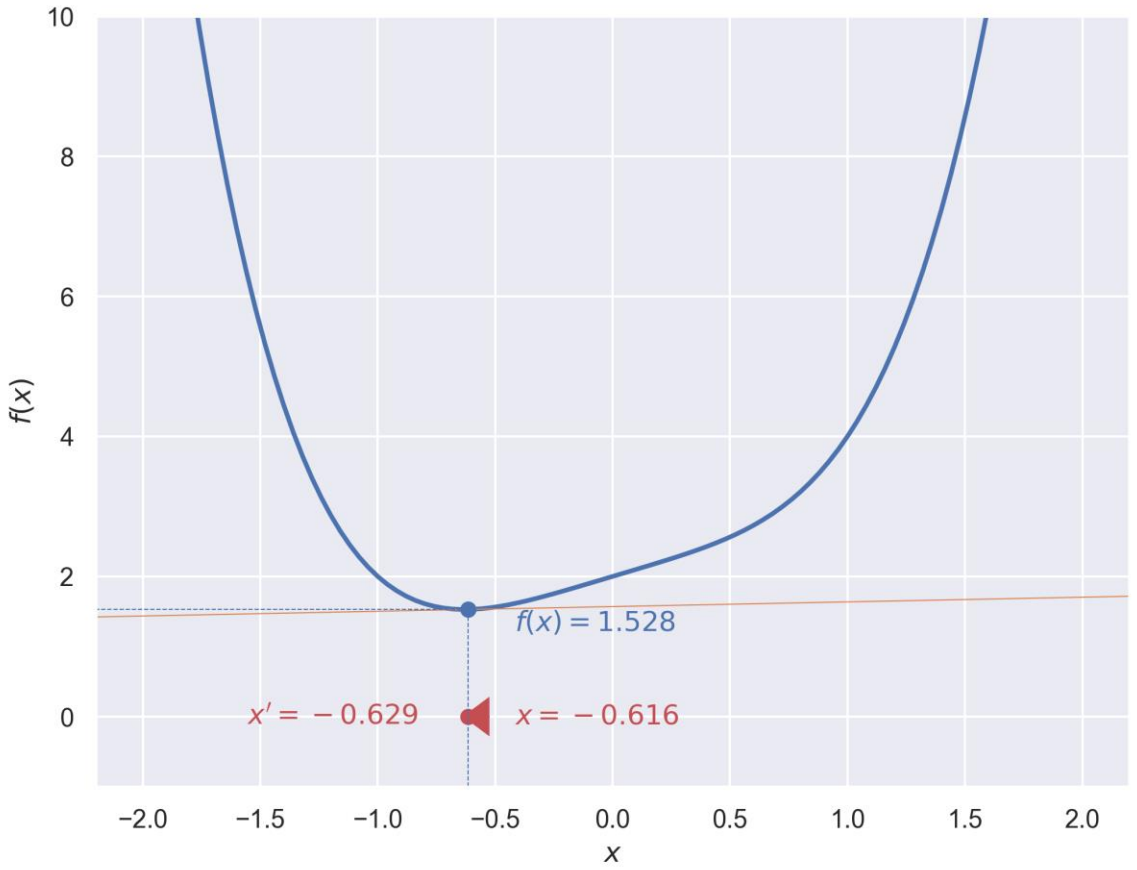
$$x = -0.546$$
$$f(x) = (-0.546)^4 + (-0.546) + 2 = 1.543$$
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \frac{f(-0.546 + 0.001) - f(-0.546)}{0.001}$$
$$= \frac{1.543 - 1.543}{0.001}$$
$$= 0.352$$
$$x' = -0.546 - 0.2 \cdot 0.352$$
$$= -0.616$$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
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

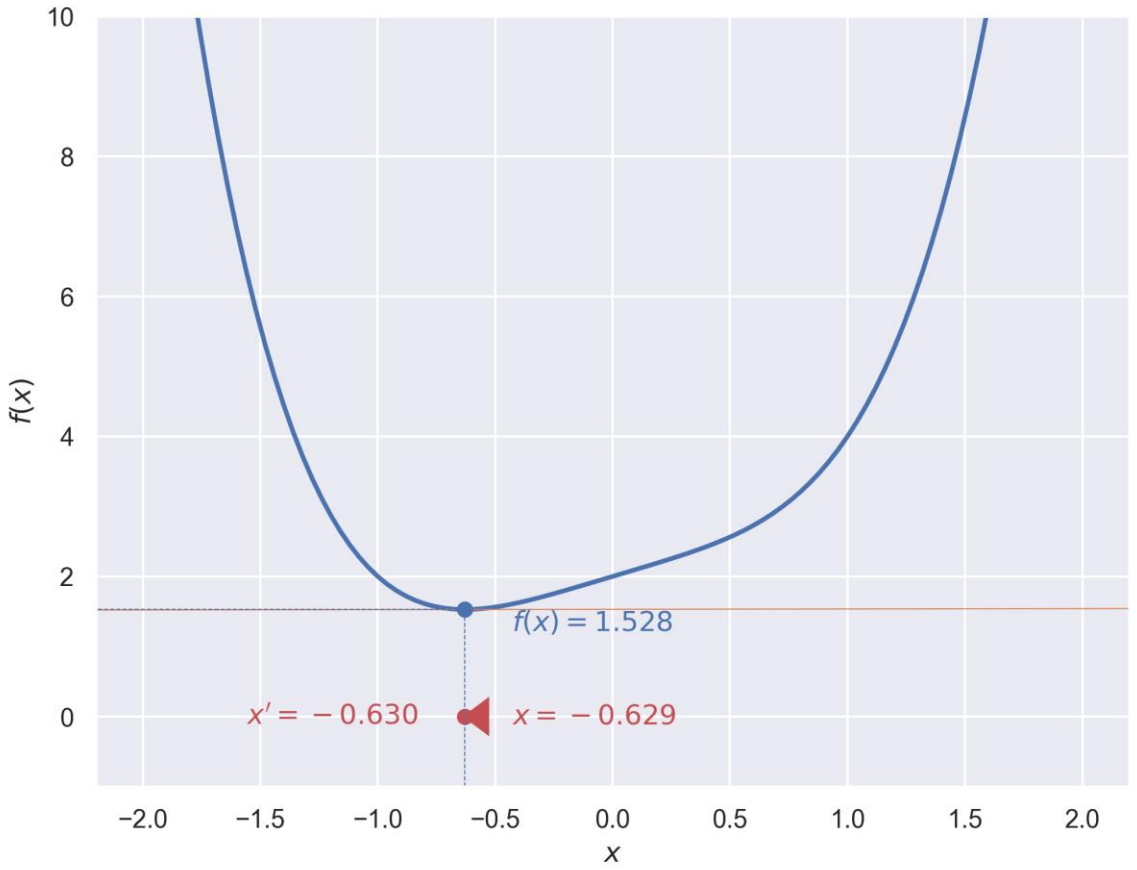
$$
\begin{aligned}
x &= -0.616 \\
f(x) &= (-0.616)^4 + (-0.616) + 2 = 1.528 \\
\frac{\mathrm{d}f(x)}{\mathrm{d}x} &= \frac{f(-0.616 + 0.001) - f(-0.616)}{0.001} \\
&= \frac{1.528 - 1.528}{0.001} \\
&= 0.067 \\
x' &= -0.616 - 0.2 \cdot 0.067 \\
&= -0.629
\end{aligned}
$$



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
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

$$x = -0.629$$
$$f(x) = (-0.629)^4 + (-0.629) + 2 = 1.528$$
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \frac{f(-0.629 + 0.001) - f(-0.629)}{0.001}$$
$$= \frac{1.528 - 1.528}{0.001}$$
$$= 0.005$$
$$x' = -0.629 - 0.2 \cdot 0.005$$
$$= -0.630$$



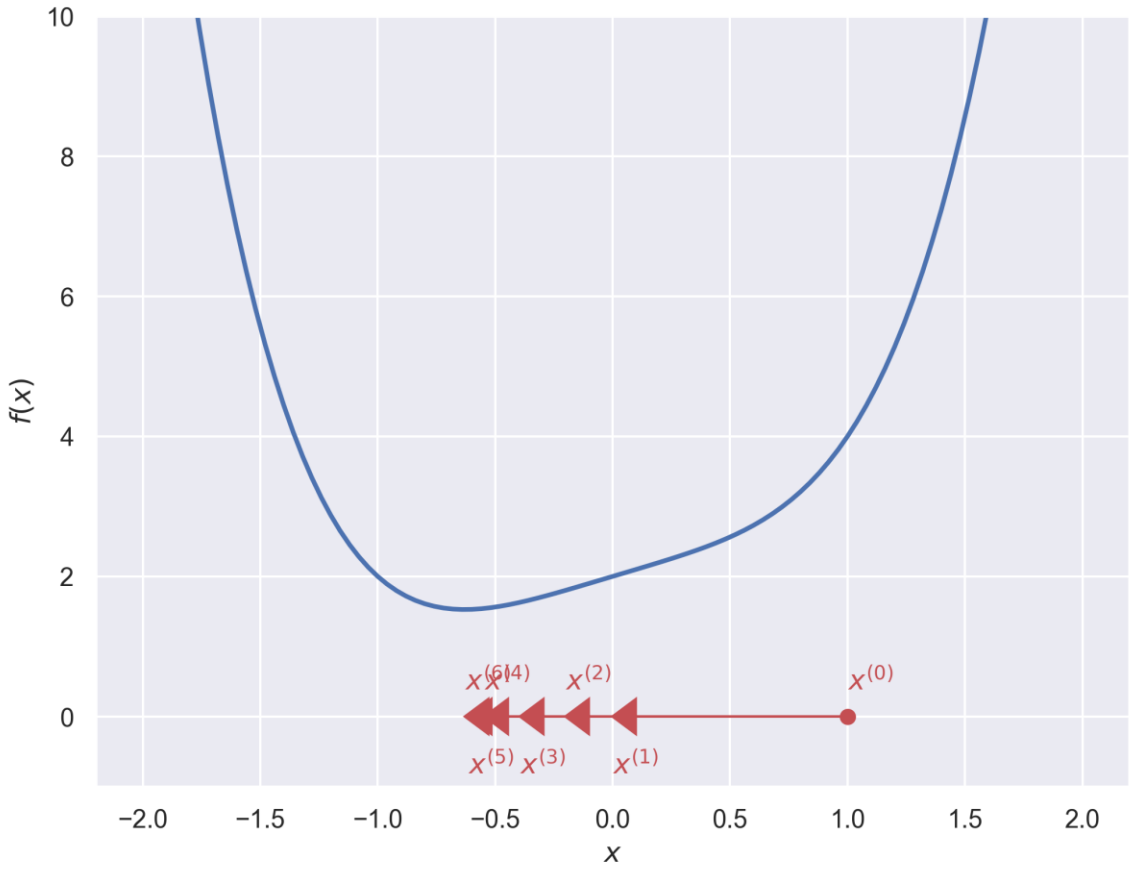
Optimální řešení dle
https://www.wolframalpha.com/

