

# Aplikace neuronových sítí

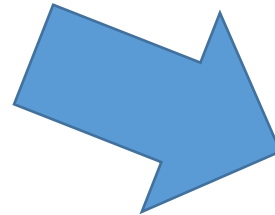
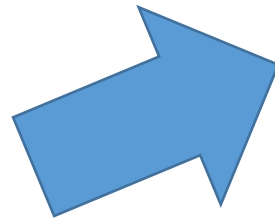
---

Lineární klasifikace, Softmax, SVM

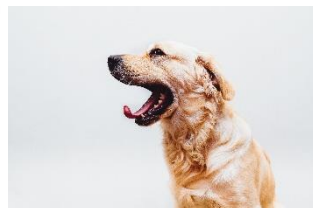
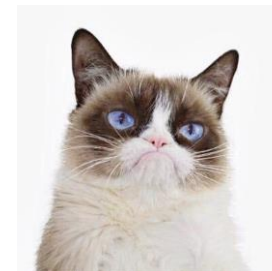
# Lineární klasifikace

---

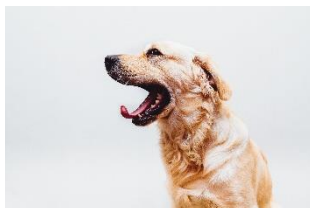
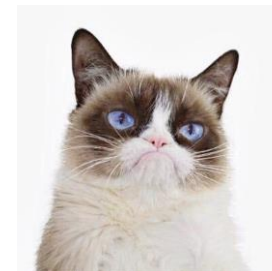
# Problém klasifikace



# Vzorové obrázky neboli trénovací data

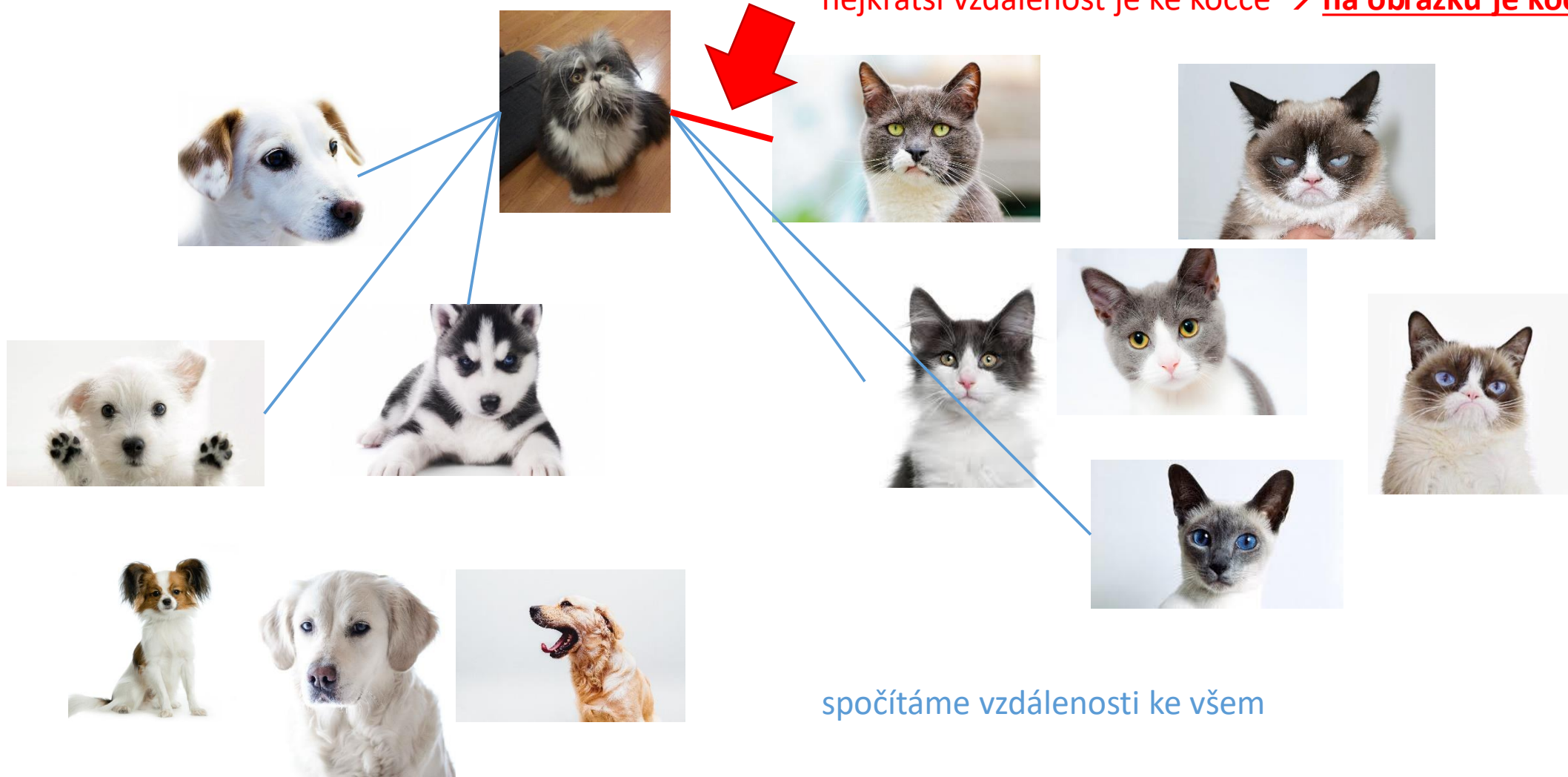


# Predikce na neznámém obrázku



# Metoda nejbližšího souseda

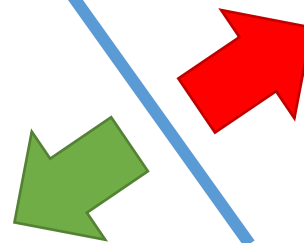
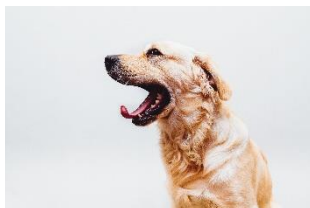
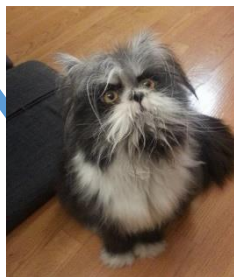
nejkratší vzdálenost je ke kočce → na obrázku je kočka





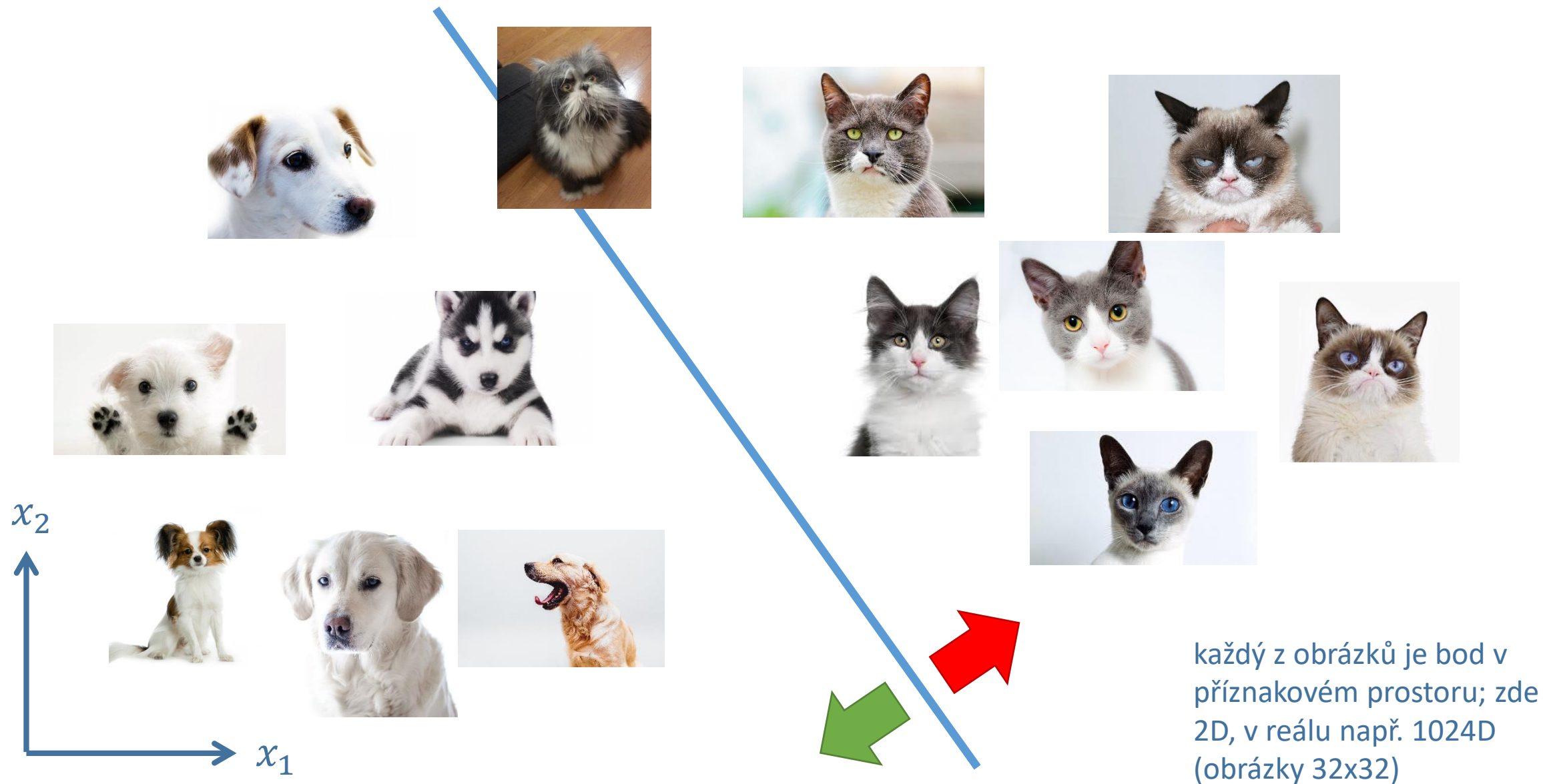
# Lineární klasifikace

obrázek je na straně koček → na obrázku je kočka



prostor přímkou (rovinou) rozdělíme  
na dvě poloviny; na jedné straně  
jsou psi, na druhé kočky

# Příznakový prostor





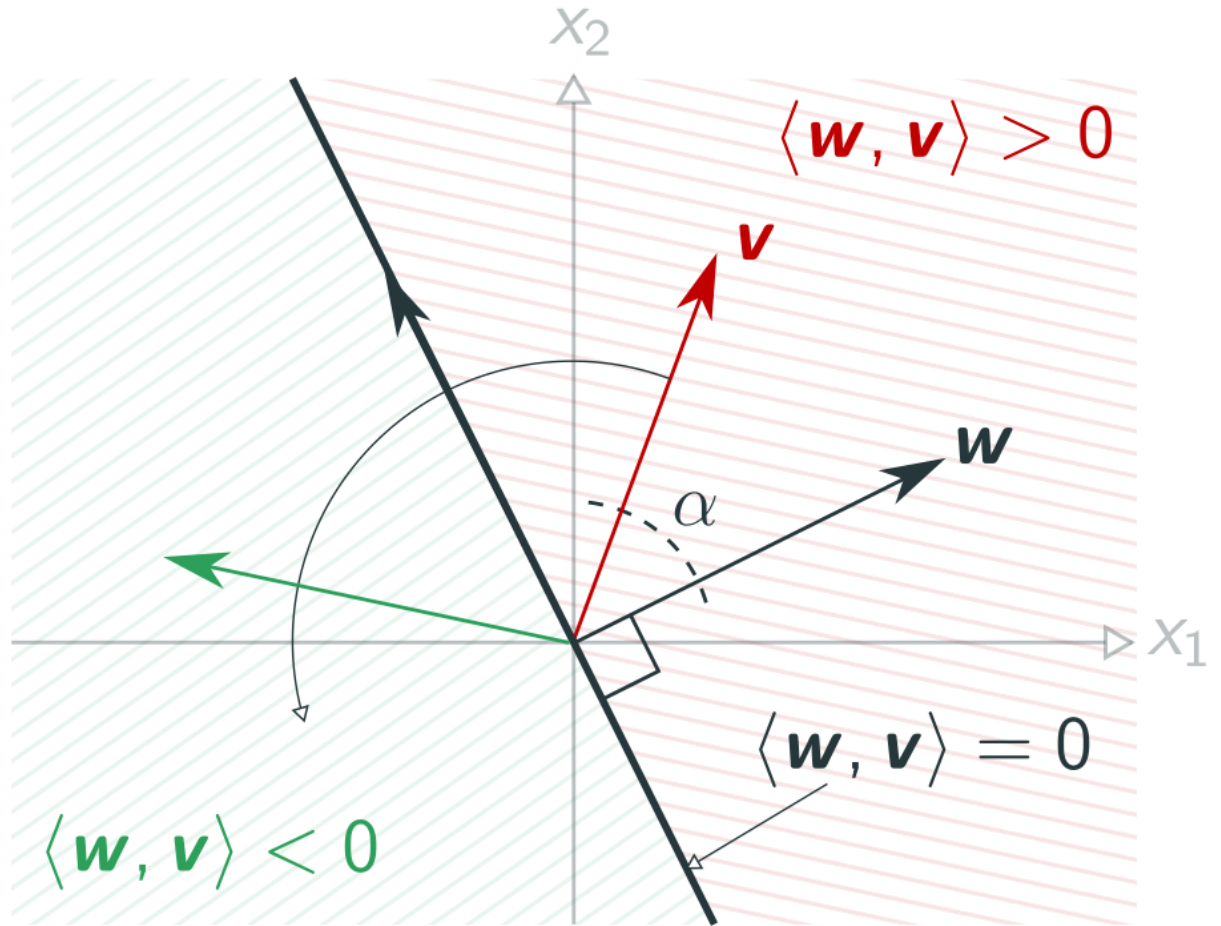
# Na jaké straně bod je? Skalární součin vektorů

- Definice:

$$\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^\top \mathbf{w} = \sum_i v_i w_i$$

- Jinak také **dot product** či **inner product** → pro naše potřeby ekvivalentní pojmy
- Souvisí s identitou:

$$\cos \alpha = \frac{\langle \mathbf{v}, \mathbf{w} \rangle}{\|\mathbf{v}\| \cdot \|\mathbf{w}\|}$$



