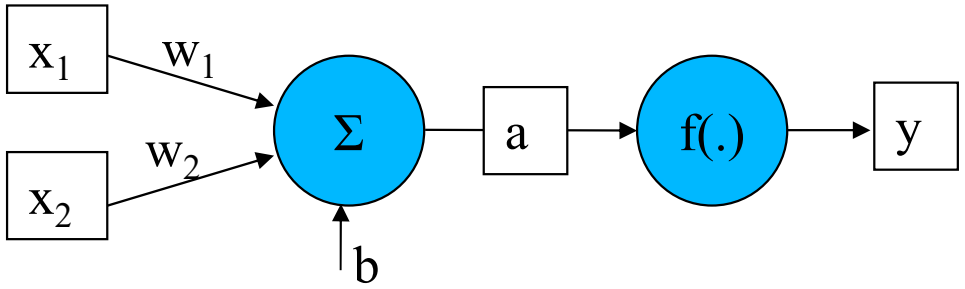
Klasifikace a rozpoznávání

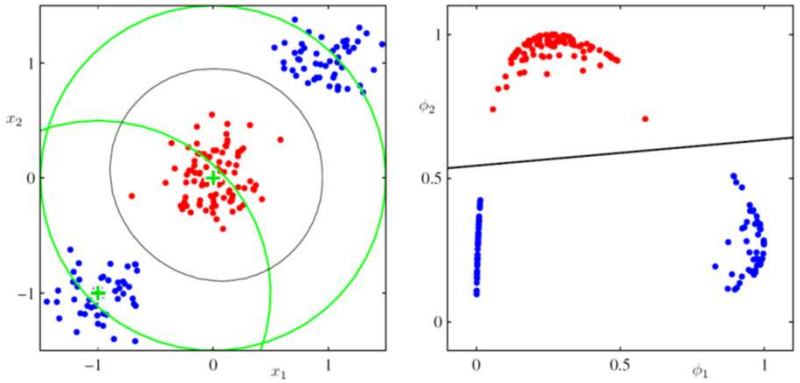
Umělé neuronové sítě
a Support Vector Machines

Lineární klasifikátory



$$y = \mathrm{f}\left(\mathbf{w}^T \mathbf{x} + b\right)$$

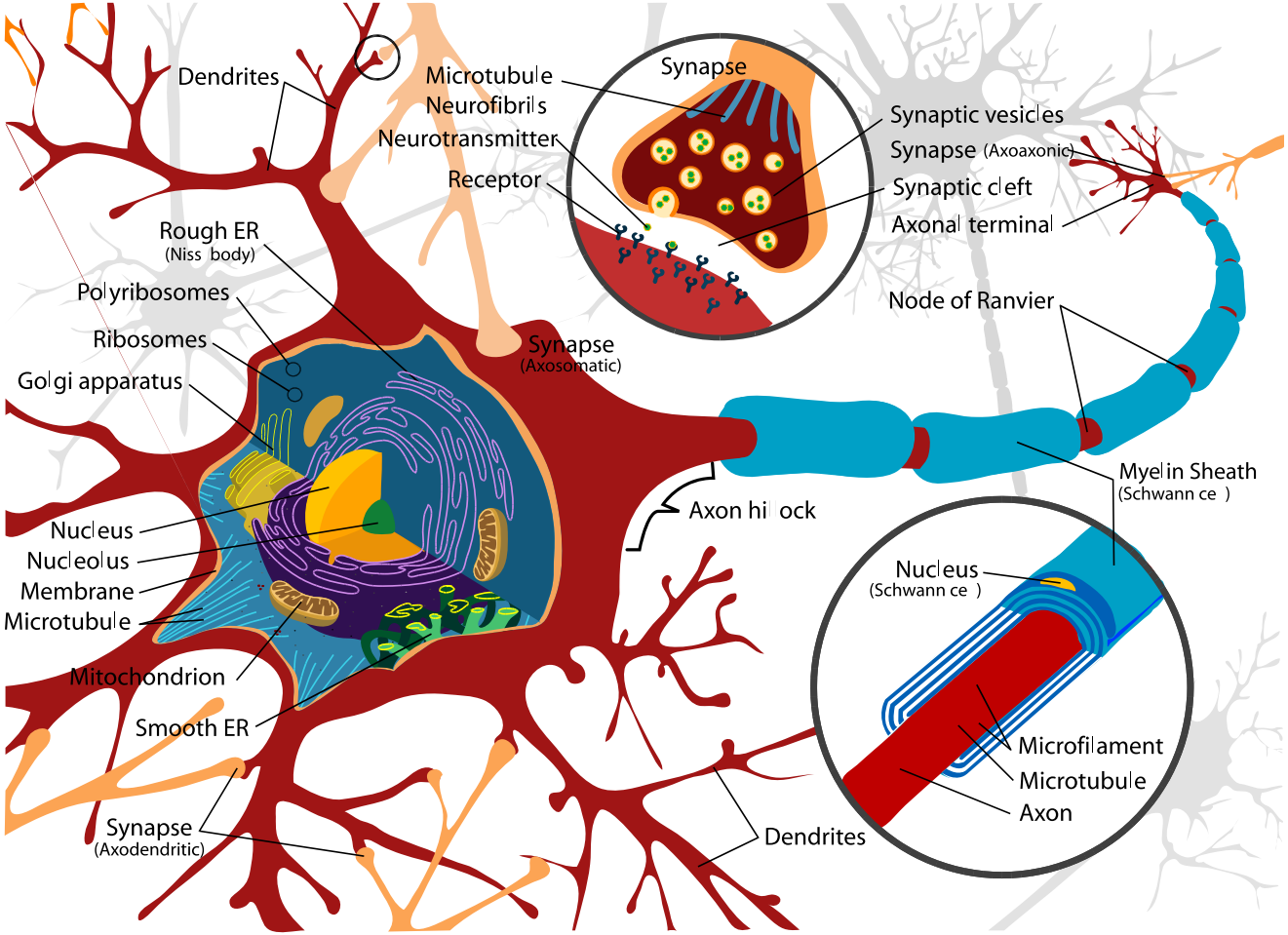
- Nevýhoda: pouze lineární rozhodovací hranice
- Možné řešení:
  - Použít jiný než lineární klasifikátor (např. GMM)
  - Nelineární transformace vstupních vektorů:

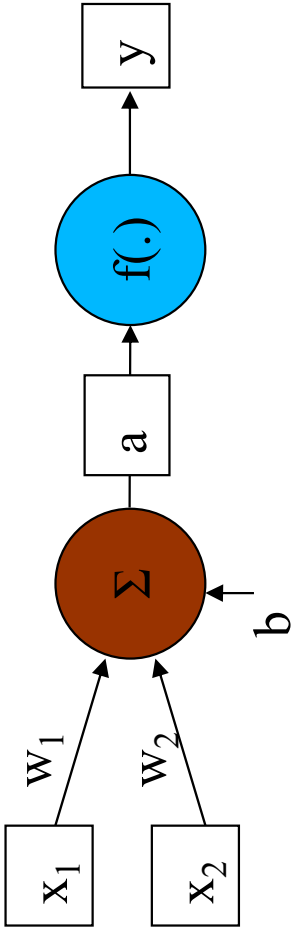


Ale jakou transformaci použít?

  - Postavit hierarchii lineárních klasifikátorů

Neuron a jeho matematický model

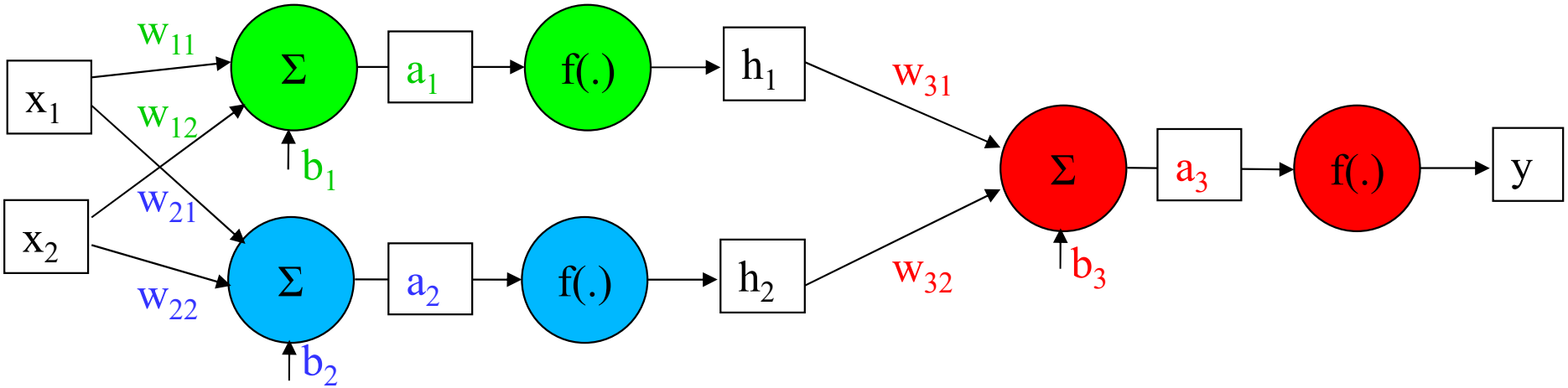




Zjednodušená abstrakce fyzického neuronu, inspirace v přírodě

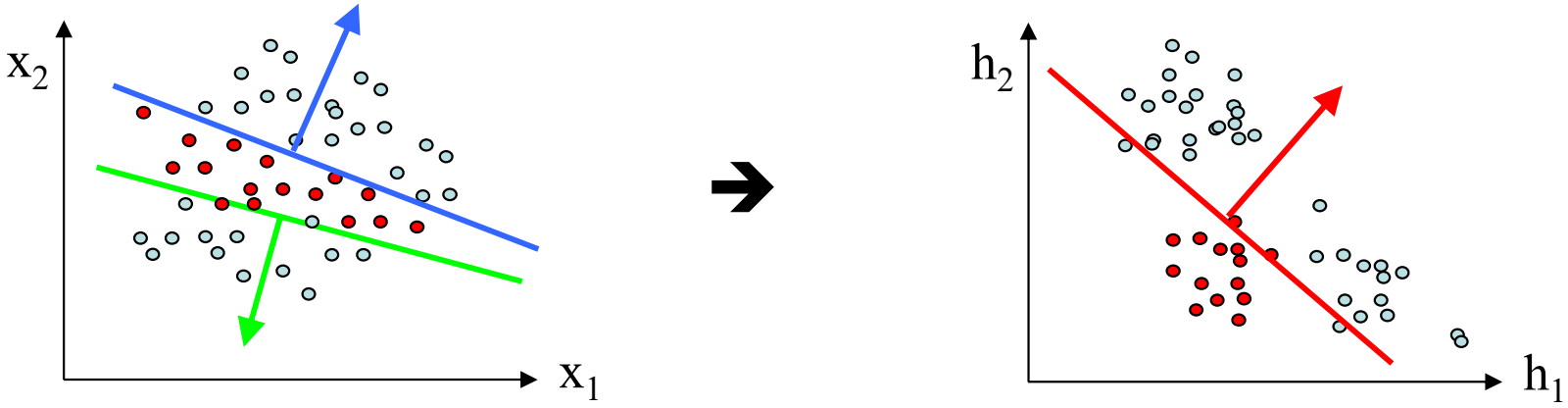
Zdroj ilustrace: wikipedia

Hierarchie lineárních klasifikátorů

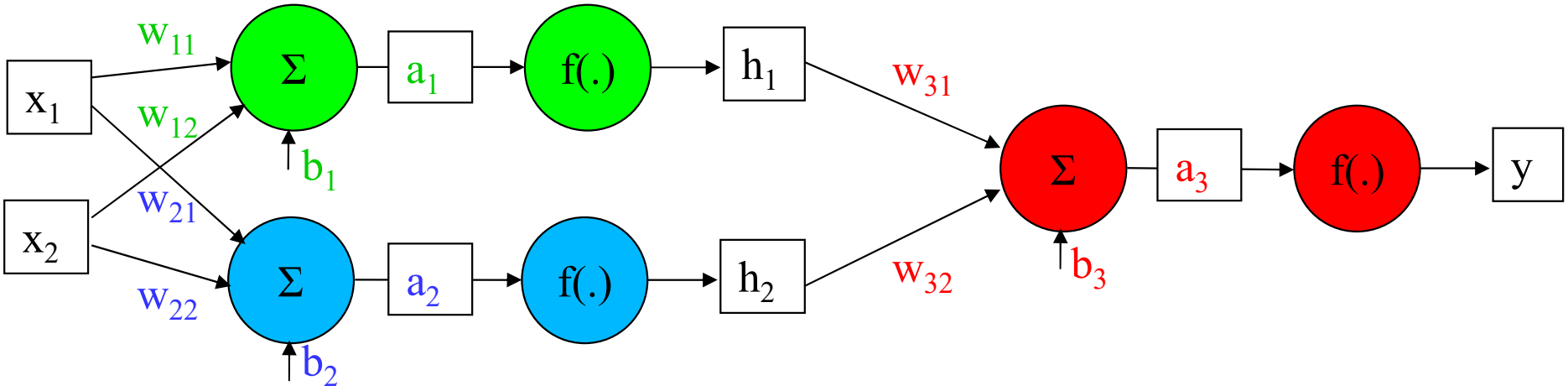


- Napřed natrénujeme modrý a zelený klasifikátor tak aby oddělily každý cluster modrých dat od zbytku.
  - Potřebujeme supervizi: Které modré data patří do kterého clusteru?

- Pak natrénujeme červený klasifikátor na pravděpodobnostních výstupech modrého a zeleného klasifikátoru



Neuronové sítě pro klasifikaci



- Takovýto hierarchický klasifikátor můžeme trénovat jako celek bez nutnosti předchozí supervize.