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

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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
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

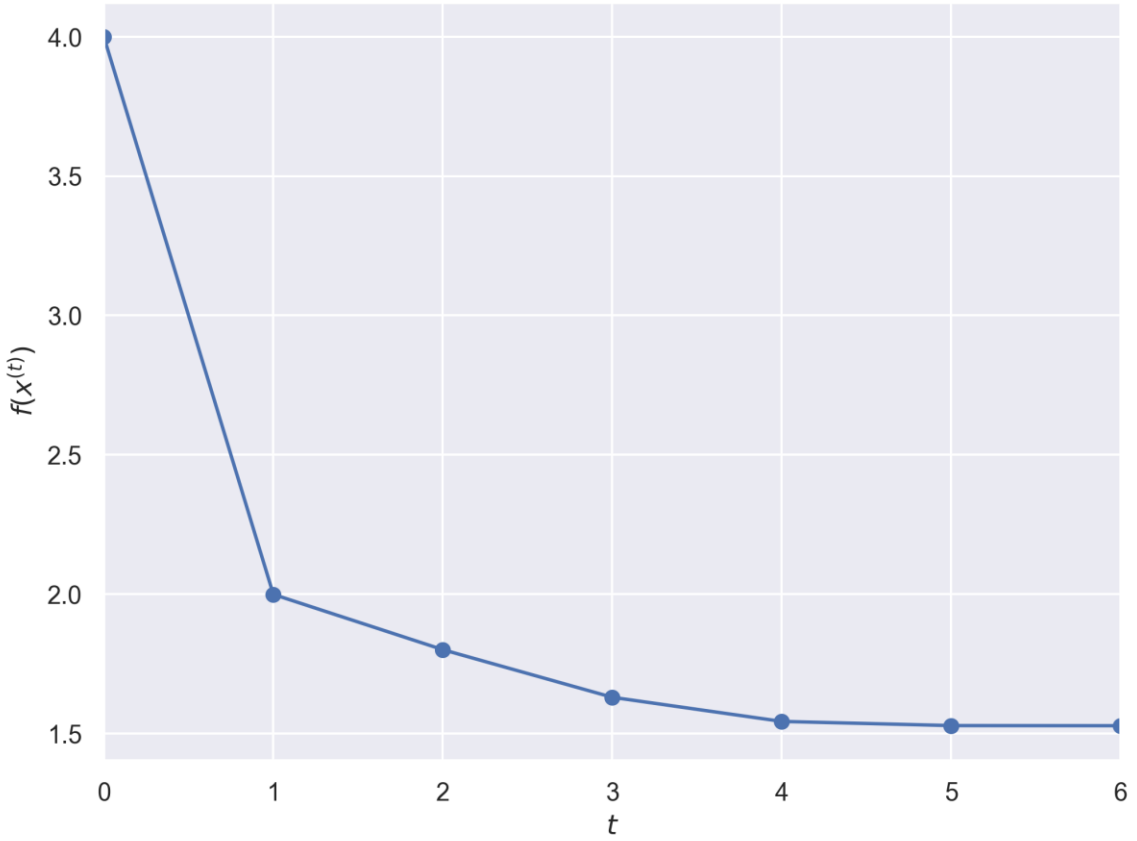
Vizualizace optimálního odhadu v čase



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

t	x	$f(x)$	$\mathrm{d}f/\mathrm{d}x$
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 $\boldsymbol{x}$ jako vektor obecně s rozměrem D
- Vrací přitom skalární hodnotu (chybovost modelu na datasetu jako jediné číslo)

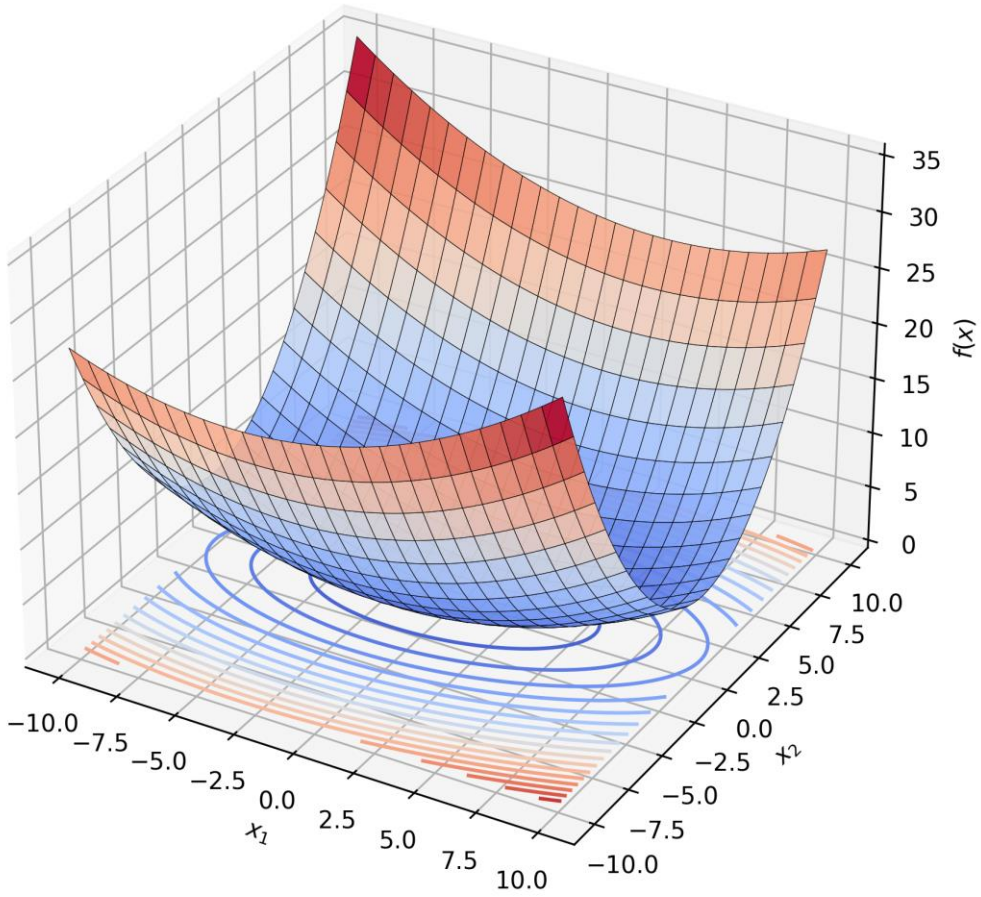
- Potřebujeme tedy minimalizovat

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

kde

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

$$\boldsymbol{x} \in \mathbb{R}^D$$



Derivace funkce: *n*D vstup, 1D výstup

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

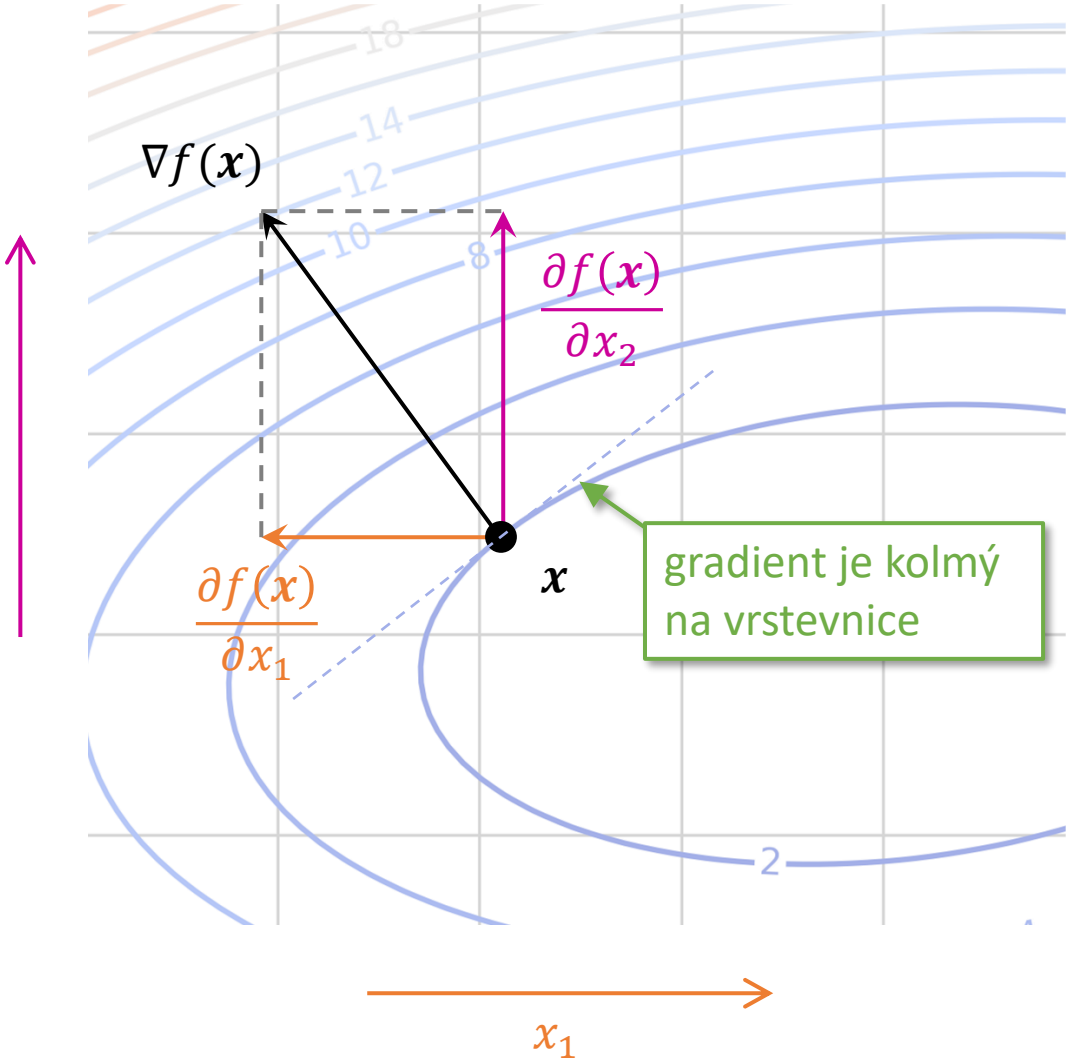
$$f(\boldsymbol{x}) = f(x_1, \ldots, x_D)$$