# Python

```
s = 0.
```

```
for d in range(D):
```

```
    s += v[d] * w[d]
```

# numpy

```
import numpy as np
```

```
s = np.dot(v, w)
```

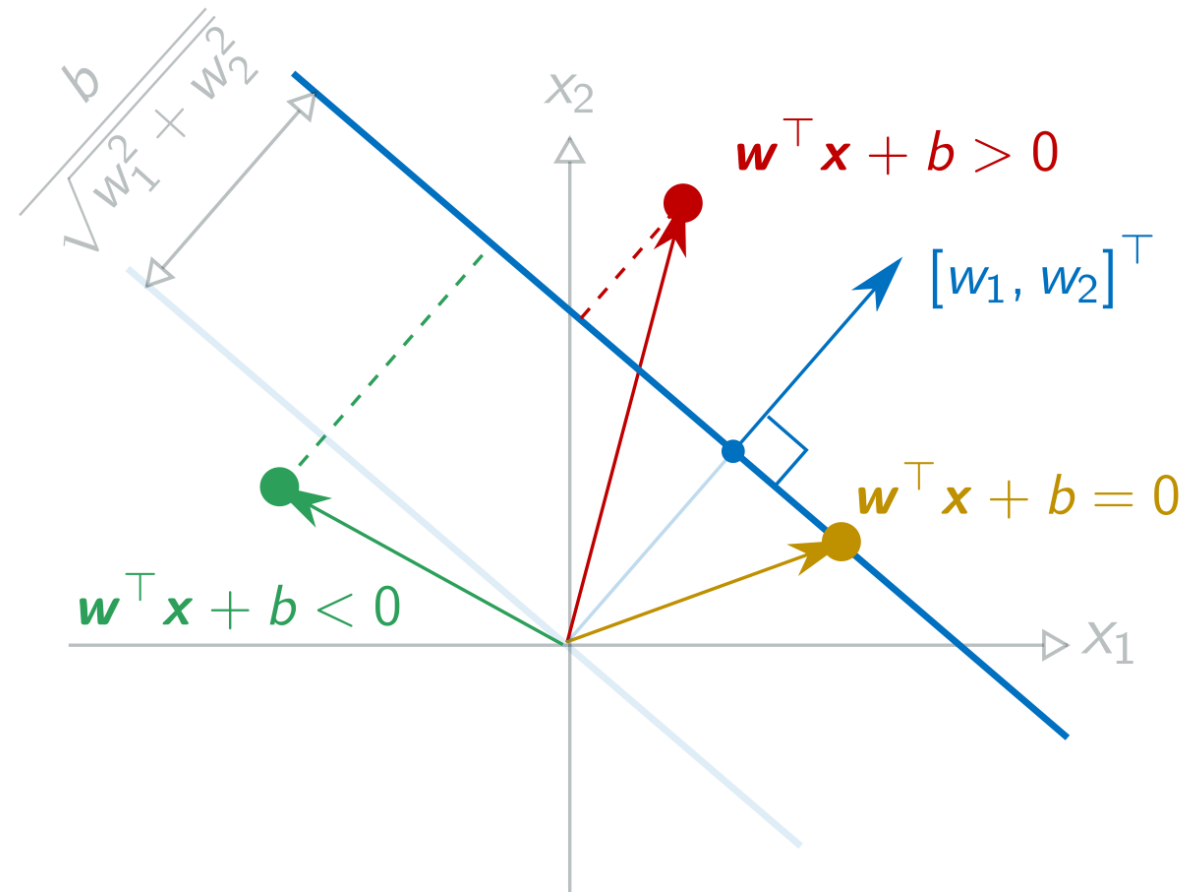
# Projekce bodu na přímku

- Pro libovolný bod  $\mathbf{x} = [x_1, x_2]^T$

$$d = \frac{w_1 x_1 + w_2 x_2 + b}{\sqrt{w_1^2 + w_2^2}}$$

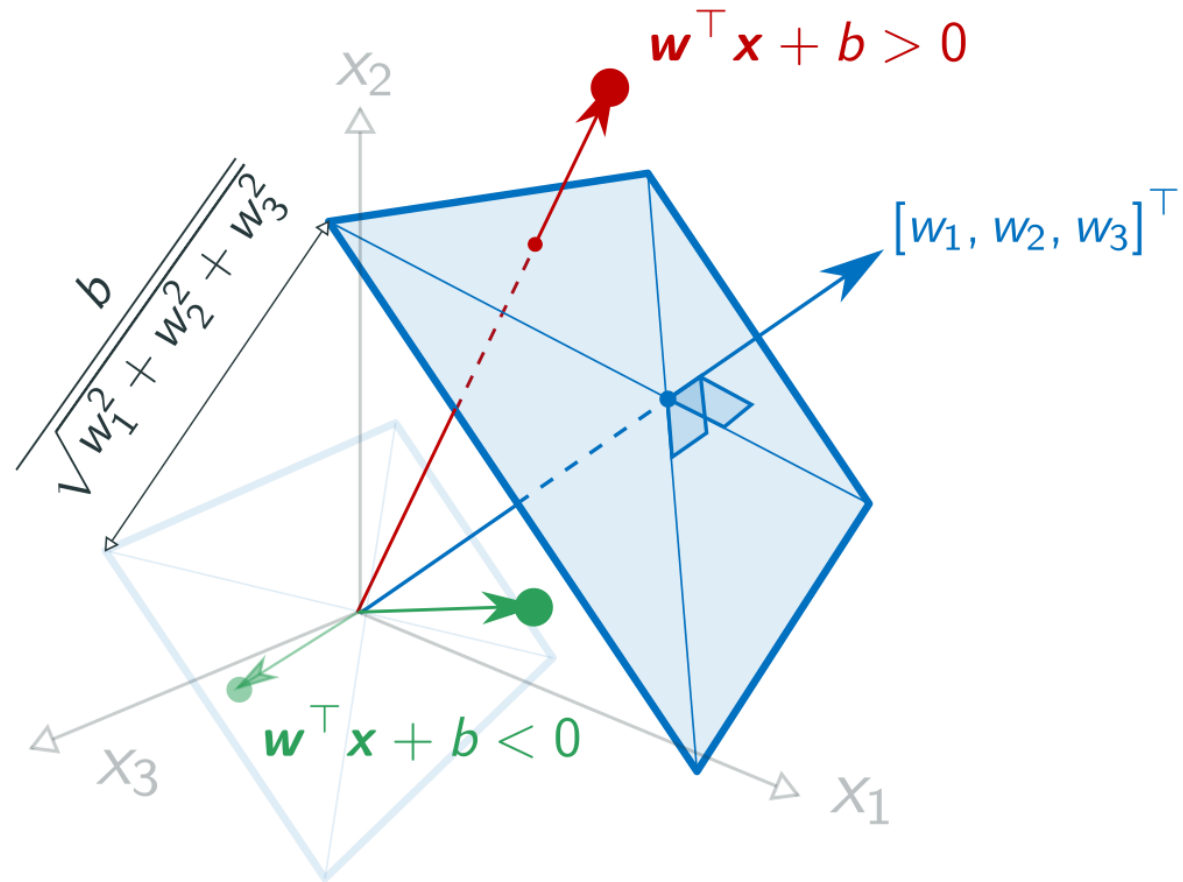
je *orientovaná* vzdálenost od přímky:

- $d > 0 \rightarrow \mathbf{x}$  leží “nad” přímkou
  - $d = 0 \rightarrow \mathbf{x}$  leží na přímce
  - $d < 0 \rightarrow \mathbf{x}$  leží “pod” přímkou
- Zdali “nad” či “pod” určuje směr normály  
 $\mathbf{w} = [w_1, w_2]^T$



# Vícerozměrný prostor

Ve více rozměrech přímku nahrazuje rovina



# No jo, jenže kde vezmu tu správnou přímku?

- Na jakou přímku/nadrovinu vektory promítat tak, abychom dosáhli max. úspěšnosti klasifikace?
- Zavedeme kritérium  $L(\boldsymbol{\theta})$ , které bude kvantifikovat, jak moc **špatné** aktuální parametry  $\boldsymbol{\theta} = (\mathbf{w}, b)$  jsou  $\rightarrow$  bude popisovat **chybu modelu**
- Úkolem potom bude nalézt optimální parametry  $\boldsymbol{\theta}^*$ , které tuto chybu na vzorových datech  $\mathbf{X}$  **minimalizují**, neboli



$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$



# Kritérium: binární logistická regrese

- Logistická regrese definuje kritérium, tzv. **křížovou entropii** (zde binární varianta)

$$L_n = -y_n \log \{\sigma(s_n)\} - (1 - y_n) \log \{1 - \sigma(s_n)\}$$

Sigmoid

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

... normalizuje do intervalu  $\langle 0, 1 \rangle$

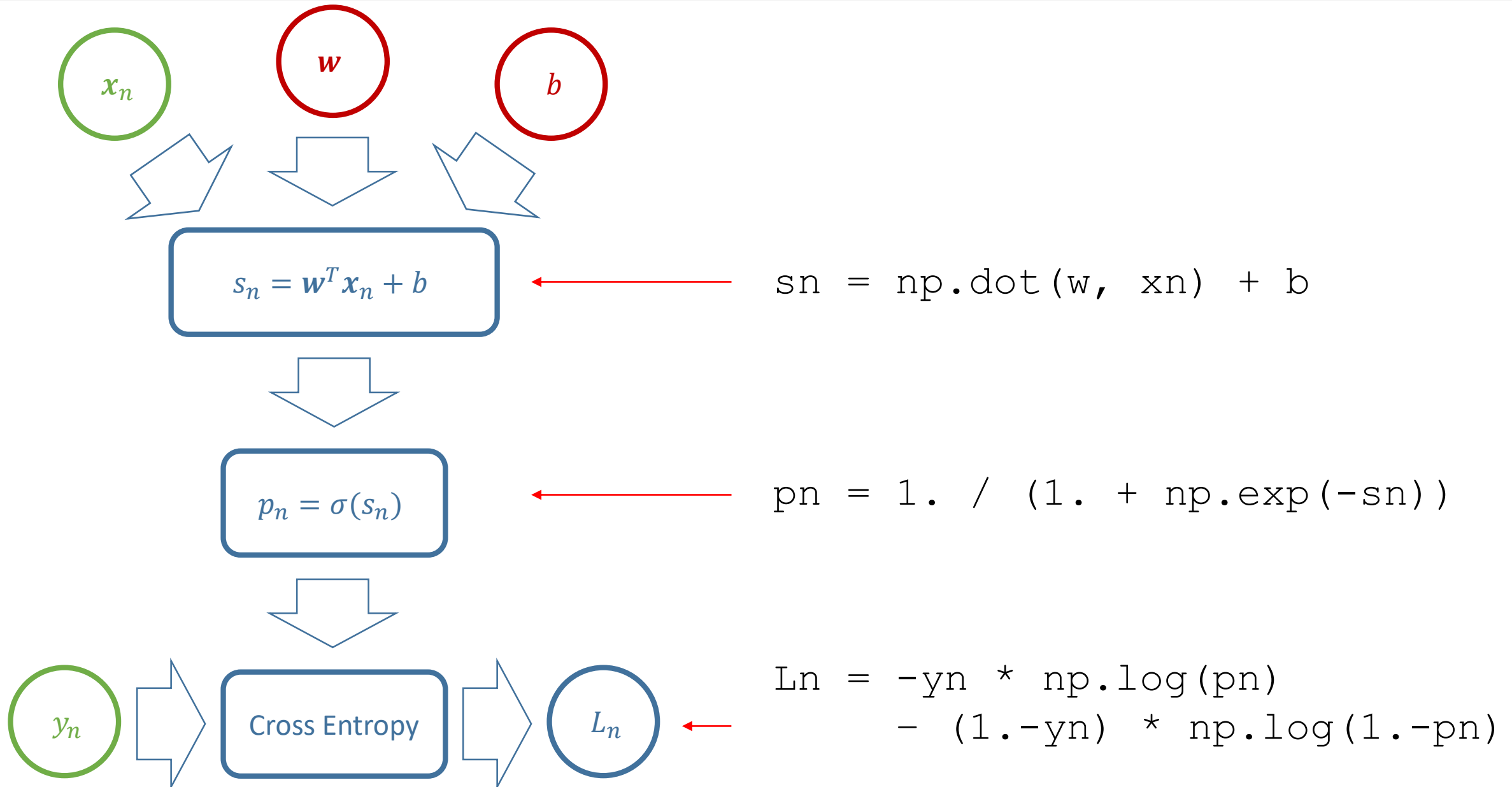
Lineární skóre

$$s_n = \mathbf{w}^\top \mathbf{x}_n + b$$

... projekce bodu  $\mathbf{x}_n$  na dělicí nadrovinu



# Schématické znázornění logistické regrese



# Logistická regrese jako neurosít'

- Na logistickou regresi možné nahlížet jako na jednovrstvou neuronovou síť, tzv. single layer perceptron
- Pozor: neplést s perceptronem jako učícím algoritmem
  - <https://en.wikipedia.org/wiki/Perceptron>
- Výstupem je pravděpodobnost, že vstup náleží třídě "+1"

# Formulace logistické regrese

- Jelikož  $\mathbf{x}_n$  považujeme za nezávislé, pro celou trénovací sadu je loss

$$L(\mathbf{w}, b) = \sum_{n=1}^N L_n = \sum_{n=1}^N (-y_n \log\{\sigma(s_n)\} - \dots)$$

- Celkově pak úloha je

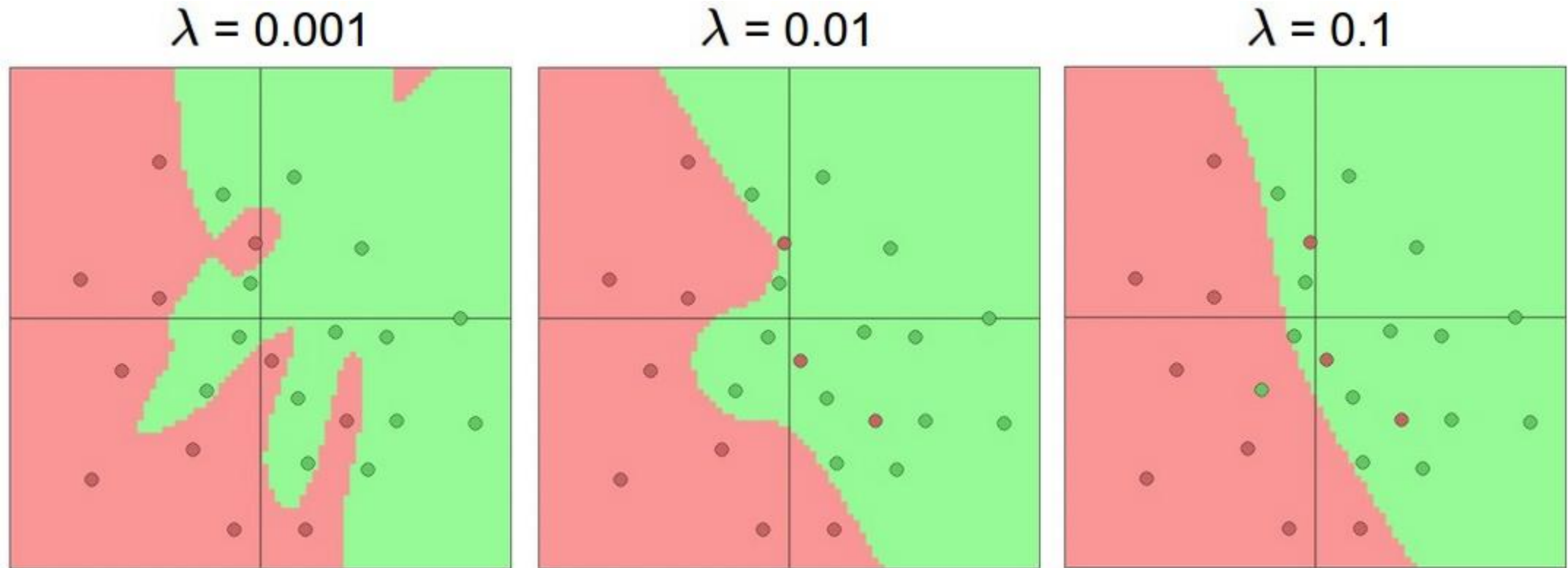
$$\mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} L(\mathbf{w}, b) + \lambda \|\mathbf{w}\|^2$$