- Jedná se o jednoduchou neuronovou síť (Neural Network) pro binární klasifikační problém
- „Klasifikátory“ v první vrstvě
  - se mají samy naučit jaké clustery je v datech třeba identifikovat, aby finální lineární klasifikátor mohl oddělit třídy.
  - lze vidět jako nelineární transformaci do prostoru, kde jsou třídy dobře lineárně separovatelné.
  - představují tzv. *skrytou vrstvu*

Trénování neuronové sítě

- Uvažujme jednoduchý případ binárního klasifikátoru. Stejně jako u logistické regrese použijeme jako objektivní funkci pravděpodobnost správných anotací

$$P(\mathbf{t}|\mathbf{X}) = \prod_n P(t_n|\mathbf{x}_n)$$

kde $P(t_n|\mathbf{x}_n)$ je pravděpodobnost správné třídy t_n predikovaná neuronovou sítí pro vzor $\mathbf{x}_n$. $\mathbf{t}$ je vektor korektních identit tříd: $t_n = 1$ resp. $t_n = 0$, pokud $\mathbf{x}_n$ paří do třídy C_1 resp. C_2. Výstup neuronové sítě $y_n = P(C_1|\mathbf{x}_n)$ a a tedy $P(C_2|\mathbf{x}_n) = 1 - y_n$

- Opět se nám bude lepe pracovat se (záporným) logaritmem této objektivní funkce, tedy *vzájemnou entropii*:

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{X}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

- Opět budeme parametry (váhy neuronové sítě) optimalizovat pomoci metody *gradient descend:*

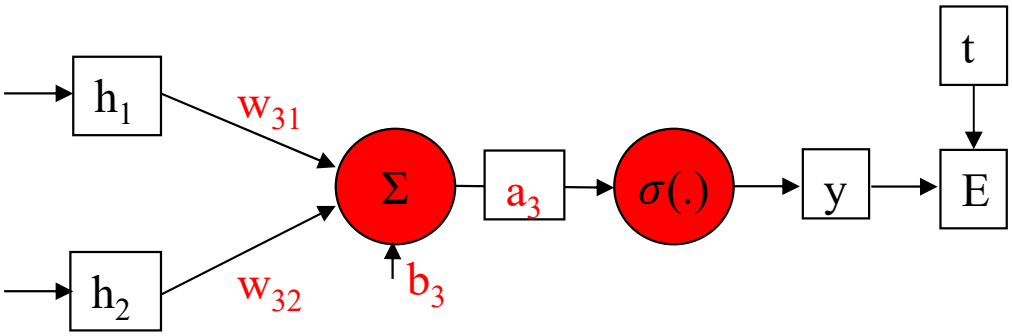
$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla E(\mathbf{w})$$

Dosazením pak dostaneme

Zpětné šíření chyby

Potřebujeme vypočítat gradient chyby:
(zde jsou vzorce jen pro jeden vzor trénovacích dat)

$$\nabla E = \left[\frac{\partial E}{\partial w_1}\frac{\partial E}{\partial w_2}\cdots\frac{\partial E}{\partial w_K}\right]^T$$