tedy funkcí s D vstupy a 1 výstupem

můžeme derivovat vzhledem ke každému z jednotlivých x_d → **parciální derivace**

- Uspořádání všech D paricálních derivací do vektoru se nazývá **gradient**:

$$\nabla f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_D}\right]^\top$$



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:

$$\boldsymbol{x}' \coloneqq \boldsymbol{x} - \gamma \cdot \nabla f(\boldsymbol{x})$$

kde

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

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

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

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

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

kde

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

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

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

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

Příklad GD: lineární softmax cross entropy na CIFAR-10

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

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

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

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{\theta})$$

- Data tedy považujeme za konstantu

- Jako počáteční odhad minima zkusíme

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

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

Gradient lineárního softmaxu a křížové entropie analyticky

Optimalizovaná funkce (loss) je

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

Potřebujeme

$$\frac{\partial L(\boldsymbol{w}, \boldsymbol{b})}{\partial \boldsymbol{w}} = ?$$

$$\frac{\partial L(\boldsymbol{w}, \boldsymbol{b})}{\partial \boldsymbol{b}} = ?$$

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

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

Gradient descent (GD) pro multiclass logistickou regresi

Incializujeme:

$$w_{kd} \sim \mathcal{N}(0,0.001)$$
$$b_k \sim \mathcal{N}(0,0.001)$$

Opakujeme:

$$\hat{\boldsymbol{p}}_n \coloneqq \frac{\exp\left(\boldsymbol{w}_{y_n,:} \cdot \boldsymbol{x}_n + b_{y_n}\right)}{\sum_{k=1}^{K} \exp\left(\boldsymbol{w}_{k,:} \cdot \boldsymbol{x}_n + b_k\right)}$$

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

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

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

$$\boldsymbol{w} \coloneqq \boldsymbol{w} - \gamma \cdot \frac{\partial L(\boldsymbol{w}, \boldsymbol{b})}{\partial \boldsymbol{w}}$$

$$\boldsymbol{b} \coloneqq \boldsymbol{b} - \gamma \cdot \frac{\partial L(\boldsymbol{w}, \boldsymbol{b})}{\partial \boldsymbol{b}}$$

```
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:

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

Opakujeme:

1. Pro každý vzorek $\boldsymbol{x}_n$ v trénovací sadě $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
   a. predikujeme pravděpodobnosti $\hat{\boldsymbol{p}}_n$
   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 $\boldsymbol{\theta}$ akumulovaným gradientem ∇L s krokem γ

Zastavíme:

- po fixním počtu iterací
- parametry $\boldsymbol{\theta}$ se ustálí
- hodnota kritéria $J(\boldsymbol{\theta})$ 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:

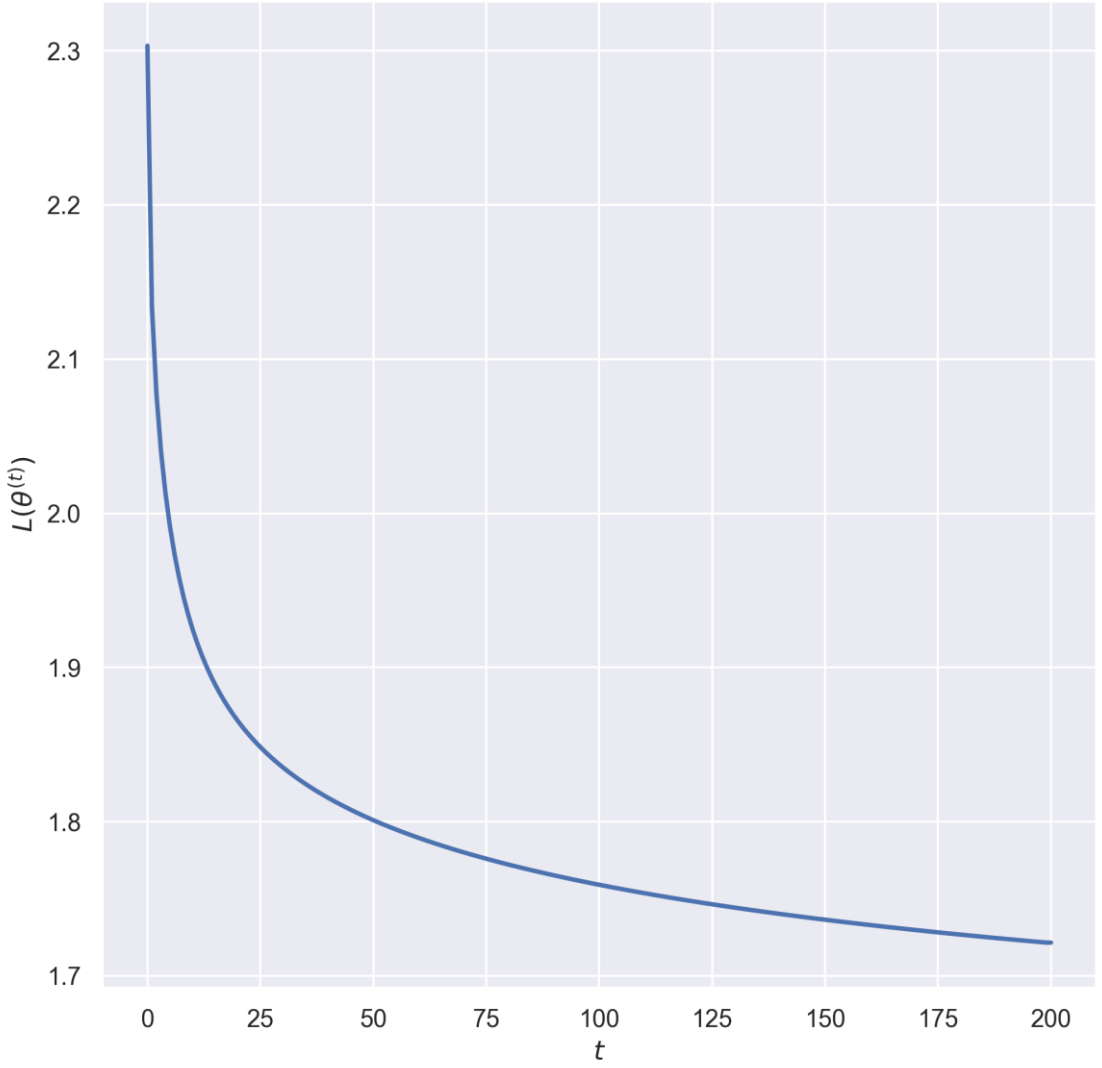
- $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$ na náhodné hodnoty

Opakujeme:

1. Pro každý vzorek $\boldsymbol{x}_n$ v trénovací sadě $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
   a. predikujeme pravděpodobnosti $\hat{\boldsymbol{p}}_n$
   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 $\boldsymbol{\theta}$ akumulovaným gradientem ∇L s krokem γ

Zastavíme:

- po fixním počtu iterací
- parametry $\boldsymbol{\theta}$ se ustálí
- hodnota kritéria $J(\boldsymbol{\theta})$ již delší dobu neklesá



Gradient descent (GD) pro multiclass logistickou regresi

Incializujeme:

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

jak budeme inicializovat?

Opakujeme:

1. Pro každý vzorek $\boldsymbol{x}_n$ v trénovací sadě $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
   a. predikujeme pravděpodobnosti $\hat{\boldsymbol{p}}_n$
   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 $\boldsymbol{\theta}$ akumulovaným gradientem ∇L s krokem γ

hyperparametry

jaký krok učení?

Zastavíme:

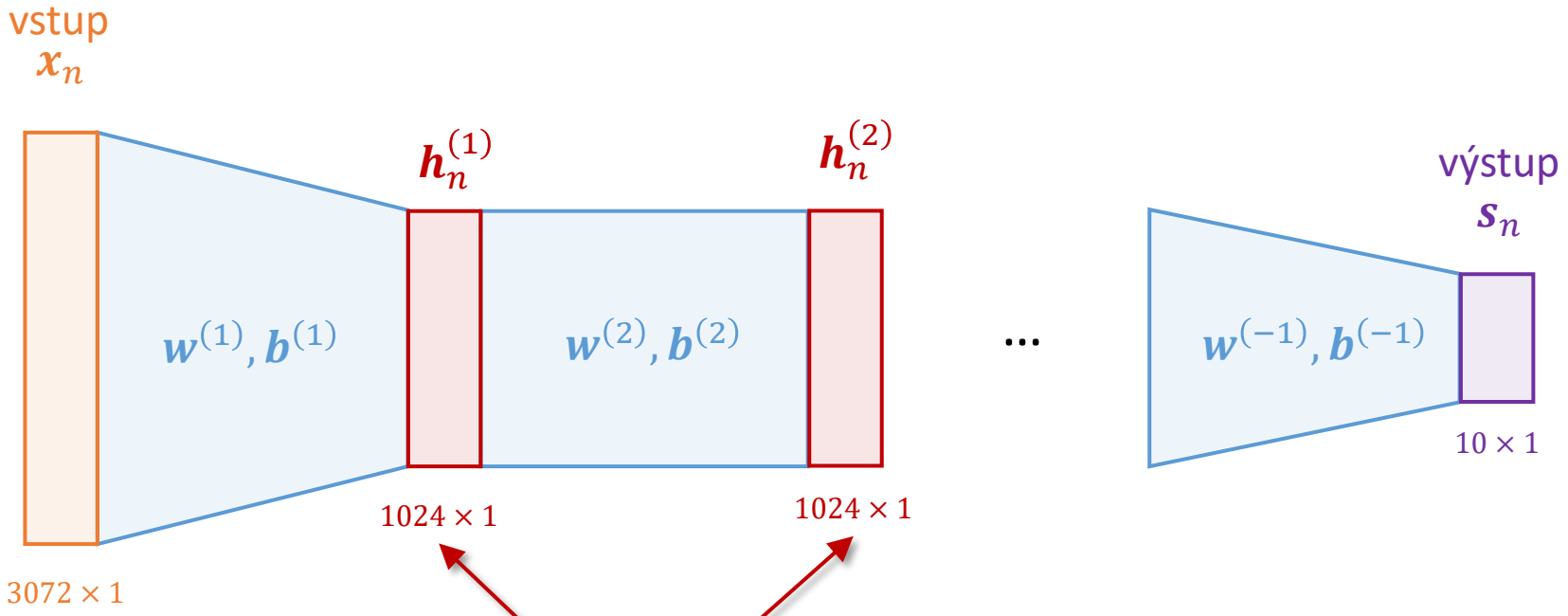
- po fixním počtu iterací
- parametry $\boldsymbol{\theta}$ se ustálí
- hodnota kritéria $J(\boldsymbol{\theta})$ již delší dobu neklesá

kolik iterací? jak dlouho?

Vícevrstvý perceptron

Vícevrstvý percpetron (Multi-Layer Perceptron, MLP)

- Označuje se také jako dopředná síť (Feed-Forward Network, FFN)
- Opakují se především dva+ typy vrstev: plně propojená a aktivace



Bloky sestávající z plně propojené vrstvy a aktivace se obvykle označují jako vrstvy a jejich výstup jako tzv. skrytá vrstva (h jako hidden)

Manuální výpočet gradientu?

Optimalizovaná funkce (loss) je

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \log \frac{\exp(f(\boldsymbol{x}_n, \boldsymbol{\theta})_{y_n})}{\sum_{k=1}^{K} \exp(f(\boldsymbol{x}_n, \boldsymbol{\theta})_k)}$$

Co když $f(\boldsymbol{x}_n, \boldsymbol{\theta})$ potažmo $L(\boldsymbol{\theta})$ jsou složité funkce jako např. hluboké neuronové sítě?

Potřebujeme

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = ?$$

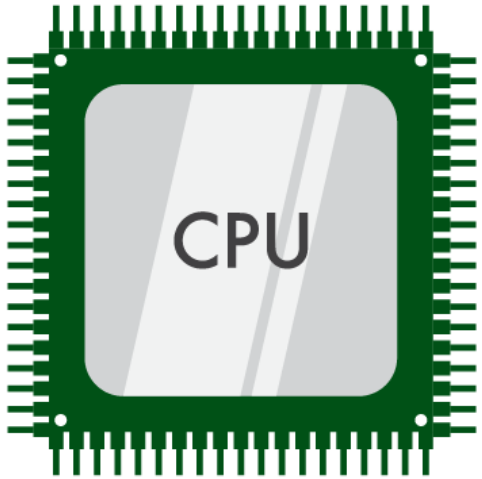
Automatický výpočet gradientu = zpětná propagace

Optimalizovaná funkce (loss) je

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \log \frac{\exp\left(f(\boldsymbol{x}_n, \boldsymbol{\theta})_{y_n}\right)}{\sum_{k=1}^{K} \exp(f(\boldsymbol{x}_n, \boldsymbol{\theta})_k)}$$

Potřebujeme

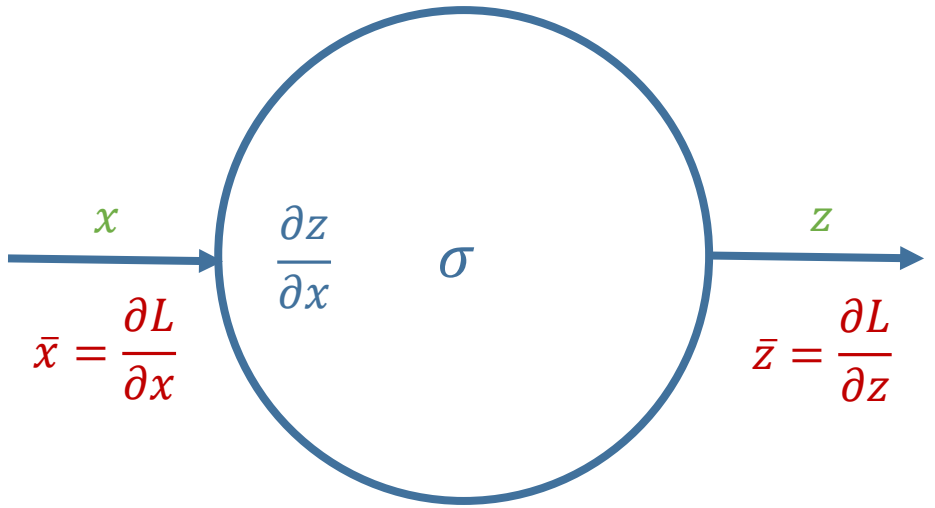
$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = ?$$



$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \ldots$$

Funkce jako uzel ve výpočetním grafu