- $\lambda \|\mathbf{w}\|^2$  je regularizační člen
  - $\lambda$  je hyperparametr, často označovaný jako weight decay

# Proč regularizace?

- Např.  $\mathbf{x} = [1, 1, 1, 1]^T$  a dvojice různé parametry:
  - $\mathbf{w}_1 = [1, 0, 0, 0]^T$
  - $\mathbf{w}_2 = [0.25, 0.25, 0.25, 0.25]^T$
- Přestože  $\mathbf{w}_1^T \mathbf{x} = \mathbf{w}_2^T \mathbf{x} = 1$ , preferujeme  $\mathbf{w}_2$
- Brání přeučení
  - $\mathbf{w}_2$  má menší normu
  - $\mathbf{w}_1$  sází všechno na jeden příznak, zatímco  $\mathbf{w}_2$  důležitost rozkládá
  - normu  $\|\mathbf{w}\|_2$  lze interpretovat jako apriorní pravděpodobnost
  - regularizace brání změnám  $\rightarrow$  trénování méně reaguje na změny

# Vliv regularizace



<http://cs231n.github.io/neural-networks-1/>



# Formy regularizace

$$J(\mathbf{W}) = L(\mathbf{W}) + \lambda R(\mathbf{W})$$

Nejčastěji L2	$R(\mathbf{W}) = \ \mathbf{W}\ _2^2 = \sum_{i,j} w_{ij}^2$
Méně často L1	$R(\mathbf{W}) = \ \mathbf{W}\ _1 = \sum_{i,j}  w_{ij} $
Kombinace L1 + L2	$R(\mathbf{W}) = \lambda_1 \sum_{i,j} w_{ij}^2 + \lambda_2 \sum_{i,j}  w_{ij} $
dropout a další	více v třetí přednášce

# Metoda největšího spádu (Gradient Descent)

## Začínáme:

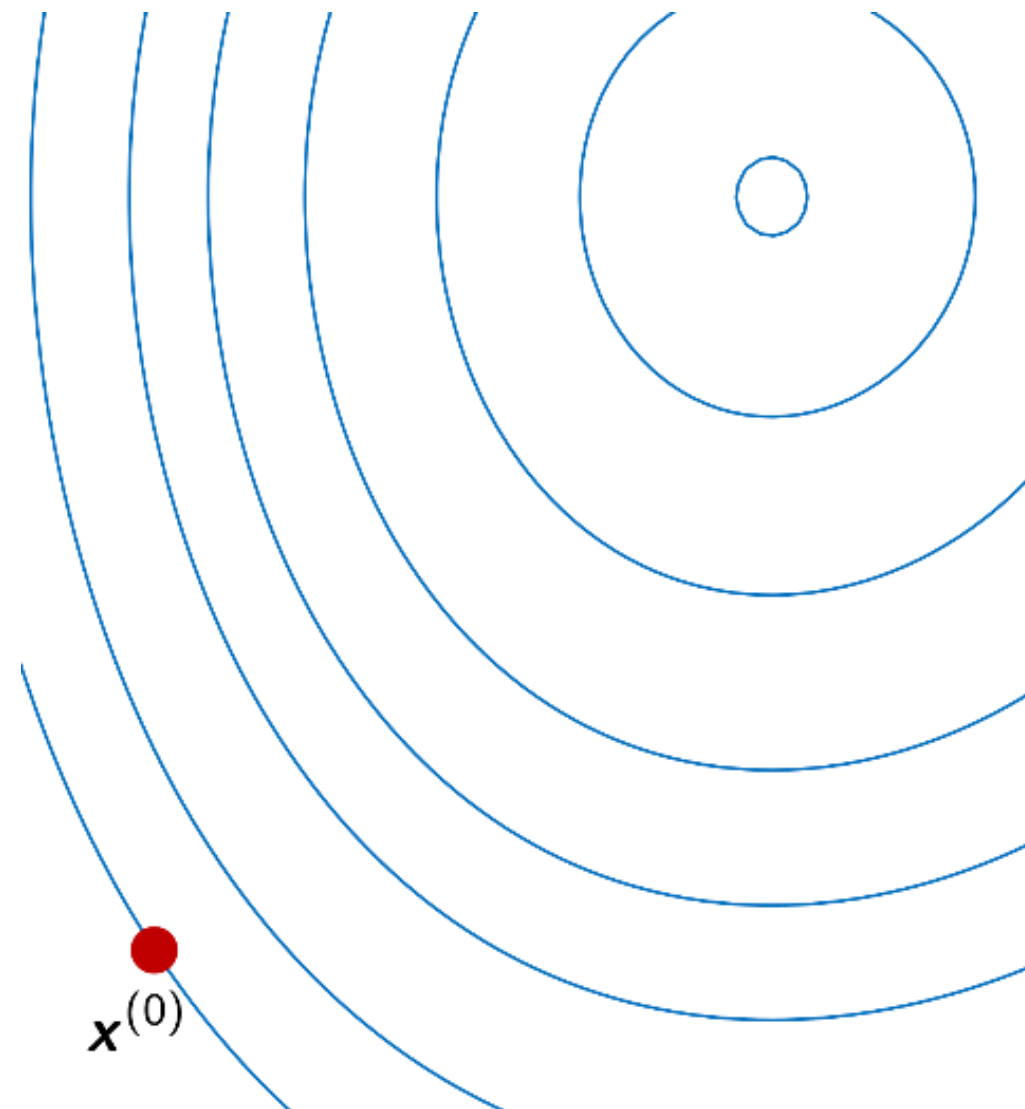
- známe výchozí pozici

## Opakujeme:

$$\mathbf{x}^{(t+1)} \longleftarrow \mathbf{x}^{(t)} - \gamma \nabla f \left( \mathbf{x}^{(t)} \right)$$

## Skončíme:

- po vykonaném počtu kroků
- poloha už se nemění (konvergence)



# Metoda největšího spádu (Gradient Descent)

## Začínáme:

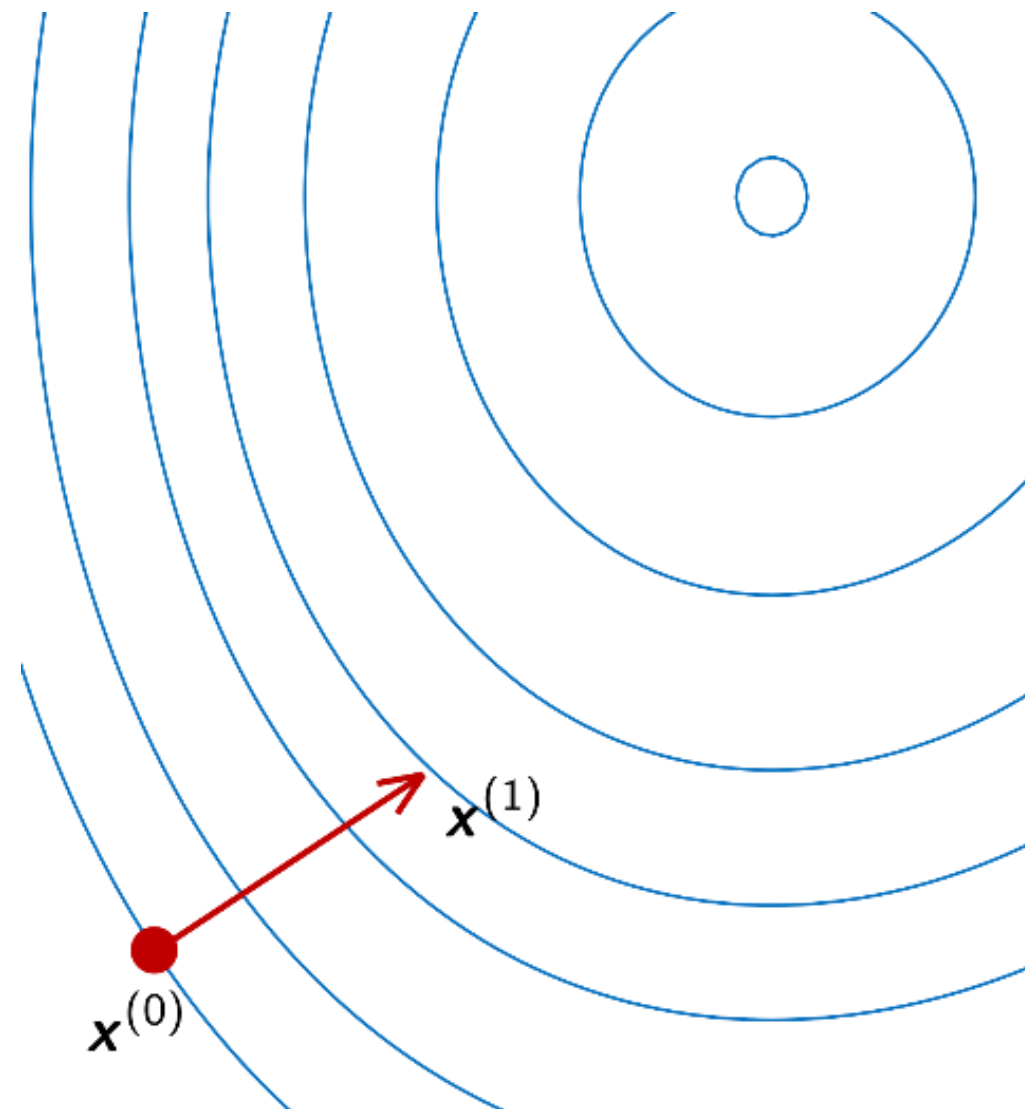
- známe výchozí pozici

## Opakujeme:

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

## Skončíme:

- po vykonaném počtu kroků
- poloha už se nemění (konvergence)



# Metoda největšího spádu (Gradient Descent)

## Začínáme:

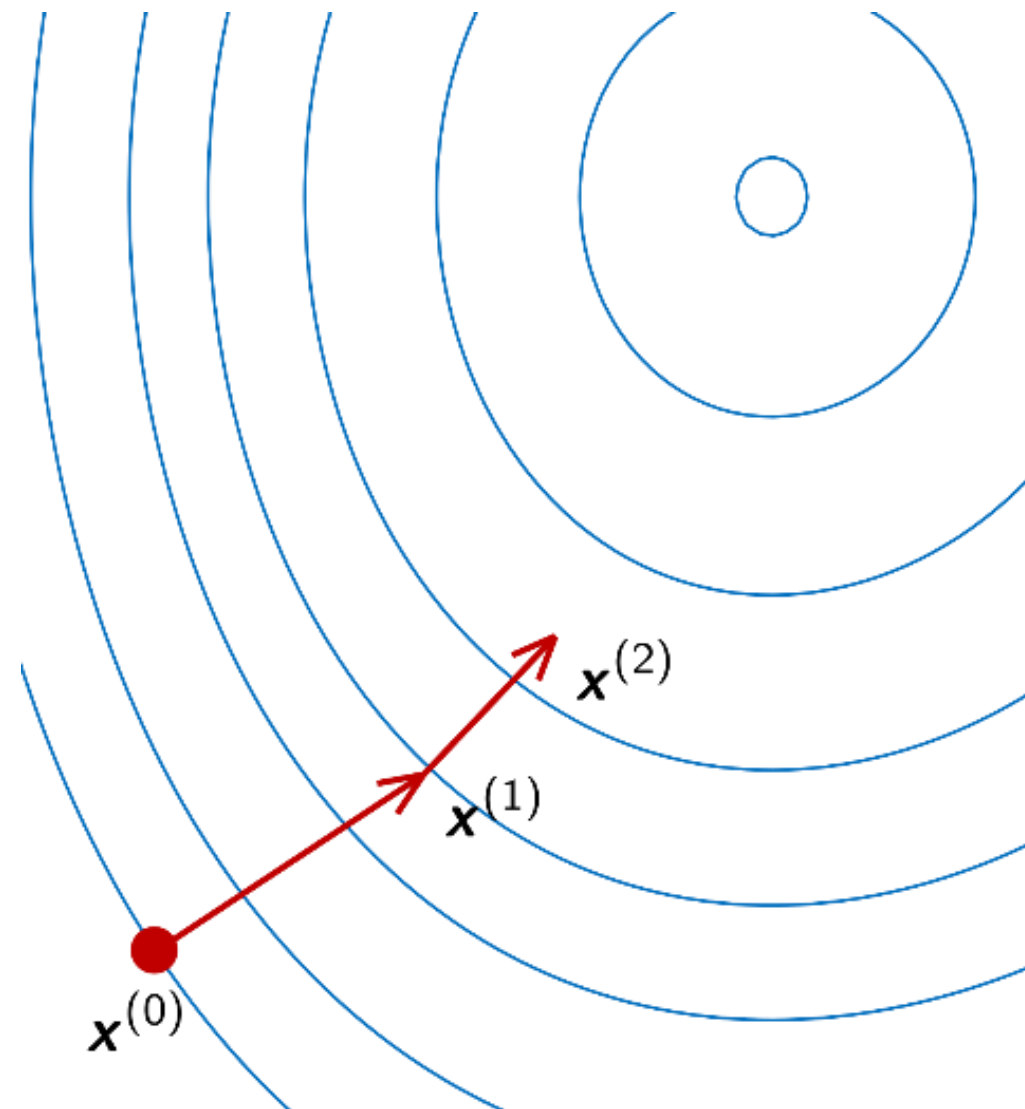
- známe výchozí pozici

## Opakujeme:

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

## Skončíme:

- po vykonaném počtu kroků
- poloha už se nemění (konvergence)