Nejprve určíme gradient vah mezi skrytou a výstupní vrstvou:

$$\frac{\partial E}{\partial w_{3,j}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a_3}\frac{\partial a_3}{\partial w_{3,j}}$$

Řetězové pravidlo: chyba E je složená funkce závislá na váze $w_{m,j}$ přes aktivaci a_3 a výstup y

$$\frac{\partial E}{\partial a_3} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a_3} = -\frac{\partial(t\ln y + (1-t)\ln(1-y))}{\partial y}\frac{\partial\,\mathrm{f}(a_3)}{\partial a_3} = \left(\frac{1-t}{1-y} - \frac{t}{y}\right)y(1-y) = y - t = \delta_3$$

$$\frac{\partial E}{\partial w_{3,j}} = \frac{\partial E}{\partial a_3}\frac{\partial a_3}{\partial w_{3,j}} = \delta_3\frac{\partial}{\partial w_{3,j}}\sum_{k=1}^{J} w_{3,k}h_k = \delta_3 h_j$$

Takzvaná *chyba* na výstupu

Zde předpokládáme, že všechny aktivační funkce jsou logistické sigmoidy $\mathrm{f}(a) = \sigma(a)$ a víme, že derivace $\sigma'(a)' = \sigma(a)\big(1-\sigma(a)\big) = y(1-y)$.

Zpětné šíření chyby II

Nyní určíme gradient vah mezi vstupní a skrytou vrstvou:

$$\frac{\partial E}{\partial w_{k,n}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a_3}\frac{\partial a_3}{\partial h_k}\frac{\partial h_k}{\partial a_k}\frac{\partial a_k}{\partial w_{k,n}}$$

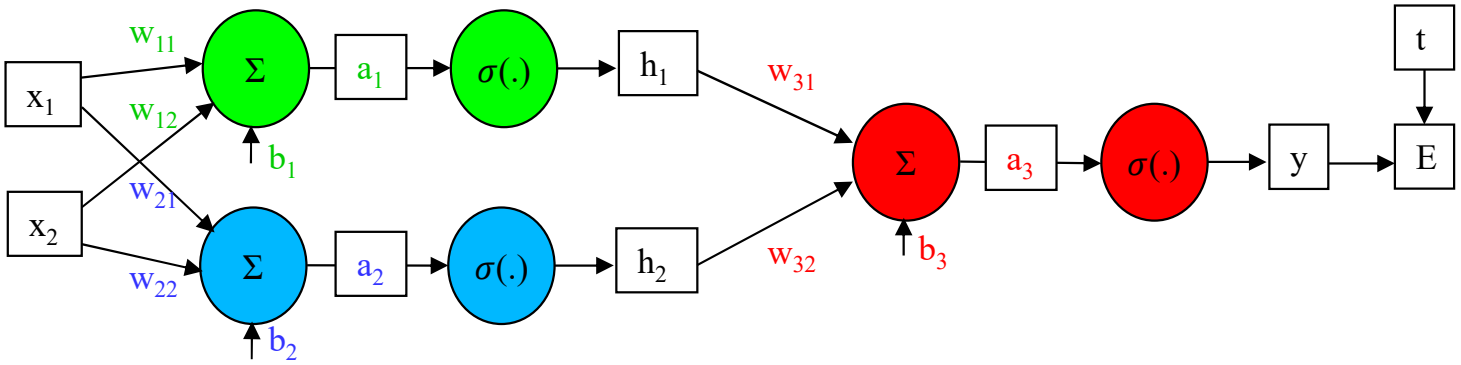
Opět řetězové pravidlo: chyba E je složená funkce závislá na váze $w_{j,n}$ přes aktivaci a_j a výstup h_j první vrstvy a dále přes aktivaci a_3 a výstup druhé vrstvy

Podobným výpočtem jako pro výstupní vrstvu a využitím už spočítaného δ_3:

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial a_3}\frac{\partial a_3}{\partial h_k}\frac{\partial h_k}{\partial a_k} = \delta_3 w_{3,k} h_k(1-h_k) = \delta_k$$

Chyba na výstupu δ_3 se *zpětně se propaguje* do „*chyby*“ ve skryte vrsve δ_k

$$\frac{\partial E}{\partial w_{k,n}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial w_{k,n}} = \delta_k x_n$$