```python
class Sigmoid(Function):

    @staticmethod
    def forward(x: float):  # zatim pouze skalary
        z = 1 / (1 - math.exp(-x))
        cache = z,
        return z, cache

    @staticmethod
    def backward(dz: float, cache: tuple):
        z, = cache
        dx = dz * z * (1 - z)  # retizkove pravidlo
        return dx
```



zpětný průchod je řetízkové pravidlo

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}$$

Každá funkce, kterou chceme použít jako stavební blok, musí mít definovaný

- dopředný průchod
- zpětný průchod

Dvouvrstvý perceptron v numpy na 11 řádků

```
X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
y = np.array([[0,1,1,0]]).T
syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,1)) - 1
for j in range(60000):
    l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
    l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
    l2_delta = (y - l2)*(l2*(1-l2))
    l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
    syn1 += l1.T.dot(l2_delta)
    syn0 += X.T.dot(l1_delta)
```

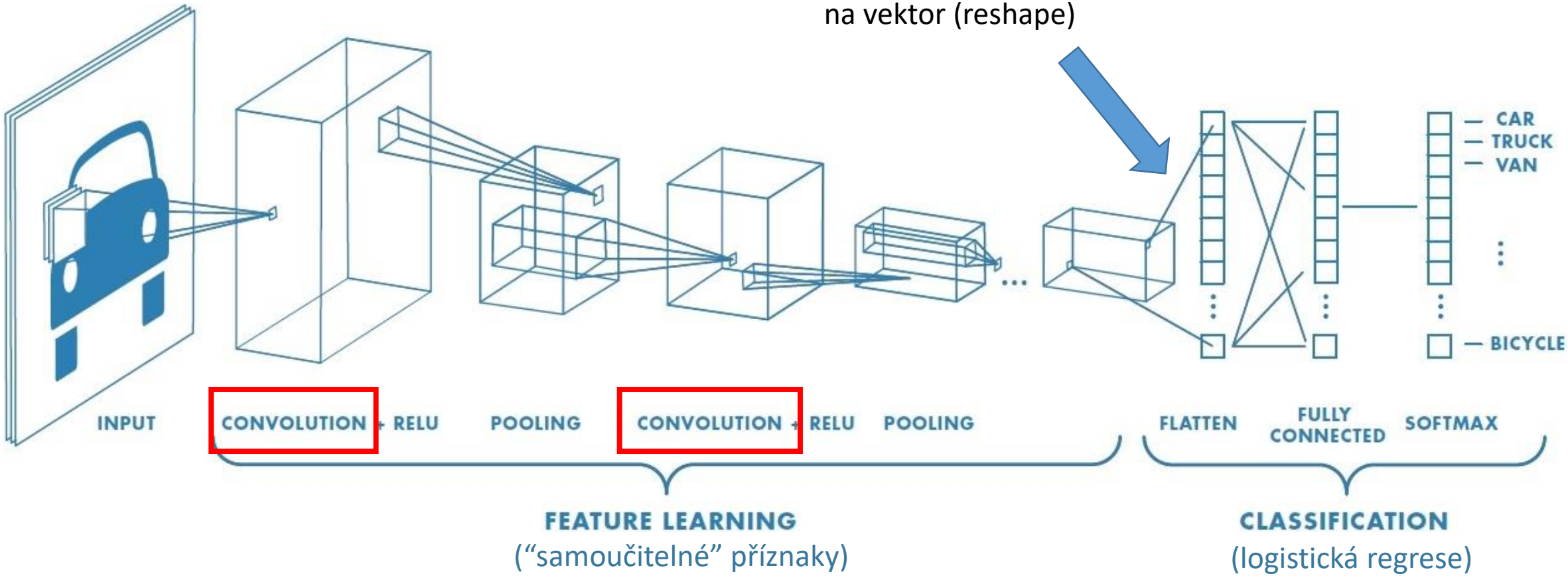
http://iamtrask.github.io/2015/07/12/basic-python-network/

Konvoluční sítě

Konvoluční síť (Convolutional Neural Network, CNN)

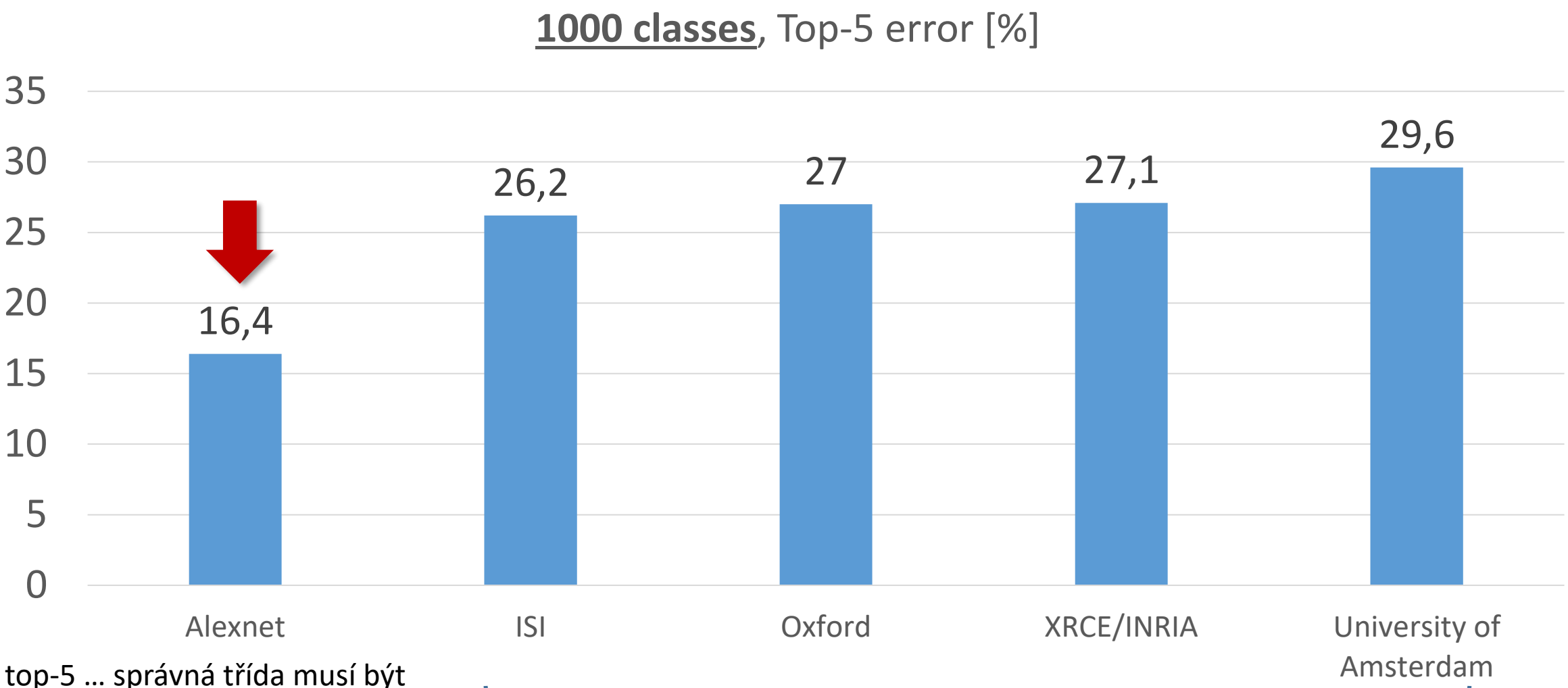
- zadefinováním konvoluce jako bloku v neurosíti nyní můžeme libovolně kombinovat s ostatními vrstvami