# Metoda největšího spádu (Gradient Descent)

## Začínáme:

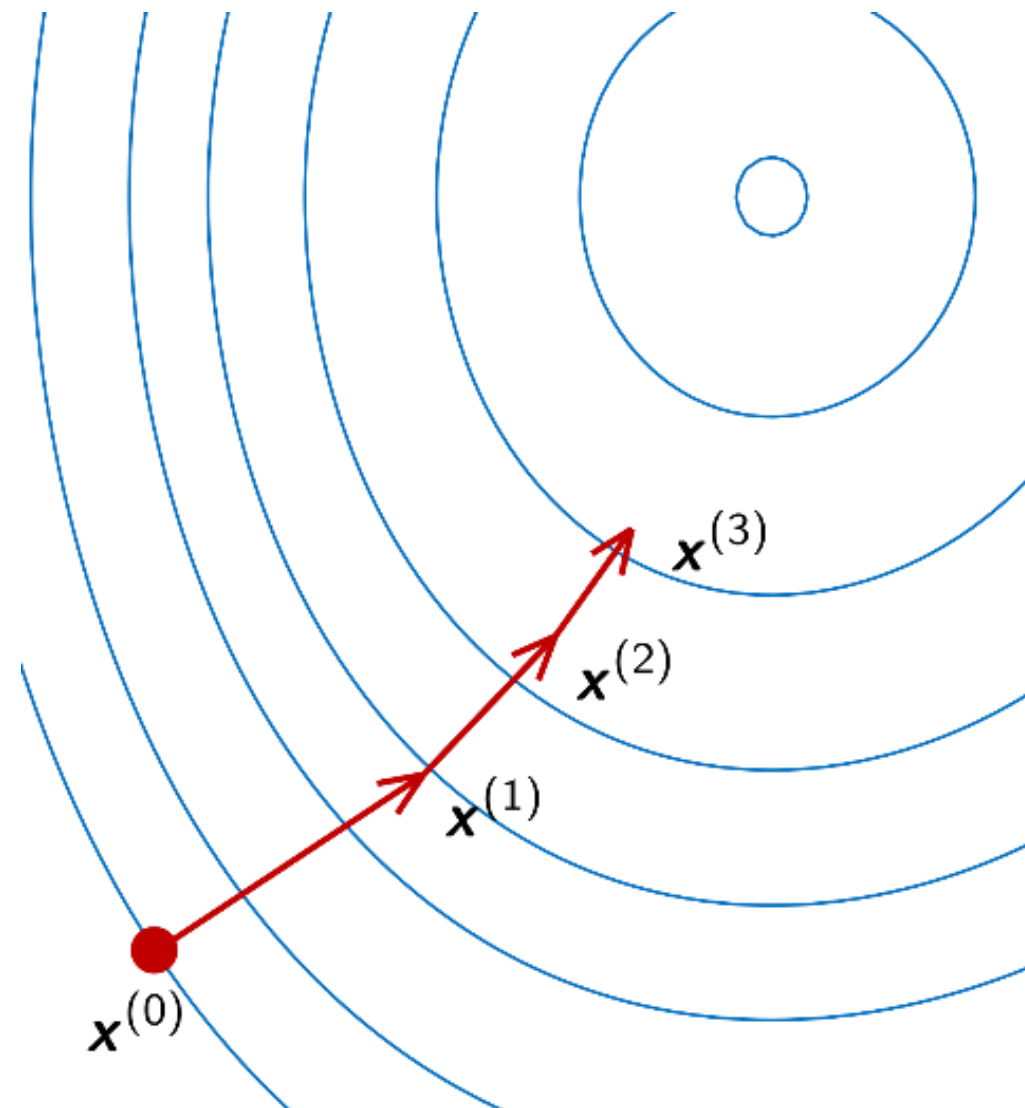
- známe výchozí pozici

## Opakujeme:

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

## Skončíme:

- po vykonaném počtu kroků
- poloha už se nemění (konvergence)





# Metoda největšího spádu (Gradient Descent)

## Začínáme:

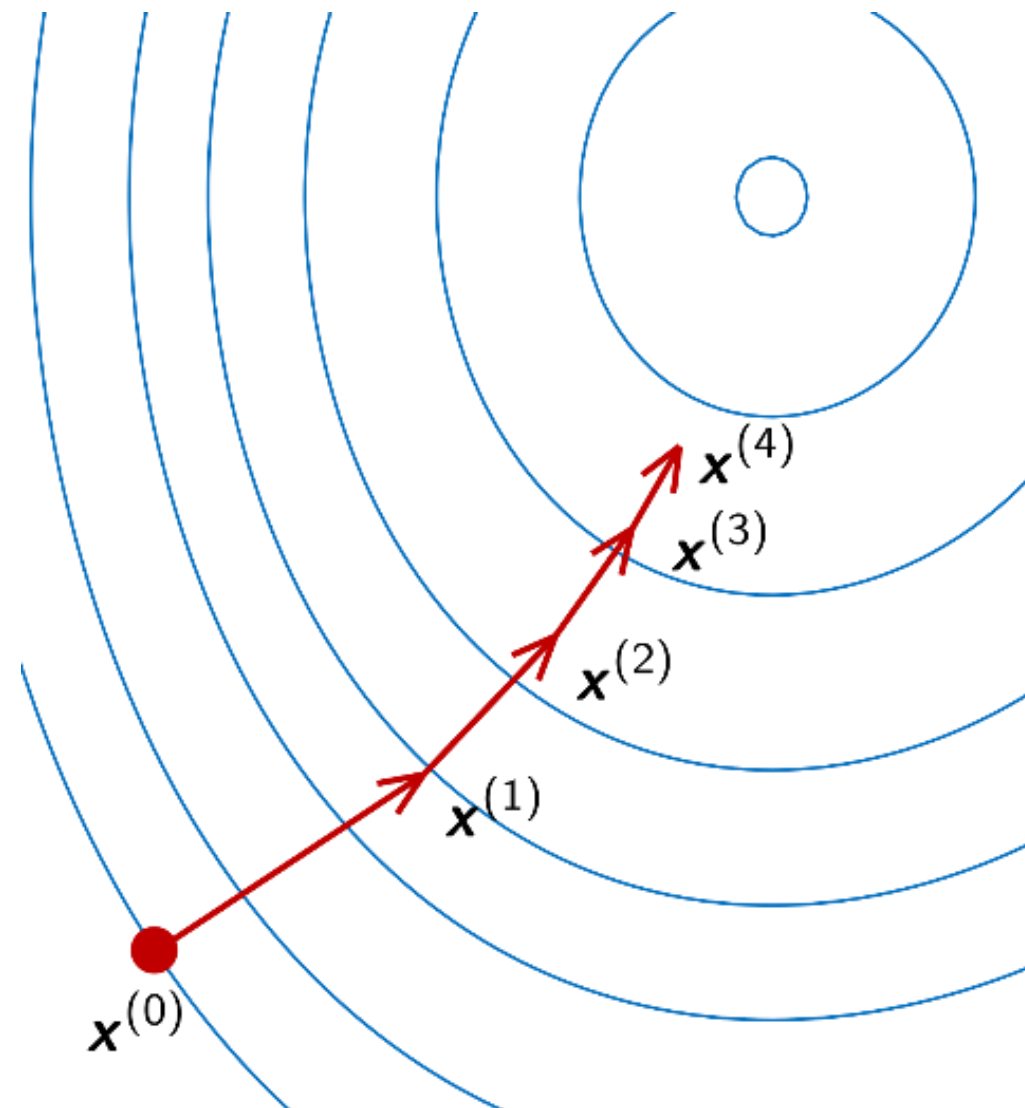
- známe výchozí pozici

## Opakujeme:

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

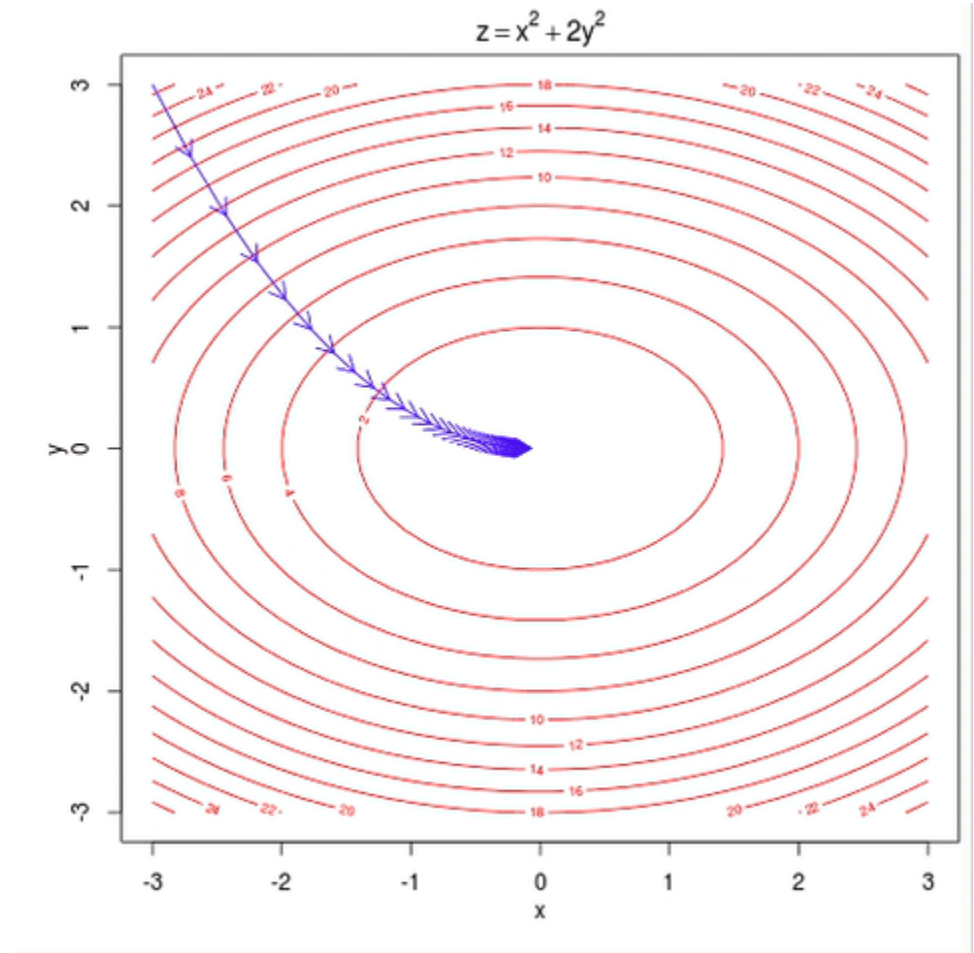
## Skončíme:

- po vykonaném počtu kroků
- poloha už se nemění (konvergence)



# Velikost kroku

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



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

# Kde jsme?

- Zdefinováno kritérium

$$L(\mathbf{w}, b) = \sum_{n=1}^N L_n = \sum_{n=1}^N (-y_n \log\{\sigma(s_n)\} - \dots)$$

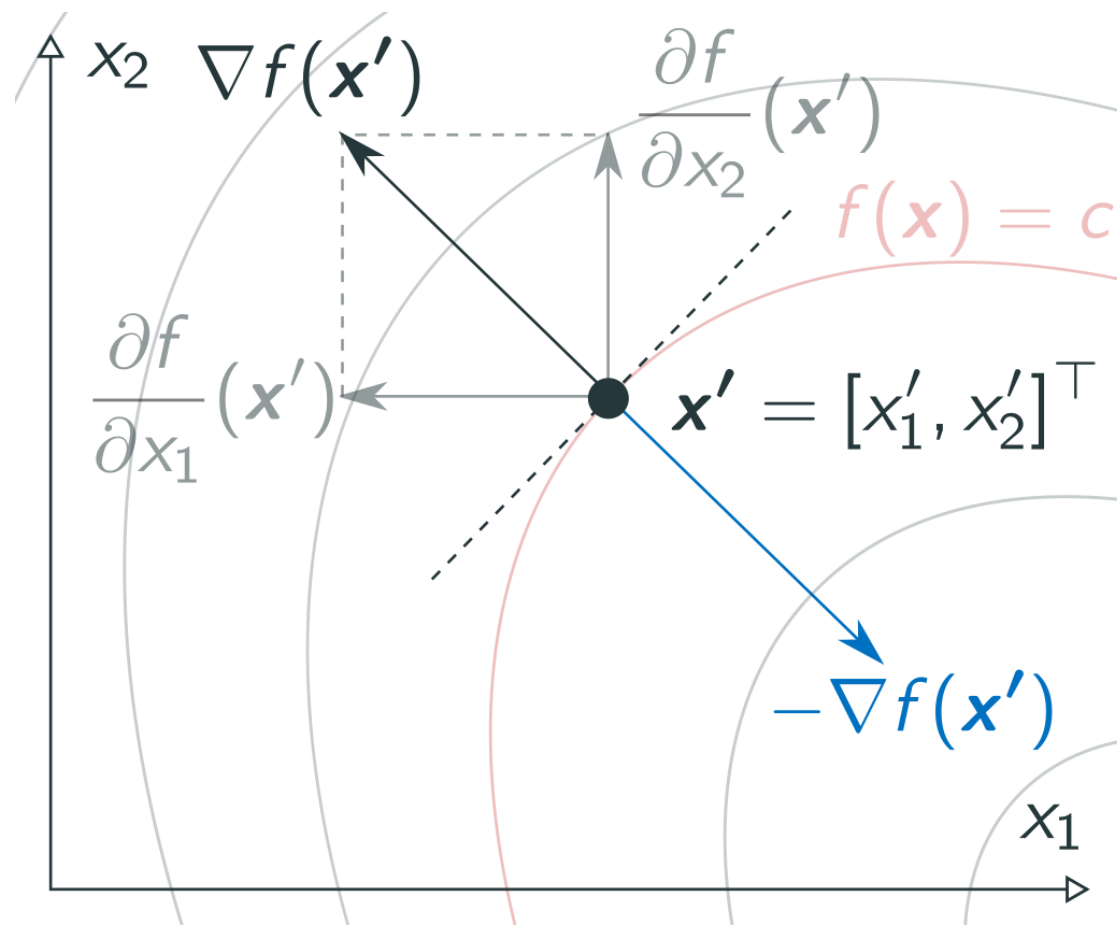
- Proměnné jsou parametry  $\mathbf{w}$  a  $b$
- Optimální  $\mathbf{w}$  a  $b$  jsou takové, které  $L$  minimalizují  $\rightarrow$  hledáme minimum  $L$
- Použijeme metodu největšího spádu  $\rightarrow$  musíme počítat gradient

# Gradient

- Gradient je vektor parciálních derivací

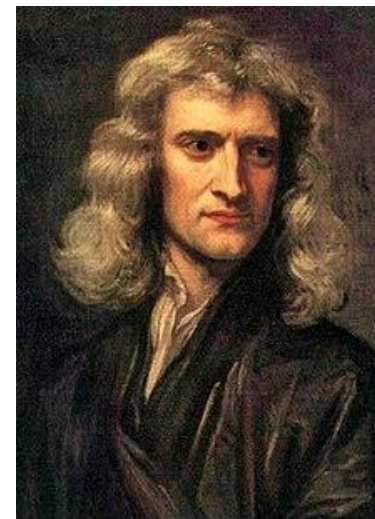
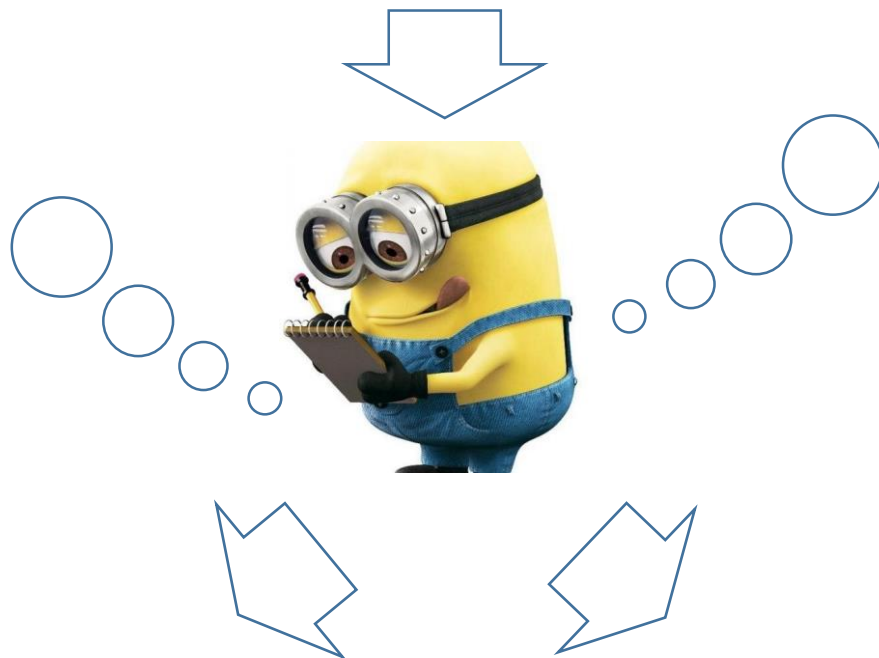
$$\nabla f(\mathbf{x}') = \left[ \frac{\partial f}{\partial x_1}(\mathbf{x}'), \dots, \frac{\partial f}{\partial x_D}(\mathbf{x}') \right]^\top$$