Varianty neuronových síťí

- Umělé neuronové sítě mohou mít více než jednu skrytou vrstvu:
  - Současný trend pro large-scale problémy (rozpoznávání řeči či obrazu) jsou hluboké sítě (Deep Neural Networks) s několika (typicky do 10) vrstvami a až desítky tisíc neuronů (miliony trénovatelných vah)

- Neuronovou síť lze použít pro jiné problémy než binární klasifikace:

  Regrese
  - Neunorová síť muže aproximovat libovolnou (M-dimensionální) nelineární funkci
  - Typicky je použita lineární (či-li žádná) aktivační funkce na vytupu NN.
  - Objektivní funkce je typicky *Mean Squared Error* (pro N vzoru a M výstupů):

$$E = -\sum_{n=1}^{N}\sum_{m=1}^{M}(t_{nm} - y_{nm})^2$$

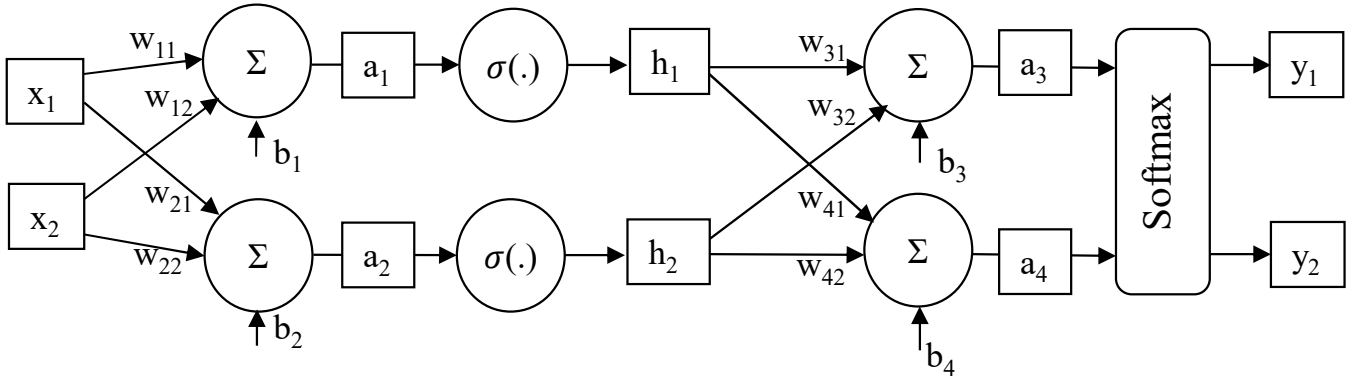
  Klasifikace do více tříd:
  - *Muticlass cross-entropy obektivní* funkce:

$$E = -\sum_{n=1}^{N}\sum_{m=1}^{M} t_{nm}\log y_{nm}$$

    kde t_{nm}, je 1 pokud n-tý vzor patří do m-té třídy, jinak je 0.
  - Softmax nelinearita na výstupu zaručí, že výstupem jsou normalizované posteriorní pravděpodobnosti říd:

$$y_m = \exp(a_m) / \sum_{i=1}^{M}\exp(a_i)$$

Dopředna klasifikační neuronová síť pro více tříd



$$\mathbf{h} = \sigma(\mathbf{W}_{ih}\mathbf{x} + \mathbf{b}_h)$$
$$\mathbf{y} = \mathrm{softmax}(\mathbf{W}_{ho}\mathbf{h} + \mathbf{b}_o)$$

$\mathbf{W}_{ih} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$ - matice vah ze vstupu do skryté vrstvy

$\mathbf{b}_h = [b_1 \quad b_2]^T$ - vektor biasů ve skryté vrstvě

$\mathbf{W}_{ho} = \begin{bmatrix} w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix}$ - matice vah ze skryté vrstvy do výstupní vrstvy.