FLATTEN: převedení tenzoru na vektor (reshape)



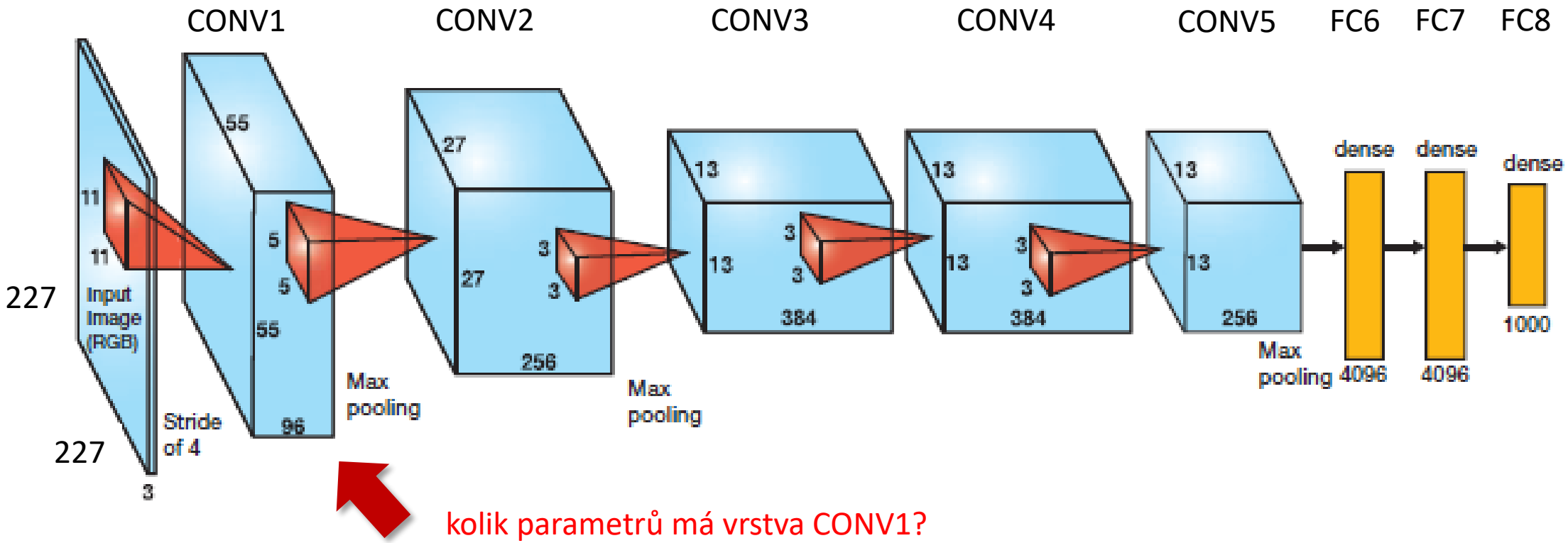
obrázek: https://ch.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Rozpoznávání ImageNet: Alexnet (2012)



top-5 … správná třída musí být do 5. místa dle skóre z výstupu klasifikátoru

Alexnet (2012)



kolik parametrů má vrstva CONV1?

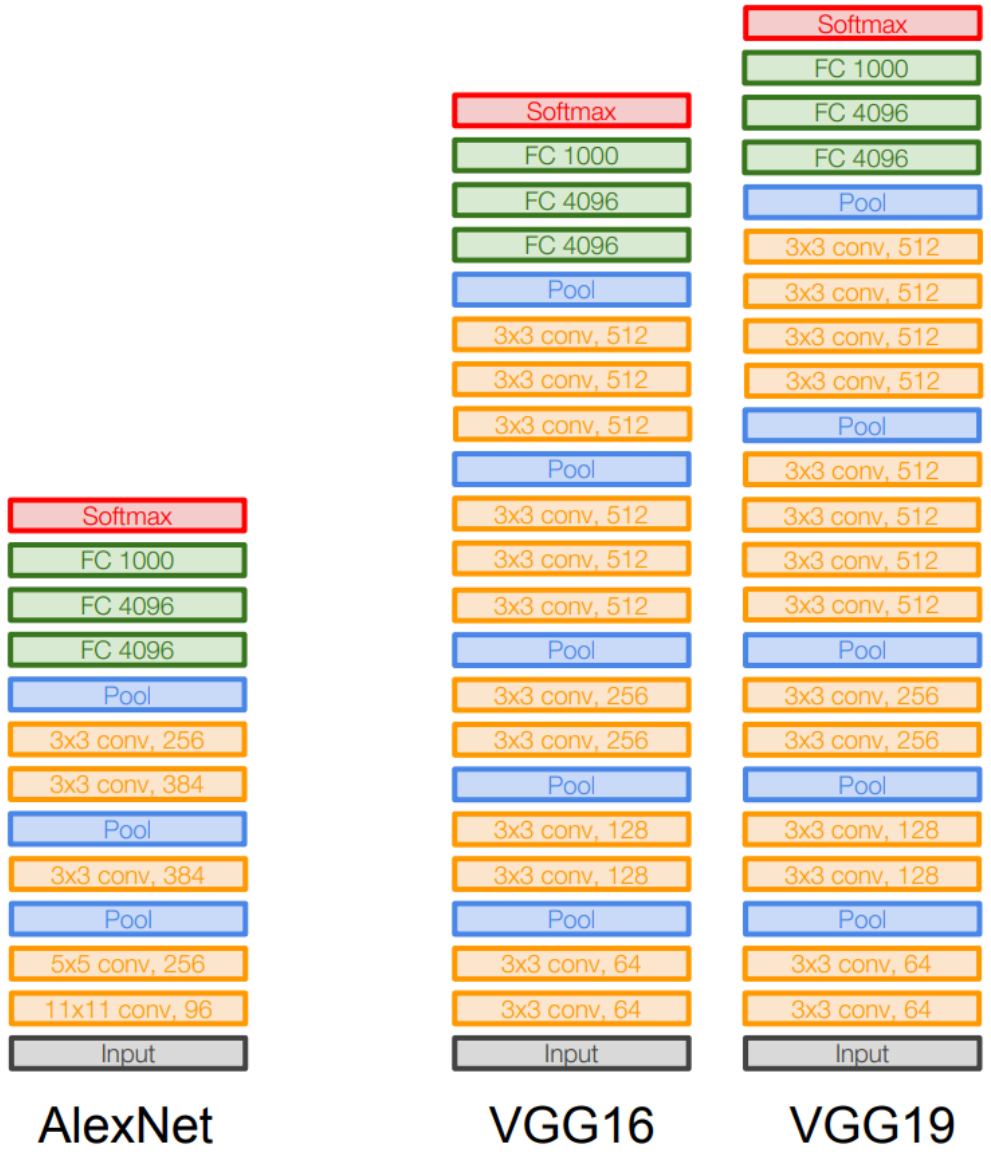
$$11 \cdot 11 \cdot 3 \cdot 96 = 34\,848$$

- architektura: CONV-POOL-NORM-CONV-POOL-NORM-CONV-CONV-CONV-FC-FC-FC
- “naškálovaná” LeNet-5

obrázek: http://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

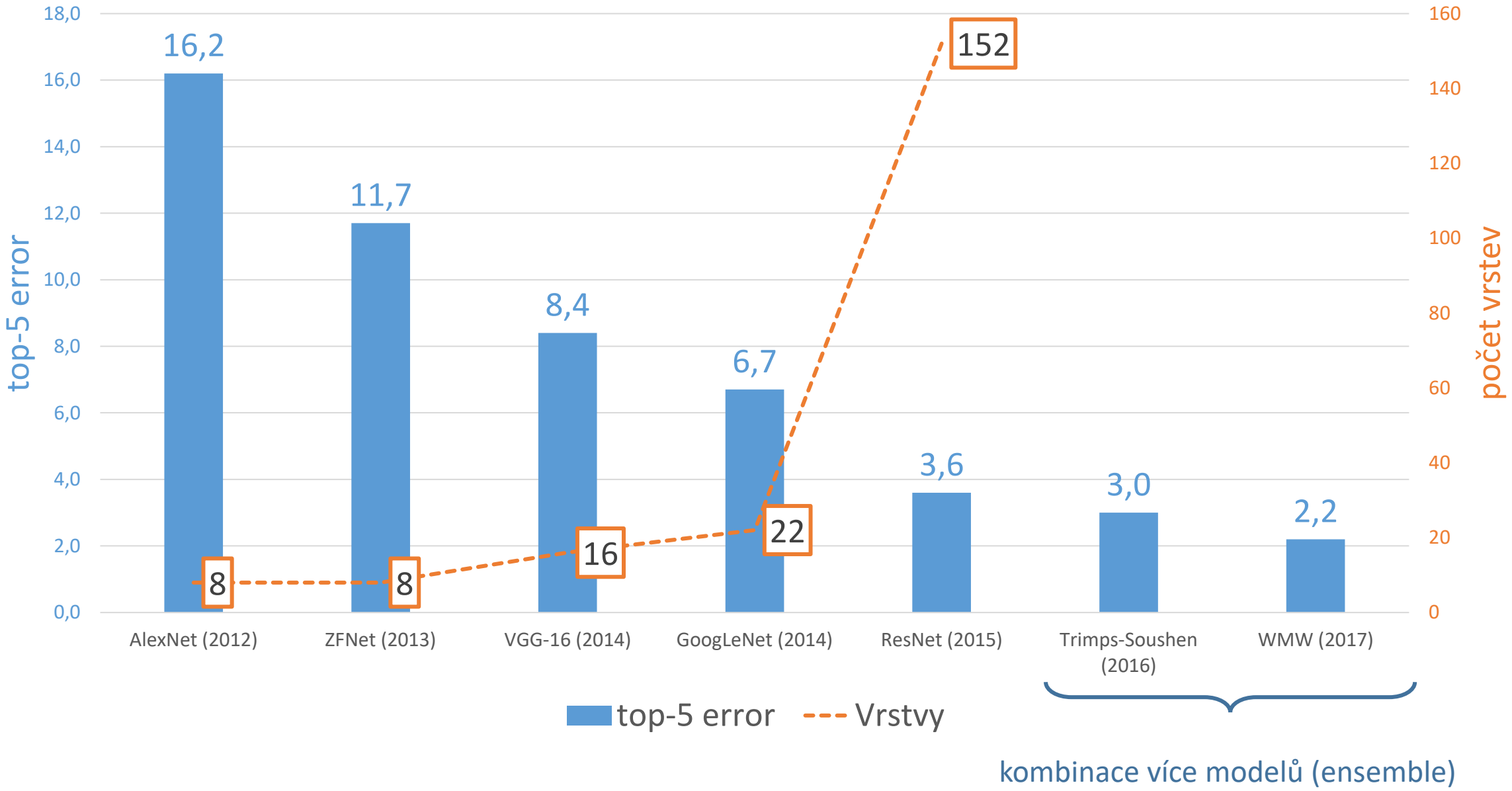
VGG (2014)

- [Simonyan, Zisserman: “Very Deep Convolutional Networks for Large-Scale Image Recognition”]
- Druhé místo ImageNet competition 2014
- Mnohem jednodušší architektura než vítěz (GoogLeNet)
- Velmi podobné AlexNet
- Místo 11x11 apod. konvolucí pouze 3x3
- Pouze 2x2 max-pooling
- Žádná lokální normalizace
- 16 a 19 vrstev
- VGG-16: 8.4 % top-5 error



obrázek: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

ImageNet klasifikace



ResNet (2015)

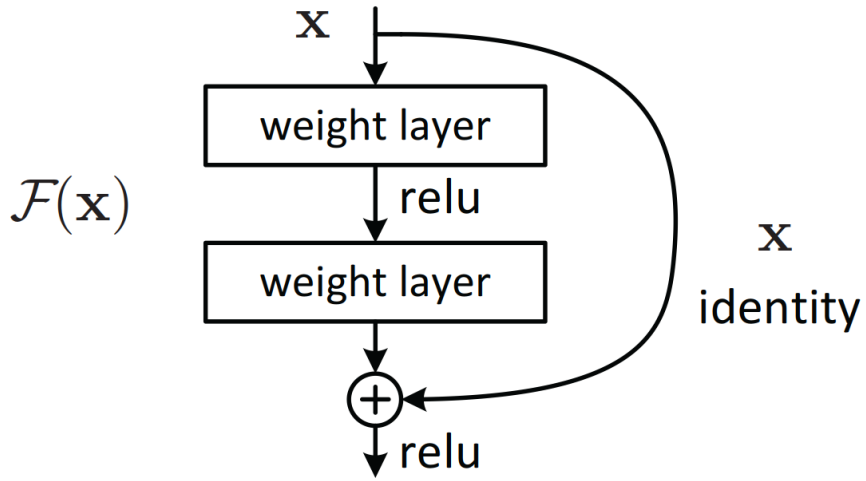
- He et al.: “Deep Residual Learning for Image Recognition”
- Cílem návrhu být co nejhlubší → 152 vrstev!
- Vítěz ImageNet 2015 ve všech kategoriích
- Vítěz MS COCO challenge