- Pro minimalizaci kritéria musíme derivovat



# Gradient křížové entropie

$$L(\mathbf{w}, b) = \sum_{n=1}^N L_n = \sum_{n=1}^N \left( -y_n \log\{\sigma(\mathbf{w}^\top \mathbf{x}_n + b)\} - (1 - y_n) \log\{1 - \sigma(\mathbf{w}^\top \mathbf{x}_n + b)\} \right)$$



$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{n=1}^N (q_n - y_n) \mathbf{x}_n + 2\lambda \mathbf{w}$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^N (q_n - y_n)$$



# Gradient descent pro logistickou regresi

## Inicializujeme:

- $\mathbf{w}, b$  na náhodné hodnoty

## Opakujeme:

1. předpočítáme výstupní pravděpodobnosti  $q_n$  pro všechny vzorky v trénovací sadě
2. počítáme gradient (suma přes  $n = 1, \dots, N$ )
3. updatujeme s krokem  $\gamma$

## Zastavíme:

- po fixním počtu iterací
- parametry  $\mathbf{w}, b$  se ustálí
- hodnota kritéria  $L(\mathbf{w}, b)$  již delší dobu neklesá

# Gradient descent pro logistickou regresi: poznámky

- Uvedený postup je velmi neefektivní
- Update vždy až po kompletním nasčítání gradientů přes celou trénovací sadu
- Např. ImageNet však cca 14 milionů obrázků
- Vstupní vektory (obrázky) sice předpokládáme nezávislé, jsou si ale podobné v tom smyslu, že pocházejí ze stejného rozdělení pravděpodobnosti
- Možná stačí malý vzorek (**minibatch**), není nutné vidět všechny obrázky
- Takto vznikne tzv. **Minibatch Gradient Descent**
- Pokud pouze jeden vzorek → **Stochastic Gradient Descent (SGD)**
  - [https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent)

# Varianty GD a názvosloví dle velikosti batche

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

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

# SGD pro logistickou regresi

## Inicializujeme:

- $\mathbf{w}, b$  na náhodné hodnoty

## Opakujeme:

1. *navzorkujeme dávku (batch)*
2. předpočítáme výstupní pravděpodobnosti  $q_n$  pro všechny vzorky *v aktuální batchi*
3. posčítáme gradient (suma přes  $n = 1, \dots, B$ )
4. updatujeme s krokem  $\gamma$

## Zastavíme:

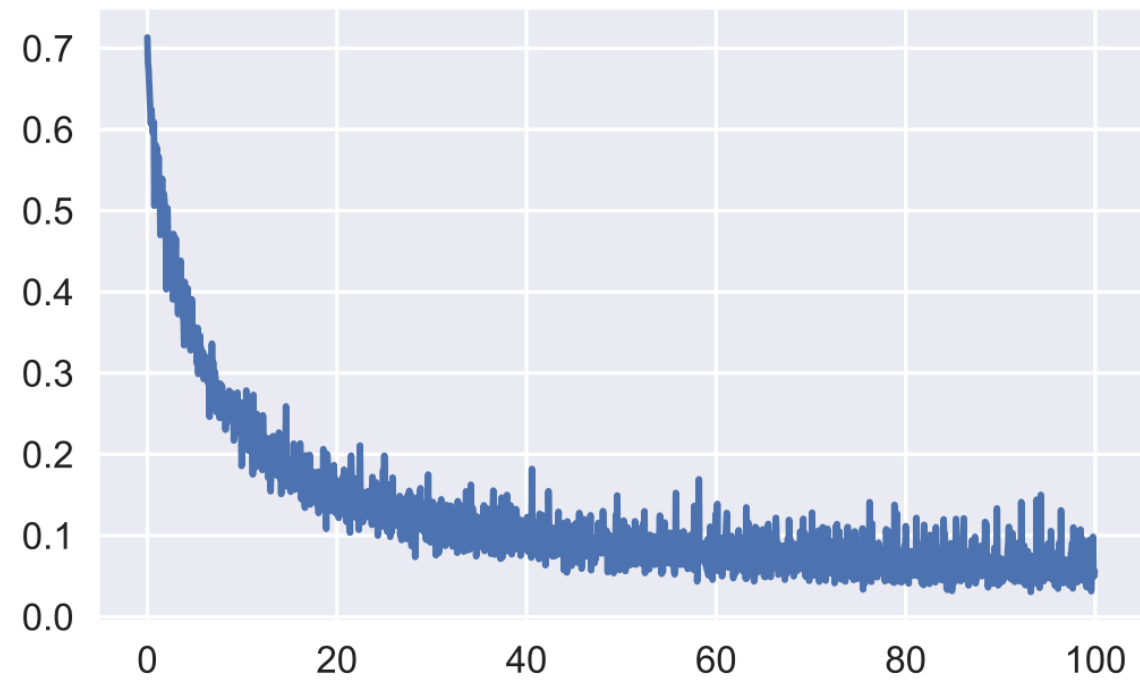
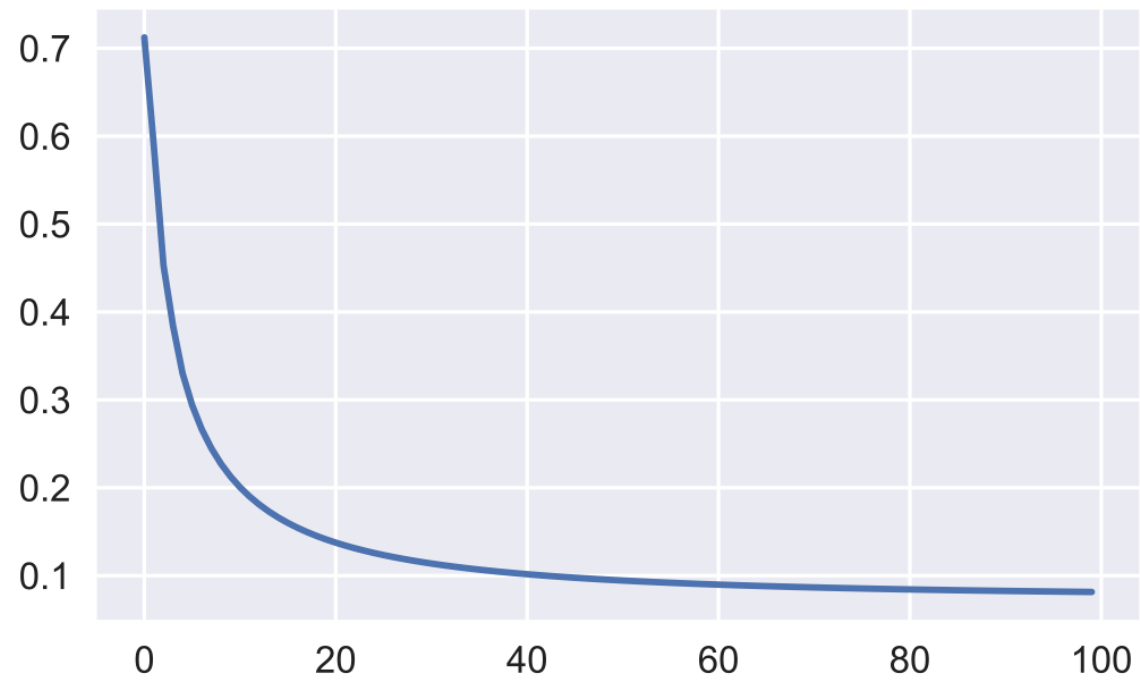
- po fixním počtu iterací
- parametry  $\mathbf{w}, b$  se ustálí
- hodnota kritéria  $L(\mathbf{w}, b)$  již delší dobu neklesá

# GD vs SGD

Gradient descent	Stochastic gradient descent
skutečný gradient	pouze aproximuje gradient
stabilnější konvergence	rychlejší konvergence
konverguje do minima	osciluje kolem minima
náchylnější k upadnutí do lokálního minima	robustnější vůči lokálním minimum
velké nároky na paměť	paměťově neefektivní

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

# GD vs SGD



Více tříd

---

# Binární vs multiclass vs multi-label klasifikace

- Doposud výstupem jediné číslo, např. pravd.  $P(\text{kočka}|\mathbf{x})$ , že na obrázku  $\mathbf{x}$  je kočka
- Opačná pravděpodobnost, že je tam pes  $P(\text{pes}|\mathbf{x}) = 1 - P(\text{kočka}|\mathbf{x})$
- **Binární klasifikace**  $\rightarrow$  počet tříd  $C = 2$ 
  - [https://en.wikipedia.org/wiki/Binary\\_classification](https://en.wikipedia.org/wiki/Binary_classification)
- Pro  $C > 2 \rightarrow$  **multiclass klasifikace**
  - [https://en.wikipedia.org/wiki/Multiclass\\_classification](https://en.wikipedia.org/wiki/Multiclass_classification)
  - např. MNIST číslovky 0, ..., 9
  - CIFAR-10: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- Pozn.: neplést s multi-label klasifikací
  - [https://en.wikipedia.org/wiki/Multi-label\\_classification](https://en.wikipedia.org/wiki/Multi-label_classification)
  - více tříd najednou, ale nezávisle na sobě – např. je na obr. kočka? pes také? a žába rovněž? ...
  - tagging



# Rozšíření z binární na multiclass klasifikaci

## 1. One-vs-rest (one-vs-all)

- $C$  samostatných klasifikátorů, z nichž každý diskriminuje jednu ze tříd vůči ostatním
- např. kočka ("1") vs ostatní ("0"), pes ("1") vs ostatní ("0"), ...
- Pro  $C = 10 \rightarrow 10$  klasifikátorů
- Vyhrává třída s nejvyšším skóre/pravděpodobností

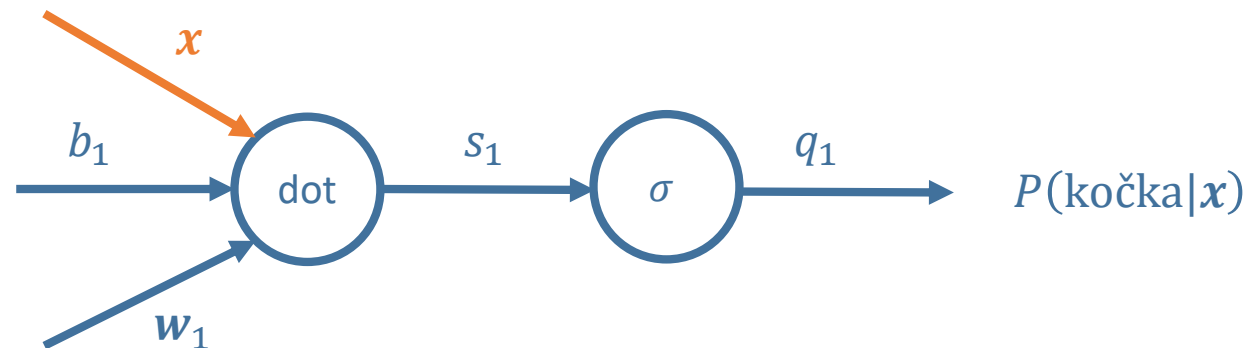
## 2. One-vs-one (all-vs-all)

- $C(C - 1)/2$  samostatných klasifikátorů pro každou dvojici tříd
- Např. kočka vs pes, kočka vs žába, pes vs žába, ...
- Pro  $C = 10 \rightarrow 45$  klasifikátorů
- Pro  $C = 1000$  (ImageNet)  $\rightarrow 499500 \approx 0.5 \cdot 10^6$  klasifikátorů!
- Vyhrává třída s nejvyšším počtem "výher z duelů"

## 3. Reformulace úlohy

- jeden klasifikátor, ale výstupem bude  $C$  pravděpodobností pro každou třídu současně (paralelně)
- vyhrává třída s nejvyšším skóre/pravděpodobností
- $\rightarrow$  tudy vede cesta!

# Binární logistická regrese

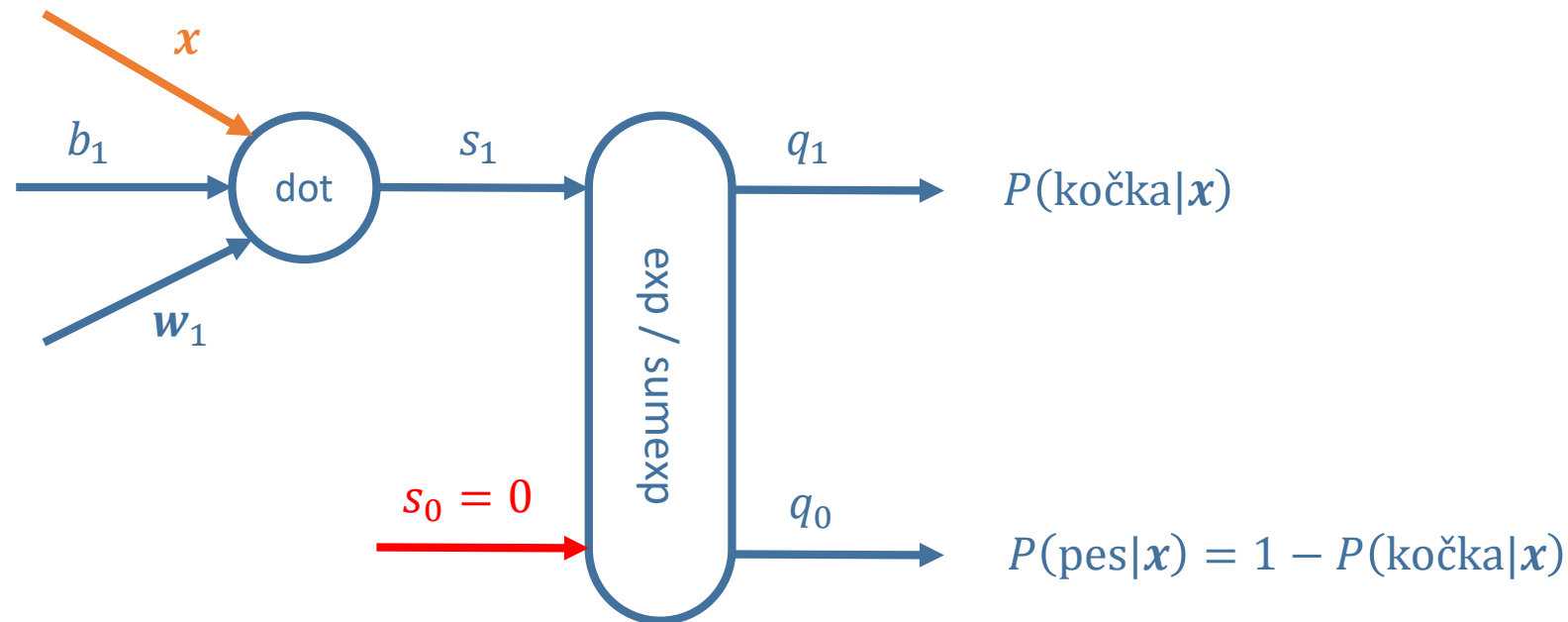


$$\longrightarrow P(\text{pes}|\mathbf{x}) = 1 - P(\text{kočka}|\mathbf{x})$$

$$P(\text{kočka}|\mathbf{x}) = \frac{1}{1 + e^{-s_1}} = \frac{e^{s_1}}{1 + e^{s_1}} = \frac{e^{s_1}}{e^0 + e^{s_1}}$$

$$P(\text{pes}|\mathbf{x}) = 1 - P(\text{kočka}|\mathbf{x}) = 1 - \frac{e^{s_1}}{e^0 + e^{s_1}} = \frac{e^0}{e^0 + e^{s_1}}$$