$\mathbf{b}_o = [b_3 \quad b_4]^T$ - vektor biasů ve výstupní vrstvě

$\sigma(\mathbf{a})$ - logistická sigmoida aplikovaná na vektor $\mathbf{a}$ po elementech

$\mathrm{softmax}(\mathbf{a}) = \frac{\exp(\mathbf{a})}{\sum_m \exp(a_m))}$ - kde a_m jsou elementy vektoru $\mathbf{a}$

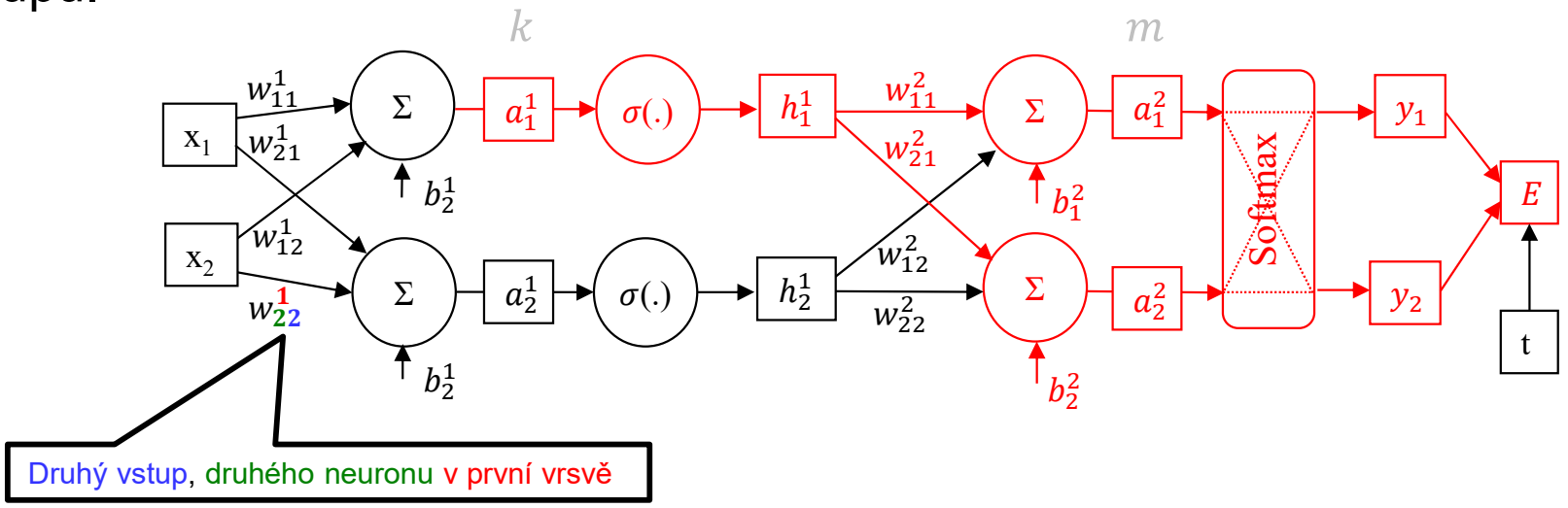
Zpětné šíření chyby III

V případě, že má neuronová sít více výstupů y_m (např. pro více tříd) nebo více vrstev, bude mít chyba: δ složitější tvar:

$$\frac{\partial E}{\partial a_k^{l-1}} = \left(\sum_{m=1}^{M} \frac{\partial E}{\partial a_m^l} \frac{\partial a_m^l}{\partial h_k^{l-1}} \right) \frac{\partial h_k^{l-1}}{\partial a_k^{l-1}} = \left(\sum_{m=1}^{M} \delta_m^l w_{mk}^l \right) h_k(1 - h_k) = \delta_k^{l-1}$$