- 3.6 % top-5 error na ImageNet: lepší než člověk (cca 5 %)

Reziduální blok

- Podobně jako inception používá složitější bloky
- Výstup sestává ze součtu konvoluce a přímo mapovaného vstupu (identity)
- Síť se tedy učí pouze rezidua

$$\mathcal{F}(\mathrm{x}) = \mathcal{H}(\mathrm{x}) - \mathrm{x}$$

- "Naučit se nuly je jednodušší než identitu"



$$\mathcal{H}(x) = \mathcal{F}(x) + \mathrm{x}$$

Figure 2. Residual learning: a building block.

EfficientNet (2019)

- Tan, Le: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

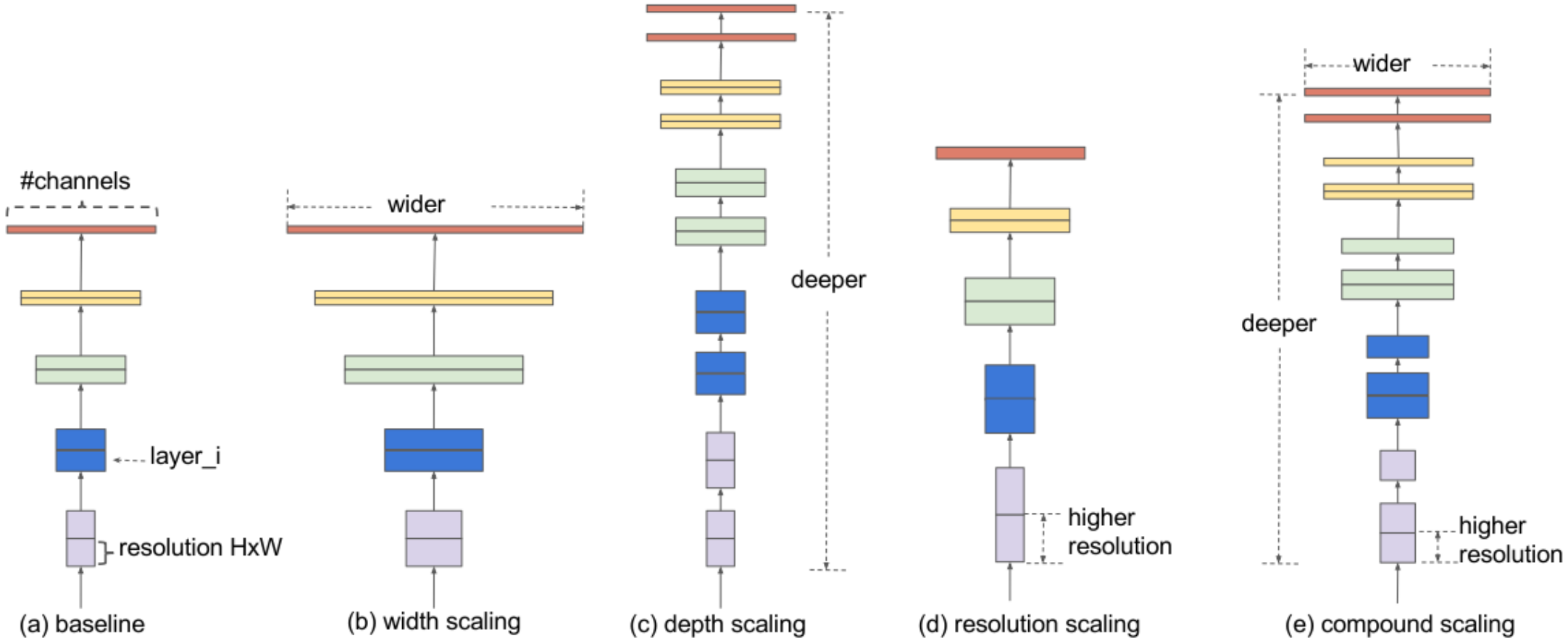
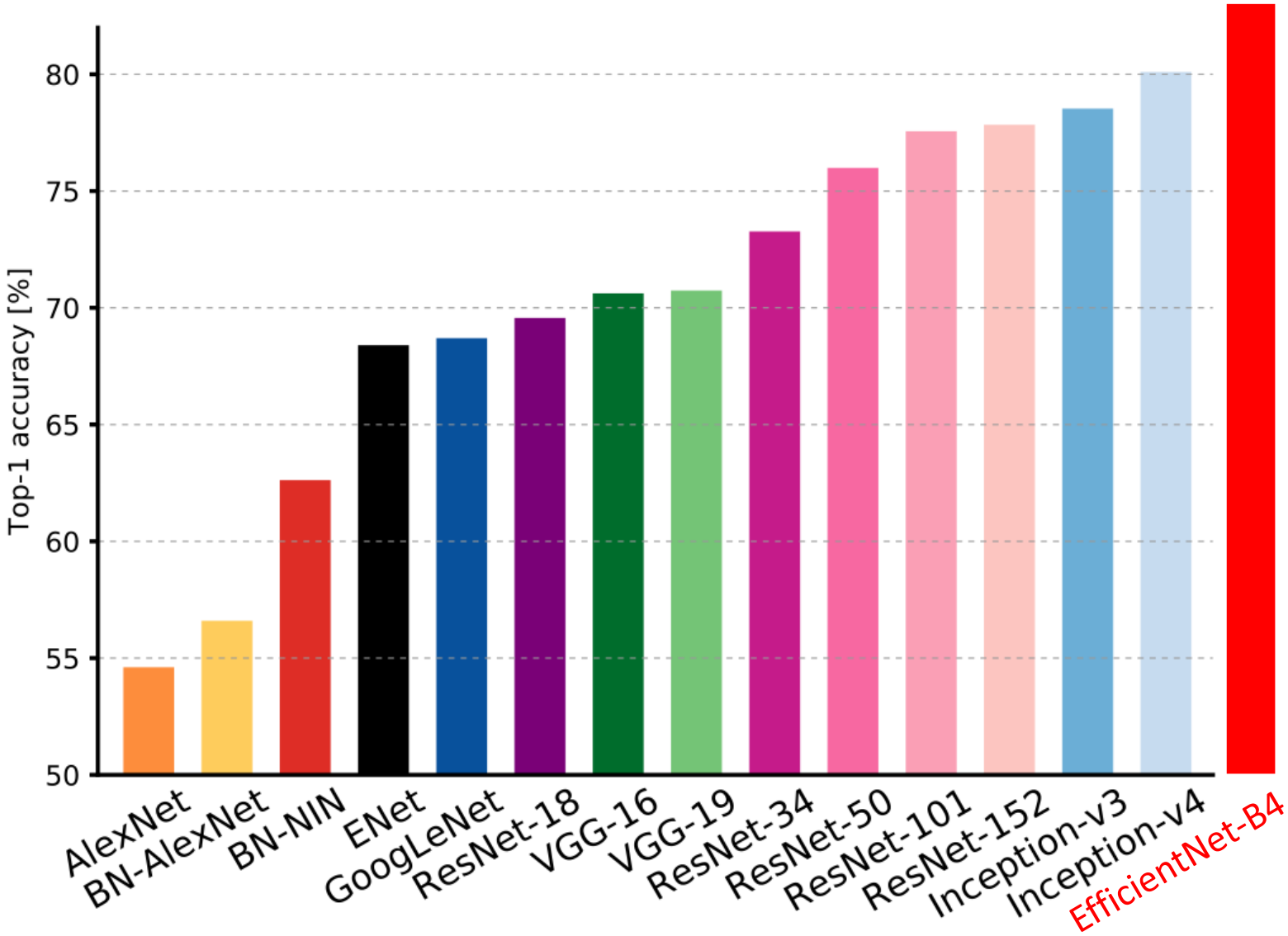


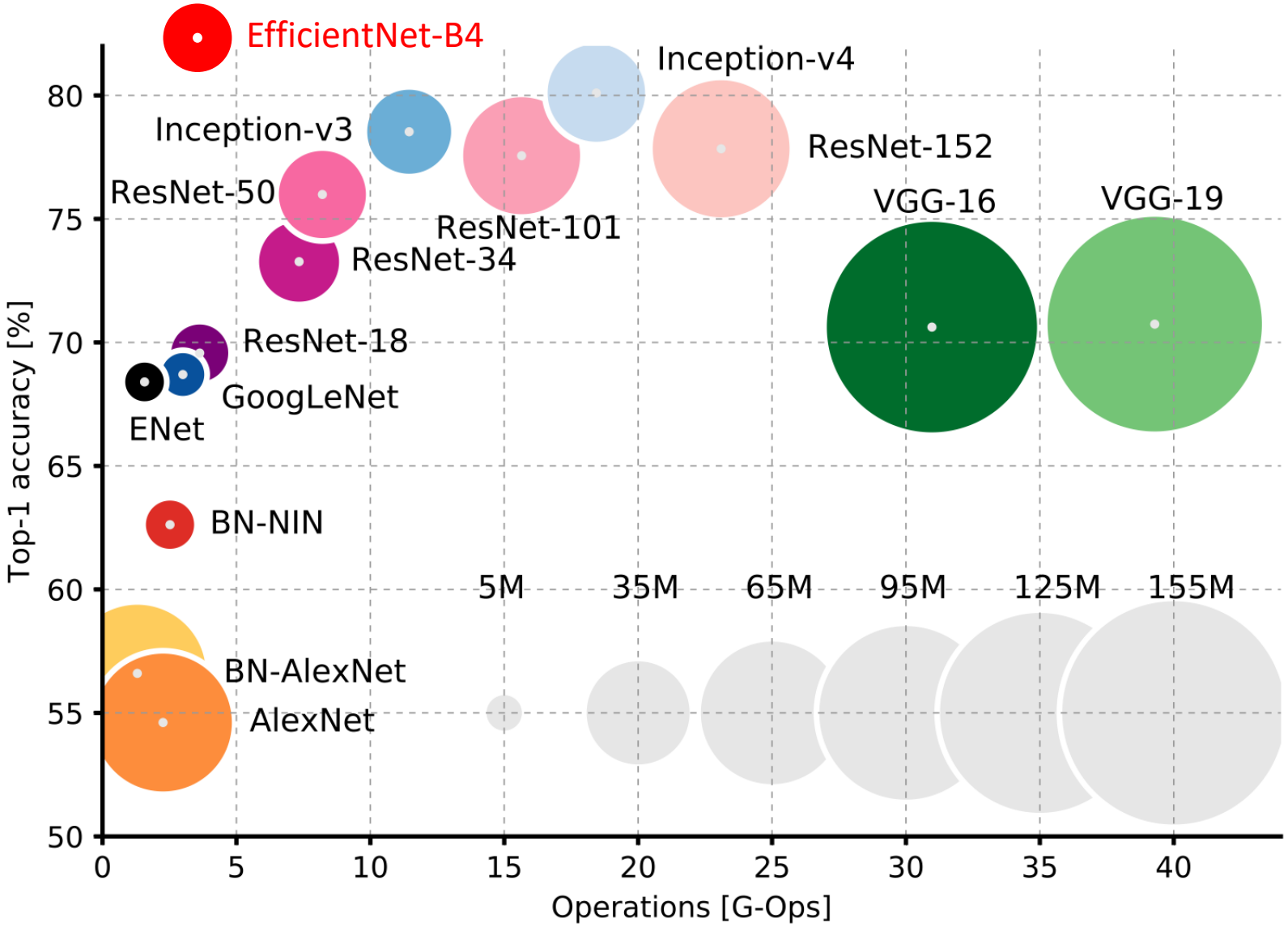
Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Srovnání nejpoužívanějších CNN architektur



obrázek: [Canziani et al.: "An Analysis of Deep Neural Network Models for Practical Applications"]

Srovnání nejpoužívanějších CNN architektur



velikost znázorňuje celkový počet parametrů

obrázek: Canziani et al.: "An Analysis of Deep Neural Network Models for Practical Applications"

Trénování sítí

Trénování sítí

- Typicky několik kroků

1. Příprava dat
2. Inicializace modelu (sítě)
3. Definice metrik a optimizéru
4. Trénovací smyčka

Trénování sítí

- Typicky několik kroků

1. **Příprava dat**
2. Inicializace modelu (sítě)
3. Definice metrik a optimizéru
4. Trénovací smyčka