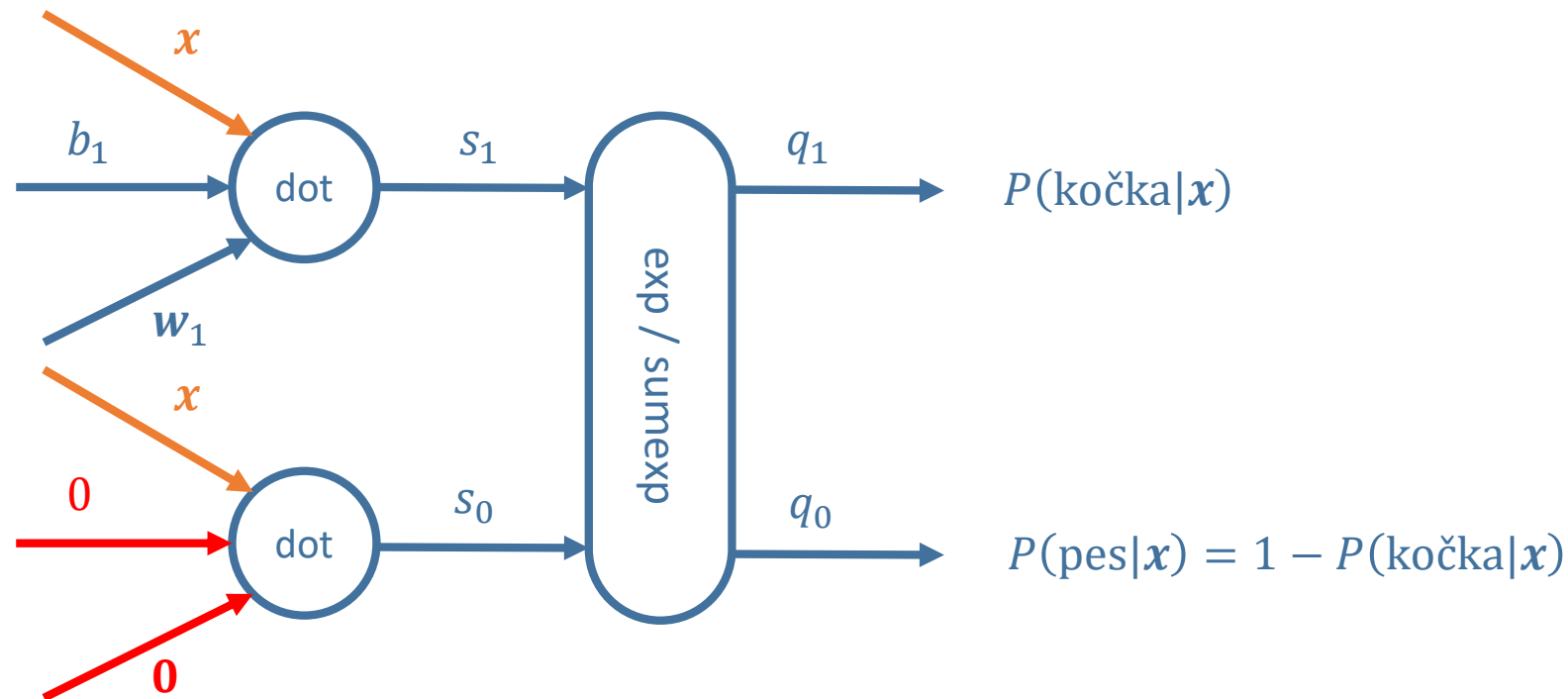
# Binární logistická regrese ... stejné



$$P(\text{kočka}|\mathbf{x}) = \frac{1}{1 + e^{-s_1}} = \frac{e^{s_1}}{1 + e^{s_1}} = \frac{e^{s_1}}{e^0 + e^{s_1}}$$

$$P(\text{pes}|\mathbf{x}) = 1 - P(\text{kočka}|\mathbf{x}) = 1 - \frac{e^{s_1}}{e^0 + e^{s_1}} = \frac{e^0}{e^0 + e^{s_1}}$$

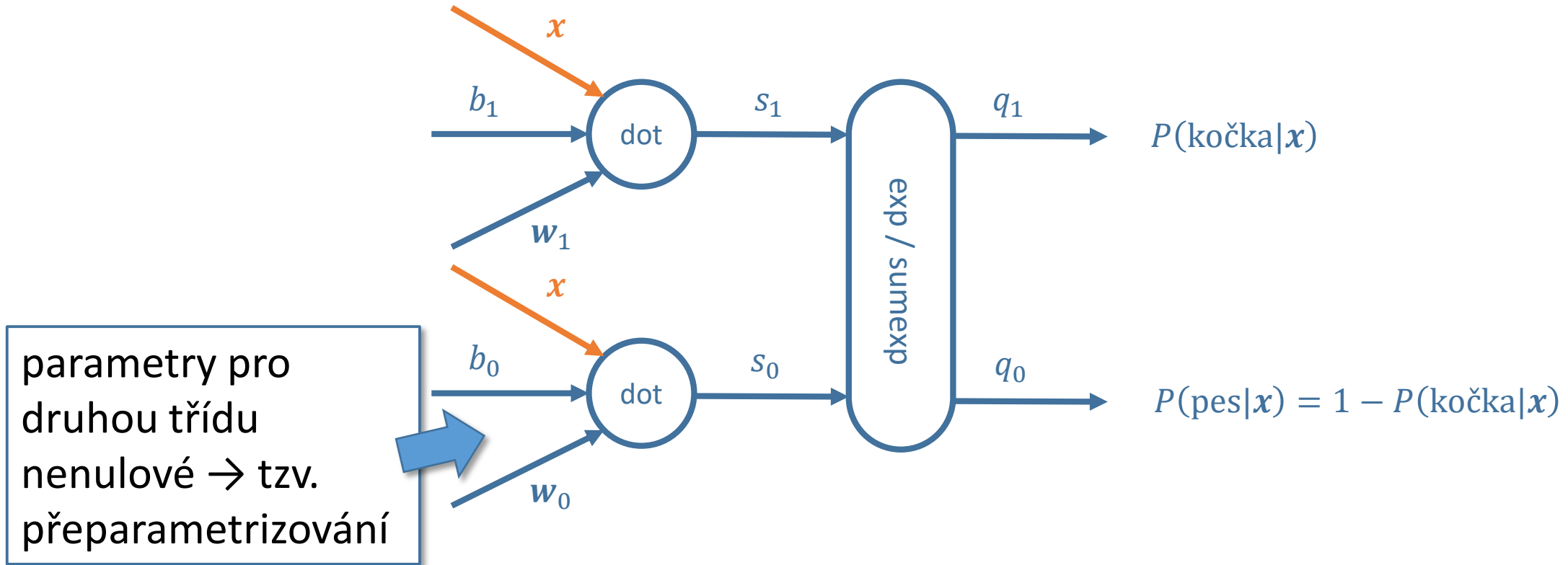
# Binární logistická regrese ... stále stejné



$$P(\text{kočka}|\mathbf{x}) = \frac{1}{1 + e^{-s_1}} = \frac{e^{s_1}}{1 + e^{s_1}} = \frac{e^{s_1}}{e^0 + e^{s_1}}$$

$$P(\text{pes}|\mathbf{x}) = 1 - P(\text{kočka}|\mathbf{x}) = 1 - \frac{e^{s_1}}{e^0 + e^{s_1}} = \frac{e^0}{e^0 + e^{s_1}}$$

# Multiclass logistická regrese



$$P(\text{kočka}|\mathbf{x}) = \frac{e^{s_1}}{e^{s_0} + e^{s_1}}$$

$$P(\text{pes}|\mathbf{x}) = \frac{e^{s_0}}{e^{s_0} + e^{s_1}}$$

podmínka na součet pravděpodobností zachována:

$$P(\text{kočka}|\mathbf{x}) + P(\text{pes}|\mathbf{x}) = \frac{e^{s_1} + e^{s_0}}{e^{s_0} + e^{s_1}} = 1$$

# Softmax

- Myšlenku lze rozšířit na libovlnný počet tříd  $C \geq 2$
- Blok exp / sumexp se označuje jako softmax

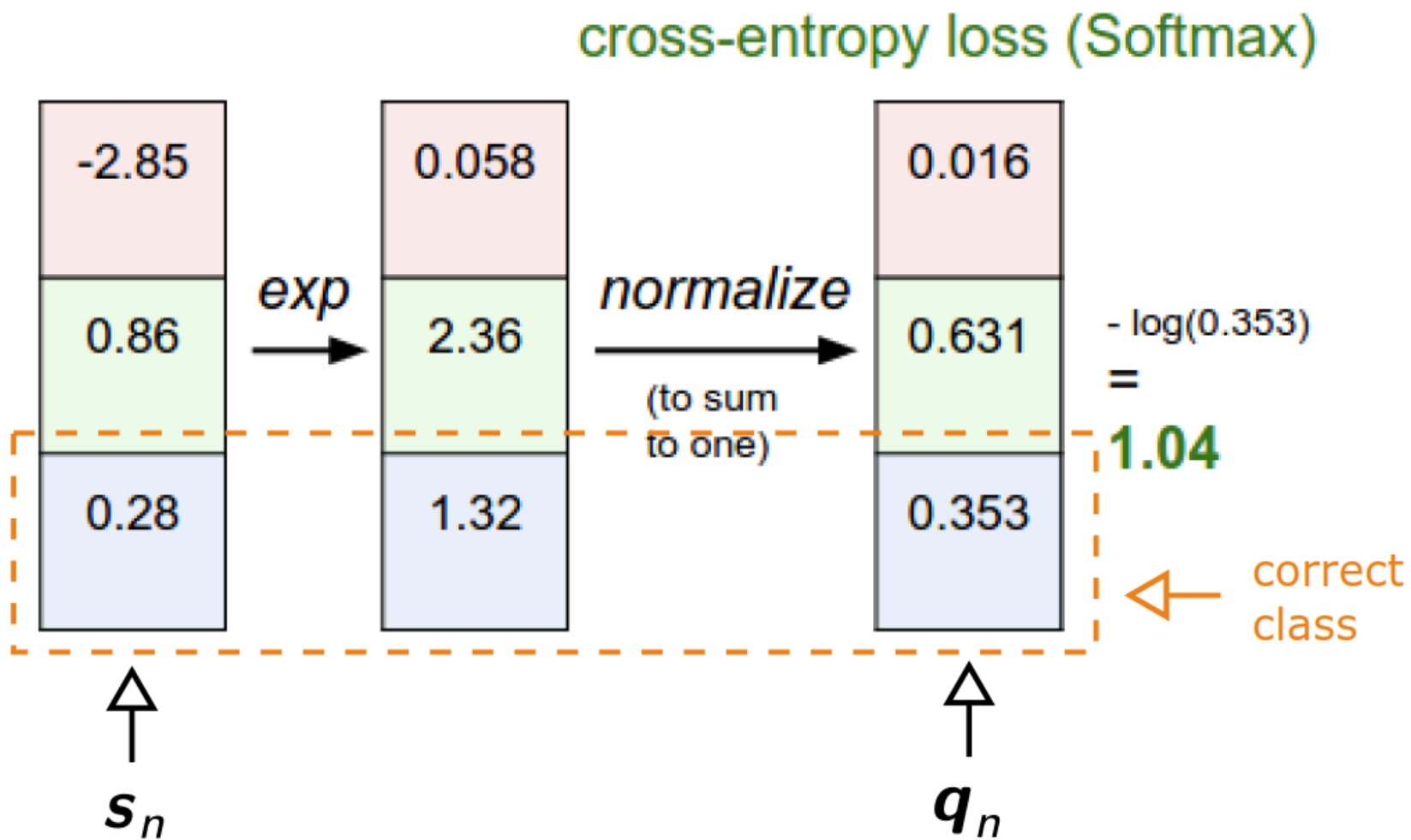
$$q_i = P(\text{třída\_}i|\mathbf{x}) = \frac{e^{s_i}}{\sum_{c=1}^C e^{s_c}}$$

- Výstupem  $C$ -dimezionální vector pravděpodobností jednotlivých tříd

$$\mathbf{q} = [q_1, q_2, \dots, q_C]^\top$$

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

# Softmax příklad



# Multiclass cross entropy

- Na trénování se oproti binární variantě téměř nic nemění

$$L_n = - \sum_{c=1}^C p_{nc} \log q_{nc}$$

- kde

$$\mathbf{p}_n = [p_{n1}, \dots, p_{nC}]^\top \quad \dots \text{požadované rozdělení (ground truth)}$$

$$\mathbf{q}_n = [q_{n1}, \dots, q_{nC}]^\top \quad \dots \text{výstup klasifikátoru}$$

jsou vektory, na které nahlížíme jako na diskrétní rozdělení

- $\rightarrow$  cross entropy = minimalizace rozdílu mezi dvěma rozděleními



# Binární vs multiclass entropy pro $C = 2$

## binární CE

$$L_n = -y_n \log q_{n1} - (1 - y_n) \log(1 - q_{n1})$$

## multiclass CE

$$L_n = -p_{n1} \log q_{n1} - p_{n2} \log q_{n2}$$

- U obou platí, že nenulový je vždy pouze jeden ze dvou členů v součtu jelikož

$$\sum_c p_{nc} = 1 \quad \Rightarrow \quad p_{n2} = 1 - p_{n1}$$

- Jediný rozdíl: u binární explicitně dopočítáváme druhý člen jako doplněk do jedničky, zatímco u multiclass mezi členy nerozlišujeme

# One hot encoding


- Pro více tříd je  $y_n$  celé číslo, tj.  $y_n \in \{1, \dots, C\}$
- Pokud  $C = 5 \rightarrow$  požadované rozdělení pak je

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

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

# Softmax + cross entropy

- V cross entropy *pro klasifikaci* tedy bude aktivní vždy pouze jeden člen:

$$-L_n = \sum_{c=1}^C p_{nc} \log q_{nc} = \log(q_{ny_n}) = \log \left( \frac{e^{s_{ny_n}}}{\sum_{c=1}^C e^{s_{nc}}} \right)$$


což je zápis, jenž najdeme např. v [poznámkách cs231](#)

- Pokud rozepíšeme logaritmus zlomku, dostaneme druhou variantu

$$L_n = -\log \left( \frac{e^{s_{ny_n}}}{\sum_{c=1}^C e^{s_{nc}}} \right) = -s_{ny_n} + \log \sum_{c=1}^C e^{s_{nc}}$$

- Softmax + CE tedy maximalizuje poměr pravděpodobnosti požadované třídy vůči součtu všech ostatních a to pro každý vzorek

# Minimalizace softmax cross entropy

- Celkově tedy kritérium je

$$L(\mathbf{W}, \mathbf{b}) = \lambda \|\mathbf{W}^2\| - \sum_{n=1}^N \sum_{c=1}^C p_{nc} \log q_{nc}$$

kde

$$\mathbf{q}_n = [q_{n1}, \dots, q_{nC}]^\top = \text{Softmax}(\mathbf{W} \mathbf{x}_n + \mathbf{b})$$

- Parametry jsou matice a vektor

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \vdots \\ \mathbf{w}_C^\top \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{C1} & w_{C2} & \dots & w_{CD} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_C \end{bmatrix}$$

- Gradienty:

$$\frac{\partial L}{\partial \mathbf{w}_c} = \sum_{n=1}^N 2\lambda \mathbf{w}_c + (q_{nc} - p_{nc}) \mathbf{x}_n$$

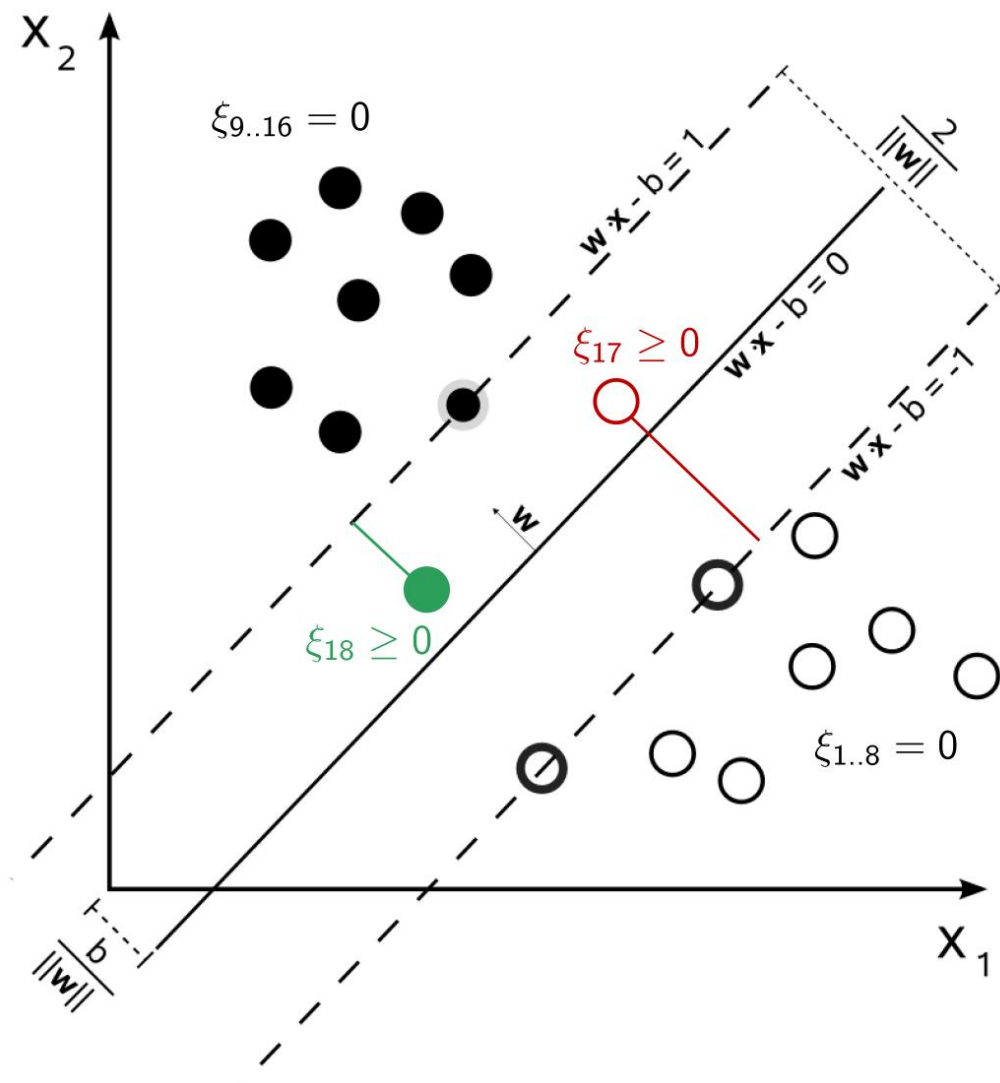
$$\frac{\partial L}{\partial b_c} = \sum_{n=1}^N (q_{nc} - p_{nc})$$

# Support Vector Machine

---

# Support Vector Machine (SVM)

- Nepravděpodobnostní model
- Max-margin klasifikátor: hledá takovou dělicí nadplochu, která je co nejdále od obou tříd
- Pokud třídy nejsou lineárně separovatelné, zavádí se tzv. slack variables  $\xi_n \geq 0$
- Pro některé body tedy podmínka nemusí být splněná



obrázek: [https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)

# Hinge loss

- Soft margin SVM loss se slack variables

$$\begin{aligned} &\underset{\mathbf{w}, b, \xi_n}{\text{minimize}} && \|\mathbf{w}\|^2 + \lambda' \sum_{i=1}^N \xi_n \\ &\text{subject to} && y_n s_n \geq 1 - \xi_n \\ &&& \xi_n \geq 0 \end{aligned}$$

- Podmínky lze sloučit do jednoho výrazu

$$\xi_n = \max(0, 1 - y_n s_n)$$

a dosadit kritéria  $\rightarrow$  vznikne **hinge loss**

- Člen  $\|\mathbf{w}\|^2$  funguje jako regularizace

# SVM jako hinge loss

- U SVM tedy minimalizujeme

$$L(\mathbf{w}, b) = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^N \max(0, 1 - y_n s_n)$$

kde  $\lambda \propto 1/\lambda'$

- Rozdíl oproti logistické regresi je tedy ve zvoleném kritériu:

logistická regrese = cross entropy

SVM = hinge loss

a v tom, že  $y_n \in \{-1, +1\}$  (u binární logistické regrese je  $y_n \in \{0, 1\}$ )



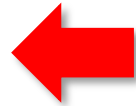
# SVM pro více tříd

## 1. Strategie nezávislé na klasifikátoru

- one-vs-rest (one-vs-all)
- one-vs-one (all-vs-all)


## 2. Reformulace úlohy

- Cramer-Singer
- Weston-Watkins
- Structured SVM
- DAGSVM
- ...



# Weston-Watkins multiclass SVM

- Upravuje hinge loss na

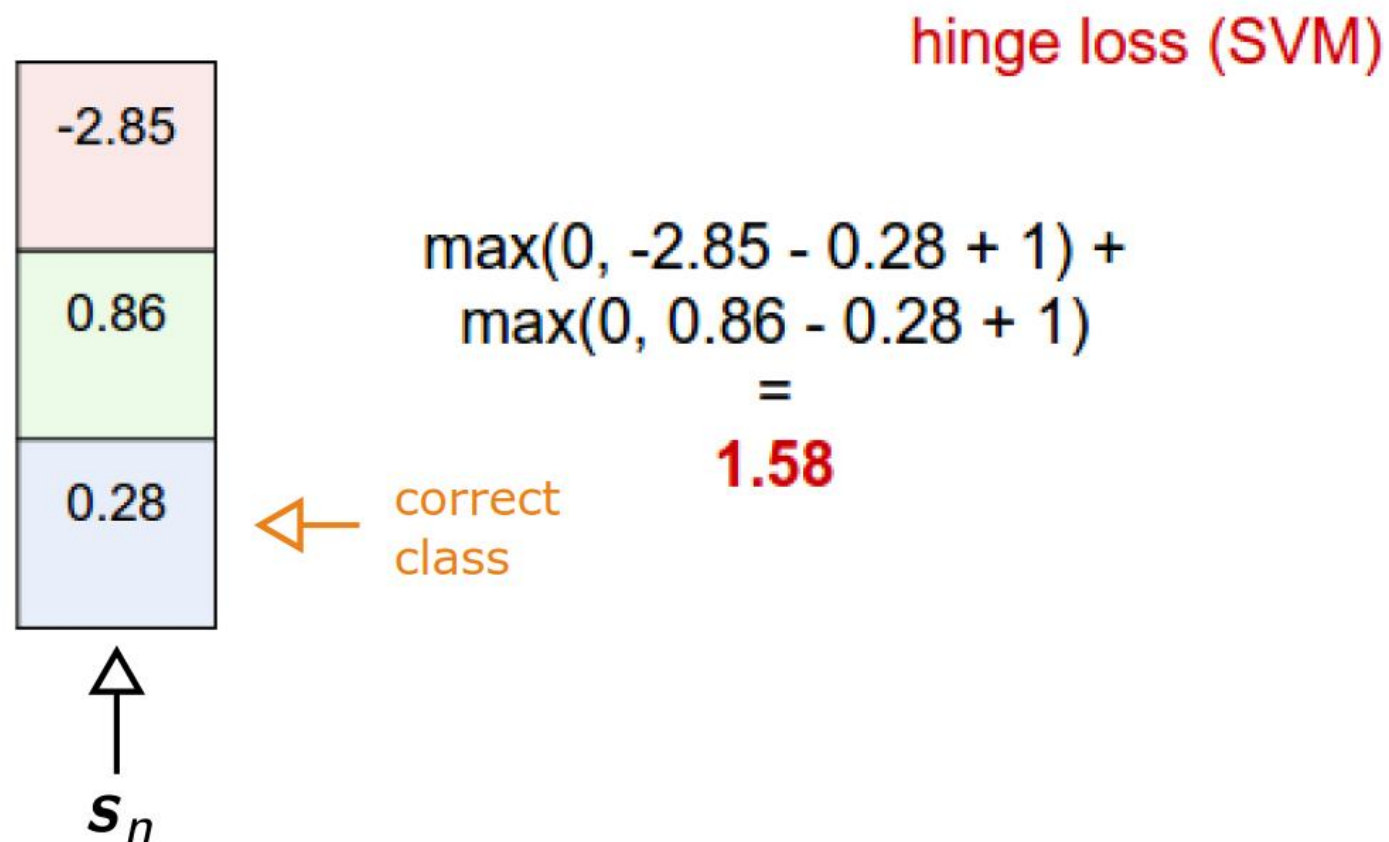

$$\xi_n = \sum_{c \neq y_n} \max(0, 1 + s_{nc} - s_{ny_n})$$

- Při více třídách je skóre vektor
- Aby  $\xi_n \rightarrow 0$ , musí:
  - skóre požadované třídy  $s_{ny_n}$  být co nejvyšší
  - skóre všech  $c = 1, \dots, C$  ostatních tříd  $s_{nc}$  co nejnižší

binární varianta

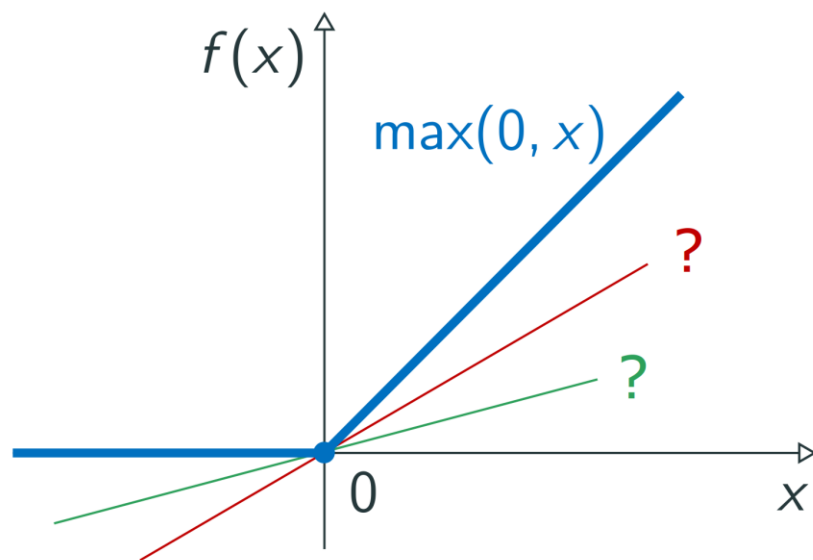
$$\xi_n = \max(0, 1 - y_n s_n)$$

# Weston-Watkins multiclass SVM příklad



# Subgradient

- Funkce  $\max(0, x)$  není diferencovatelná
- Problém “bod zlomu” v nule:



- Řeší tzv. subgradient  $\rightarrow$  prostě vybereme jednu z možných variant
- Např. v nule bude gradient nula
- (Sub)gradient tedy může být:

$$\frac{\partial}{\partial x} \max(x) = \begin{cases} 0 & \text{pokud } x \leq 0 \\ 1 & \text{pokud } x > 0 \end{cases}$$

# (Sub)gradient hinge kritéria

$$\xi_n = \sum_{c \neq y_n} \max(0, \underbrace{1 + s_{nc} - s_{ny_n}}_{\zeta_{nc}})$$