```
training_set =
torchvision.datasets.FashionMNIST(
    './data',
    train = True,
    transform = transform,
    download = True
)

validation_set =
torchvision.datasets.FashionMNIST(
    './data',
    train = False,
    transform = transform,
    download = True
)
```

https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Trénování sítí

- Typicky několik kroků

1. **Příprava dat**
2. Inicializace modelu (sítě)
3. Definice metrik a optimizéru
4. Trénovací smyčka

```
training_loader =
torch.utils.data.DataLoader(
    training_set,
    batch_size = 4,
    shuffle = True
)
```

```
validation_loader =
torch.utils.data.DataLoader(
    validation_set,
    batch_size = 4,
    shuffle = False
)
```

https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Trénování sítí

- Typicky několik kroků

1. Příprava dat
2. **Inicializace modelu (sítě)**
3. Definice metrik a optimizéru
4. Trénovací smyčka

```
class GarmentClassifier(nn.Module):
    def __init__(self):
        super(GarmentClassifier, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = GarmentClassifier()
```

https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Trénování sítí

- Typicky několik kroků

1. Příprava dat
2. Inicializace modelu (sítě)
3. **Definice metrik a optimizéru**
4. Trénovací smyčka

```
loss_fn = torch.nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(
    model.parameters(),
    lr = 0.001,
    momentum = 0.9
)
```

https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Trénování sítí

- Typicky několik kroků

1. Příprava dat
2. Inicializace modelu (sítě)
3. Definice metrik a optimizéru
4. **Trénovací smyčka**

```
def train_one_epoch():
    running_loss = 0.
    last_loss = 0.

    for i, data in enumerate(training_loader):
        # forward
        inputs, labels = data
        outputs = model(inputs)
        loss = loss_fn(outputs, labels)

        # backprop
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Gather data and report
        running_loss += loss.item()
        if i % 1000 == 999:
            last_loss = running_loss / 1000 # loss per batch
            ...
    return last_loss
```

https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Trénování sítí

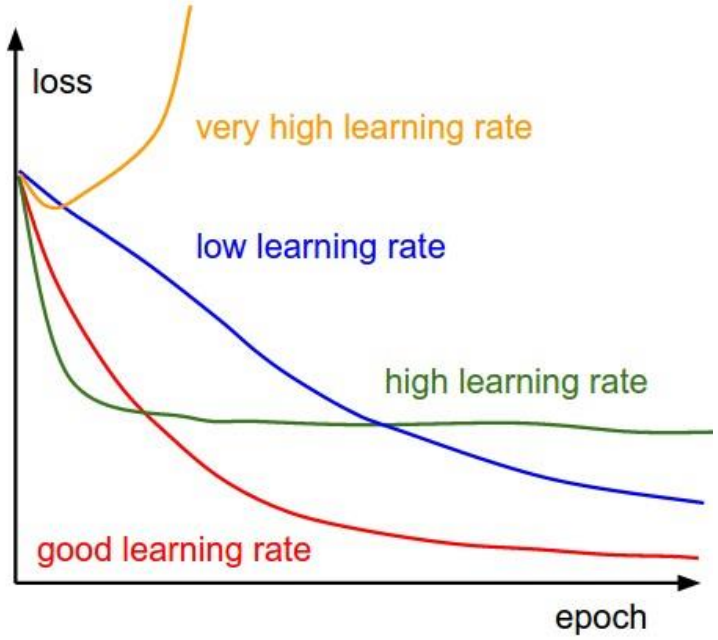
- Typicky několik kroků

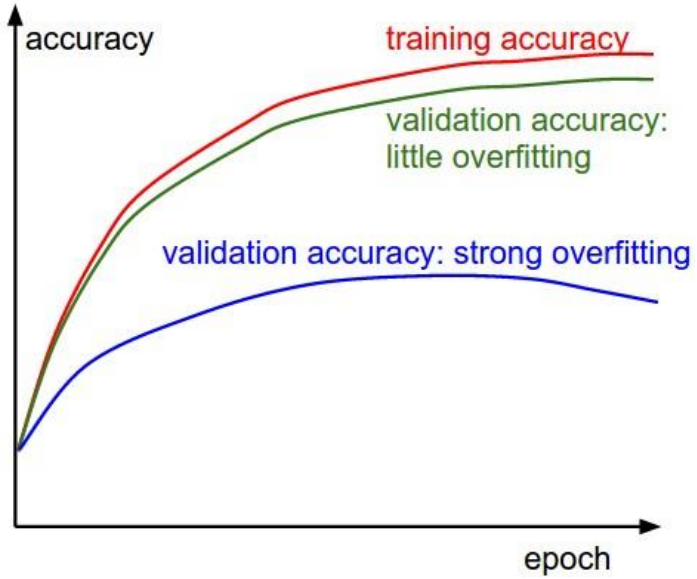
1. Příprava dat
2. Inicializace modelu (sítě)
3. Definice metrik a optimizéru
4. **Trénovací smyčka**

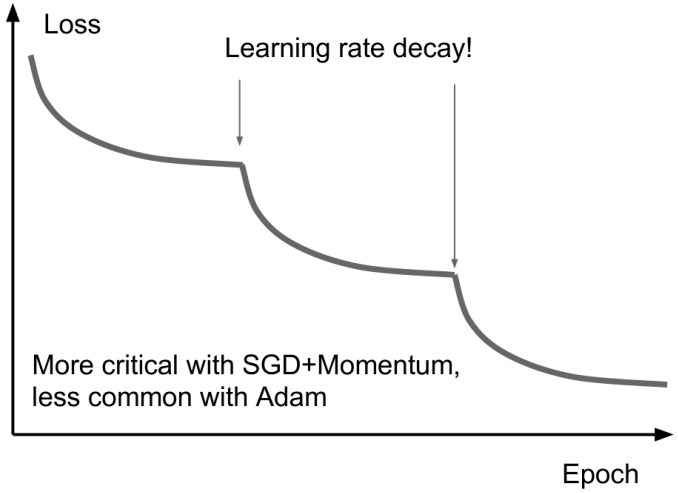
```
for epoch in range(EPOCHS):
    # training
    model.train(True)
    avg_loss = train_one_epoch()

    # validation
    running_vloss = 0.0
    model.eval()
    with torch.no_grad():
        for i, vdata in enumerate(validation_loader):
            vinputs, vlabels = vdata
            voutputs = model(vinputs)
            vloss = loss_fn(voutputs, vlabels)
            running_vloss += vloss

    avg_vloss = running_vloss / (i + 1)
    print('LOSS train {} valid {}'.format(avg_loss, avg_vloss))

    # Track best performance, and save the model's state
    if avg_vloss < best_vloss:
        best_vloss = avg_vloss
        torch.save(model.state_dict(), model_path)
```

https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

Trénování







- Monitorovat hodnotu lossu a podle toho nastavit lr
- Nebo lze použít automatické hledání lr
- Pokud funguje, zkusit lr decay

obrázky: https://cs231n.github.io/neural-networks-3/

Transfer learning

Trénování konvolučních sítí při málo datech

- Popsané architektury mají obvykle miliony parametrů
- Malé datasety na jejich trénování nestačí → výrazný overfit
- I pokud data máme: trénování VGG na ImageNet trvalo autorům 2-3 týdny, a to i s 4x NVIDIA Titan Black GPU
- **Naštěstí lze obejít!**
    1. Můžeme vzít existující již natrénovaný model (např. VGG-16)
    2. Odstraníme poslední klasifikační vrstvu
    3. Nahradíme vlastní

Transfer learning

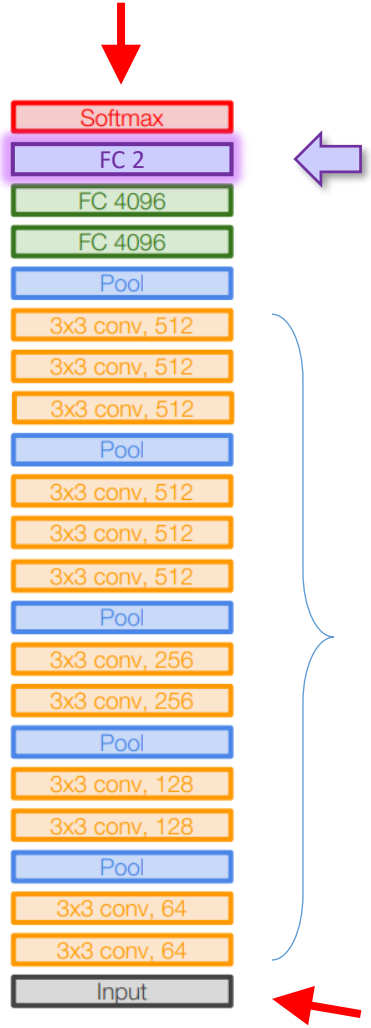
1000 tříd pro ImageNet

VGG16

ImageNet data

poslední lineární vrstvu tvaru 4096×1000 zahodíme

2 třídy: kočky vs psi



VGG16

připojíme vlastní lineární vrstvu tvaru 4096×2 a náhodně inicializujeme

pokud máme málo dat, je lepší konvoluční vrstvy netrénovat → **layer freeze**

můžeme použít vlastní data

Transfer learning v PyTorch

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)
```

"zmrazení" vrstev, nebudou se trénovat a zůstávají konst. → síť pouze jako extractor příznaků

poslední vrstvu klasifikující do 1000 ImageNet tříd nahradíme vlastní, která má pouze 2 třídy

jako seznam parametrů pro optimalizaci předáváme pouze poslední lineární vrstvu (pouze pro urychlení)

- poté, co je poslední vrstva natrénovaná, je možné opět uvolnit ("rozmrazit") i konvoluční vrstvy a model dále zlepšit (fine tuning)

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Předzpracování dat u konvolučních sítí

- U konvolučních sítí často používané konstantní hodnoty
- Odečtení průměrného pixelu
  `out = rgb - mean_pixel`
  kde `mean_pixel` je trojice [r, g, b]

často lze vidět konkrétní hodnoty [123.68, 116.779, 103.939], které jsou průměrným pixelem na databázi ImageNet

- Méně časté: odečtení průměrného obrázku

  `out = rgb - mean_image`

  kde `mean_image` je 32x32x3

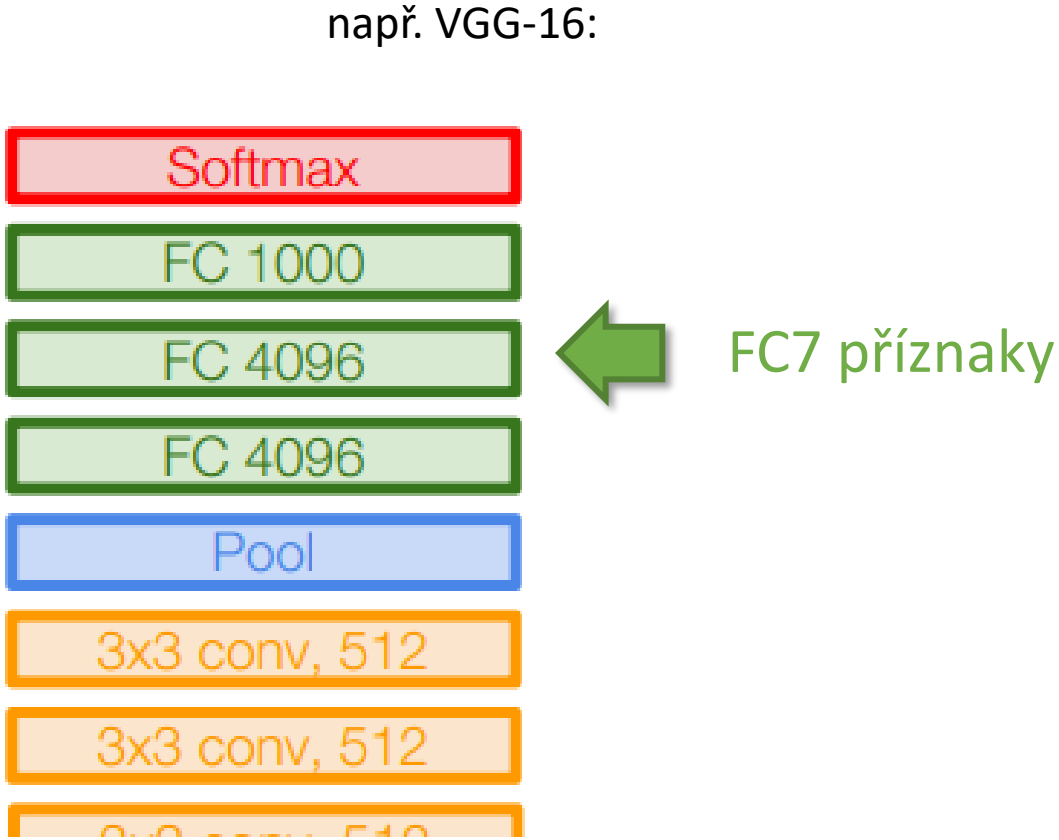
- Např. všechny předtrénované modely Pytorch:

```
transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

odečtení průměru a normalizace standardní odchylky

CNN příznaky

- Výstup z posledních lineárních vrstev lze použít např. jako příznaky (tzv. FC7) → CNN jako “feature extractor”
- Např. VGG-16 předposlední vrstva má rozměr 4096
- Nad těmito příznaky je možné natrénovat libovolný klasifikátor, třeba i rozhodovací stromy/lesy, bayesovské klasifikátory, …
- Lze take využít pro urychlení trénování: celý dataset projet sítí a pro každý obrázek uložit na disk FC7 příznaky
- Během trénování se pak nemusí znovu a znovu provádět dopředný průchod celou sítí, pouze těmi posledními



Transfer learning: shrnutí

specifičtější příznaky (obličeje, text, …)

obecné příznaky (hrany, bloby, …)

	podobná data	**odlišná data**
málo dat	trénovat spíše jen poslední vrstvu	problém ☺
hodně dat	fine tune několika vrstev (lze ale i celou síť)	fine tune více vrstev nebo i celé sítě

slide: http://cs231n.stanford.edu/