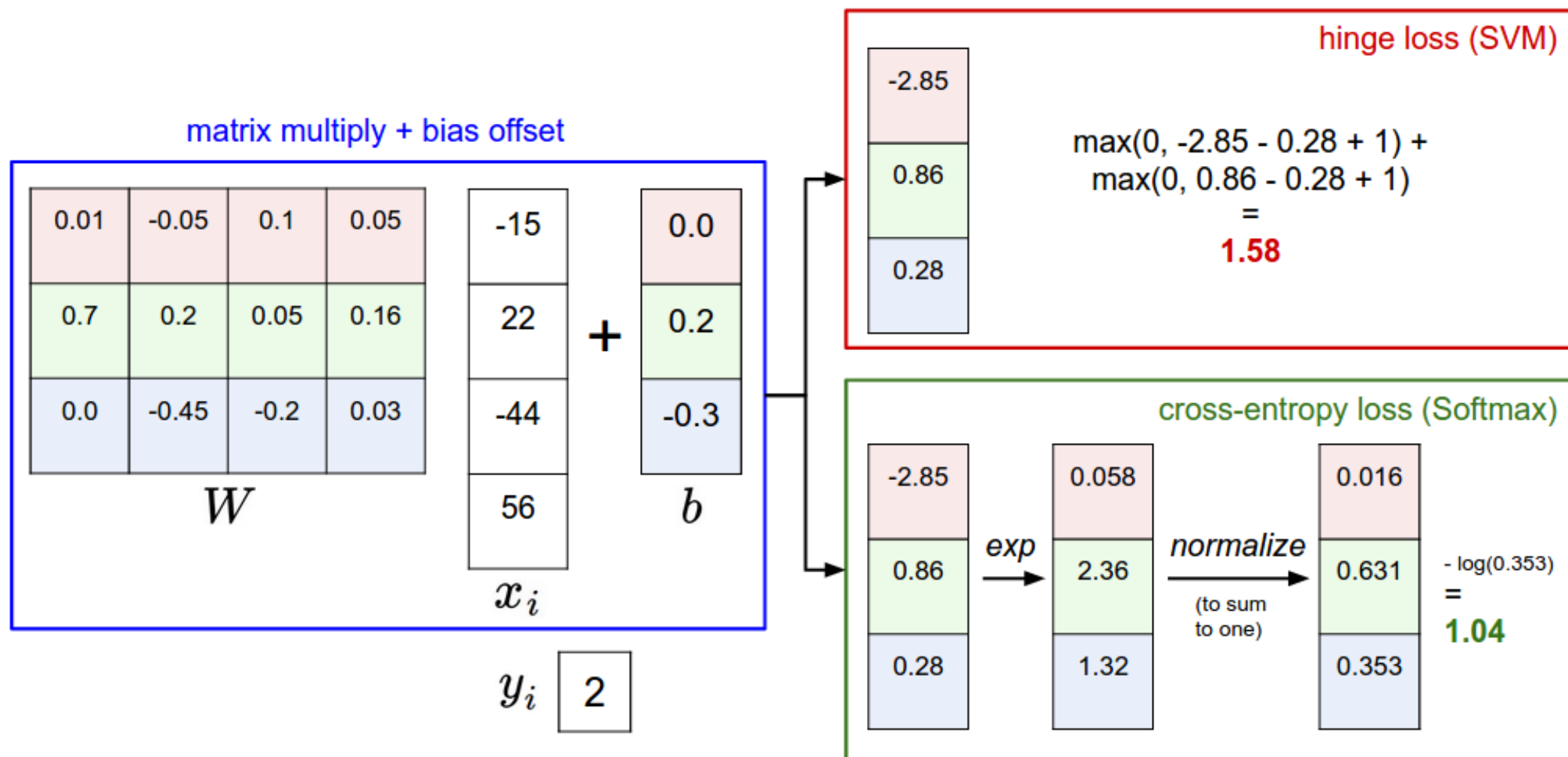
- Všimněme si, že obě skóre  $s_i$  závisí pouze na  $i$ -tém řádku  $\mathbf{W}$  a  $\mathbf{b}$
- Gradient na  $i$ -tý řádek  $\mathbf{W}$ :

$$\frac{\partial L_n}{\partial \mathbf{w}_i} = \begin{cases} - \sum_{c \neq y_n} \mathbb{1}(\zeta_{nc} > 0) \mathbf{x}_n & \text{pokud } i = y_n \\ \mathbb{1}(\zeta_{ni} > 0) \mathbf{x}_n & \text{pokud } i \neq y_n \end{cases}$$

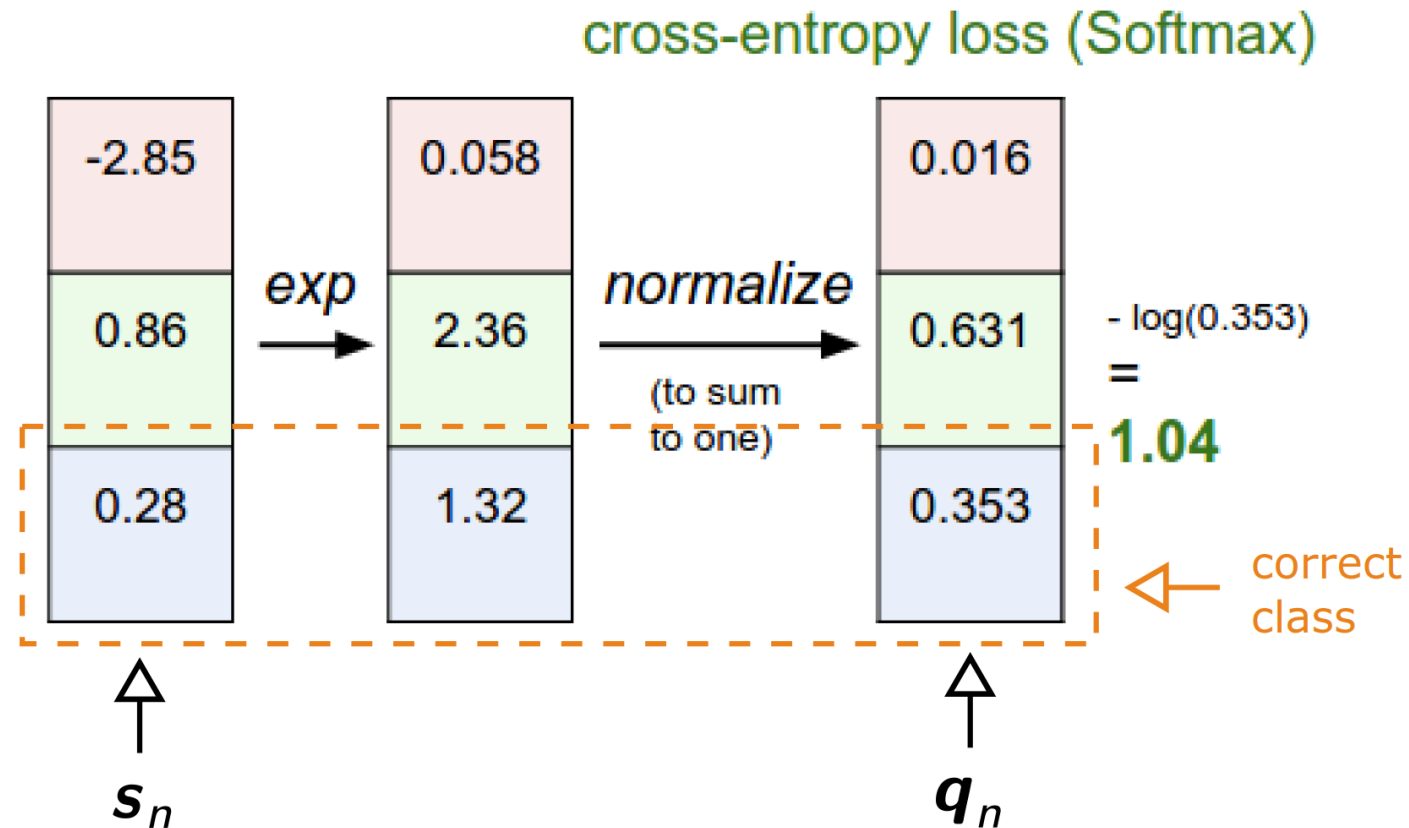
kde  $\mathbb{1}(\text{podmínka}) = 1$ , pokud je podmínka splněna, jinak 0

- Pro biasy podobně, pouze bez  $\mathbf{x}_n$

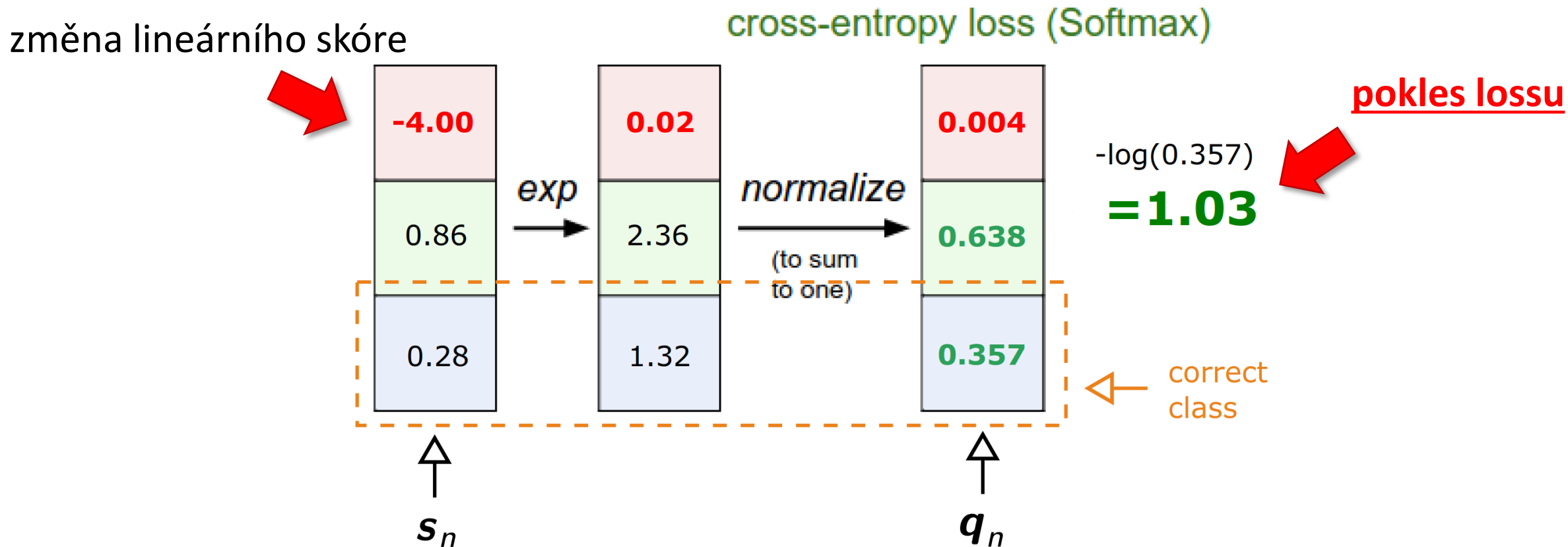
# Cross entropy a hinge loss



# Citlivost cross entropy na změnu skóre

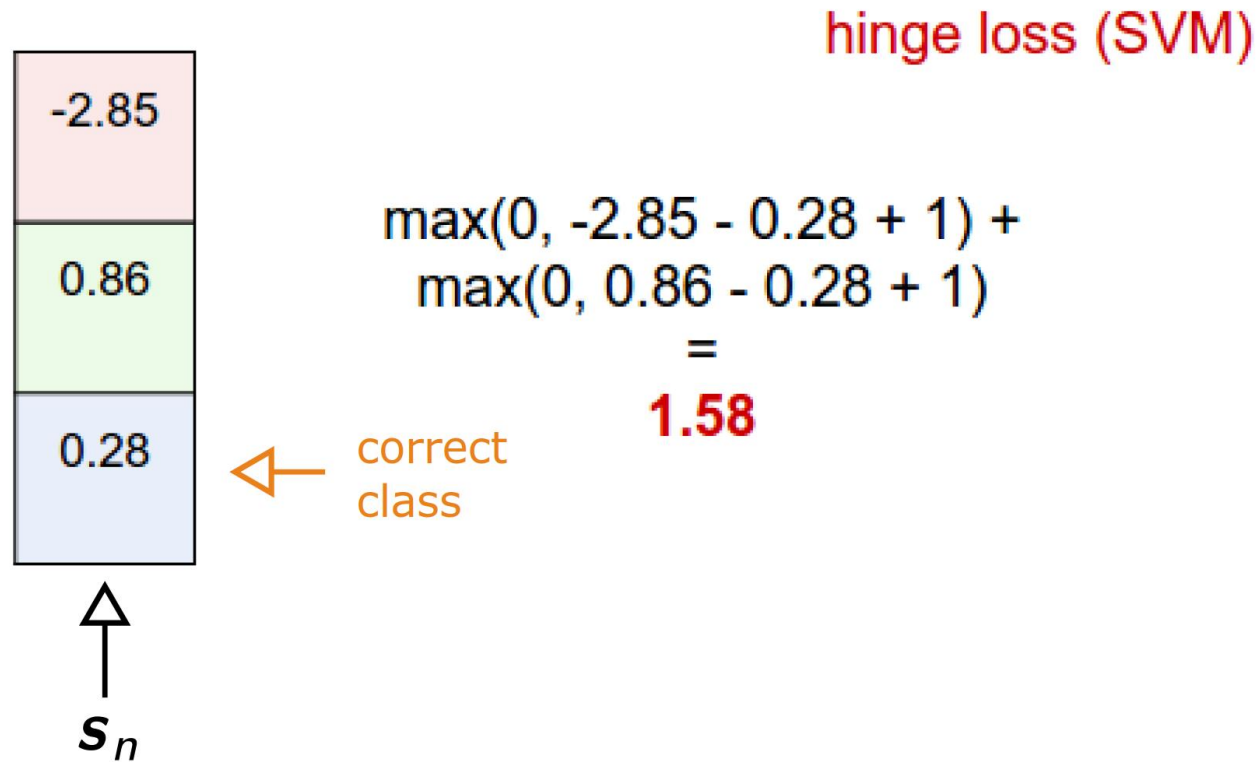


# Citlivost cross entropy na změnu skóre



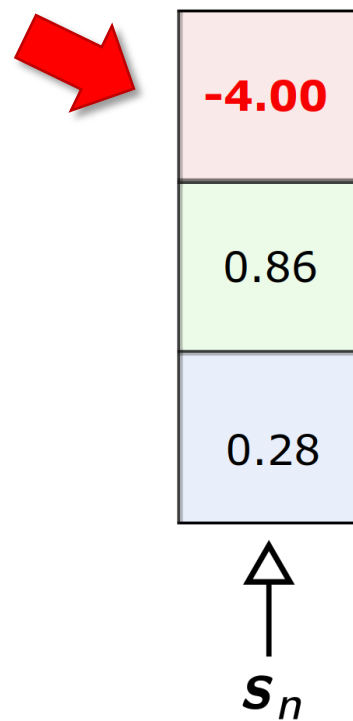


# Citlivost hinge lossu na změnu skóre



# Citlivost hinge lossu na změnu skóre

změna lineárního skóre



correct  
class

hinge loss (SVM)

$$\begin{aligned} &\max(0, -4.00 - 0.28 + 1) + \\ &\max(0, 0.86 - 0.28 + 1) \\ &= \\ &1.58 \end{aligned}$$

loss nezměněn

# Cross entropy vs hinge loss

- SVM zahrnuje vnitřní “regularizaci”: pokud je hinge podmínka u bodu nějakého splněna, kritérium zde má nulovou hodnotu a tedy i přírůstek gradientu od tohoto bodu je nulový
- Logistická regrese naopak bez explicitní regularizace nikdy nekonverguje, skóre se donekonečna snaží zlepšit
- Pro účely klasifikace obě kritéria přibližně stejně dobrá
- Díky robustnosti vůči outlierům na menších datasetech výkonnější spíše SVM
- Overhead způsobený funkcí softmax je především u hlubokých sítí zanedbatelný

# Shrnutí

- Diskriminativní klasifikace je vlastně jen minimalizace funkce
- Funkce kvantifikuje, jak moc špatný náš klasifikátor je → tzv. loss či kritérium
- Proměnná, vůči které minimalizujeme, jsou tedy parametry klasifikátoru
- Funkce může být libovolně složitá, avšak mělo by být snadné spočítat její gradient
- Díky tomu můžeme minimalizovat pomocí metody největšího spádu (GD)
- Gradient není nutné počítat úplně, efektivnější je aproximace a častější update (SGD)
- Logistická regrese a SVM jsou obojí lineární klasifikátory
- Liší se pouze kritériem

# Shrnutí: trénování pomocí SGD

## Inicializujeme:

- $\mathbf{w}, b$  na náhodné hodnoty

## Opakujeme:

1. navzorkování dávky (batch)
2. forward + cache
3. backward (gradient)
4. update s krokem  $\gamma$



různé pro LR a SVM

## Zastavíme:

- po fixním počtu iterací
- parametry  $\mathbf{w}, b$  se ustálí
- hodnota kritéria  $L(\mathbf{w}, b)$  již delší dobu neklesá