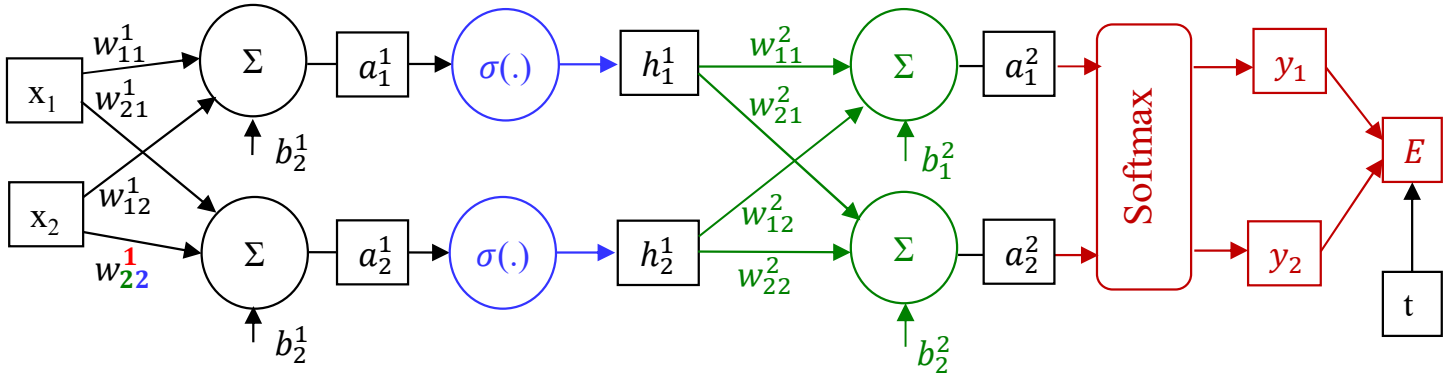
Index vrstvy

Protože všechny výstupy sítě závisí na hodnotě skryté vrstvy $\mathrm{h_k}$, sčítáme parciální derivace ze všech neuronů další vrstvy. Výraz $\delta_m^2 = y_m - t_m$ je chyba na m-tém výstupu.



Zpětné šíření chyby IV

Jak už jsme viděli, vyhodnocení neuronové sitě můzeme výjádřit vektorový zápisem:



$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{x} + \mathbf{b}^1$$
$$\mathbf{h}^1 = \sigma(\mathbf{a}^1)$$
$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2$$
$$\mathbf{y} = \mathrm{softmax}(\mathbf{a}^2)$$

Jodobně můžeme výjádřit i zpětné šíření chyby

$$\frac{\partial E}{\partial a_k^{l-1}} = \left(\sum_{m=1}^{M} \frac{\partial E}{\partial a_m^l}\frac{\partial a_m^l}{\partial h_k^{l-1}}\right)\frac{\partial h_k^{l-1}}{\partial a_k^{l-1}} = \left(\sum_{m=1}^{M} \delta_m^l w_{mk}^l\right) h_k(1-h_k) = \delta_k^{l-1}$$

jako:

$$\frac{\partial E}{\partial \mathbf{a}^{l-1}} = \frac{\partial E}{\partial \mathbf{a}^l}\frac{\partial \mathbf{a}^l}{\partial \mathbf{h}^{l-1}}\frac{\partial \mathbf{h}^{l-1}}{\partial \mathbf{a}^{l-1}} = \boldsymbol{\delta}^l \mathbf{W}^l \,\mathrm{diag}\left(\mathbf{h}^{l-1} \circ \left(1 - \mathbf{h}^{l-1}\right)\right) = \delta_k^{l-1}$$

Kde Jacobiho matice:

$$\frac{\partial E}{\partial \mathbf{a}^2} = \frac{\partial E(\mathrm{softmax}(\mathbf{a}^2))}{\partial \mathbf{a}^2} = \mathbf{y} - \mathbf{t} \qquad \frac{\partial a^l}{\partial \mathbf{h}^{l-1}} = \frac{\partial \mathbf{W}^l\mathbf{h}^{l-1} + \mathbf{b}^l}{\partial \mathbf{h}^{l-1}} = \mathbf{W}^l$$

$$\frac{\partial \mathbf{h}^l}{\partial \mathbf{a}^l} = \frac{\partial \sigma(\mathbf{a}^l)}{\partial \mathbf{a}^l} = \mathrm{diag}\left(\mathbf{h}^l \circ \left(1 - \mathbf{h}^l\right)\right)$$

Úprava vah

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla E$$

Úpravu vah lze provádět:

1. po předložení všech vektorů trénovací sady (*batch training*)
   - gradienty pro jednotlivé vzory z předchozích slajdů se akumulují
   - pomalejší konvergence, nebezpečí uvíznutí v lokálním minimu
2. po každém vektoru (*Stochastic-Gradient Descent)*
   - rychlejší postup trénování při redundantních trénovacích datech
   - riziko přetrénování na posledních pár vektorů z trénovací sady
   - data je lepší předkládat v náhodném pořadí
3. po předložení několika vektorů (*mini-batch training*)

Ochrana proti přetrénování

- Rozdělení dat: trénovací, cross-validační, testovací sada
- Algoritmus New-Bob:
  1. proveď jednu iteraci trénovaní
  2. zjisti úspěšnost NN na CV sadě
     - pokud přírůstek je menší než 0.5%, sniž „teplotu“ *ϵ o 1/2*
     - pokud přírůstek opětovně menší než 0.5%, zastav trénování
  3. jdi na 1.

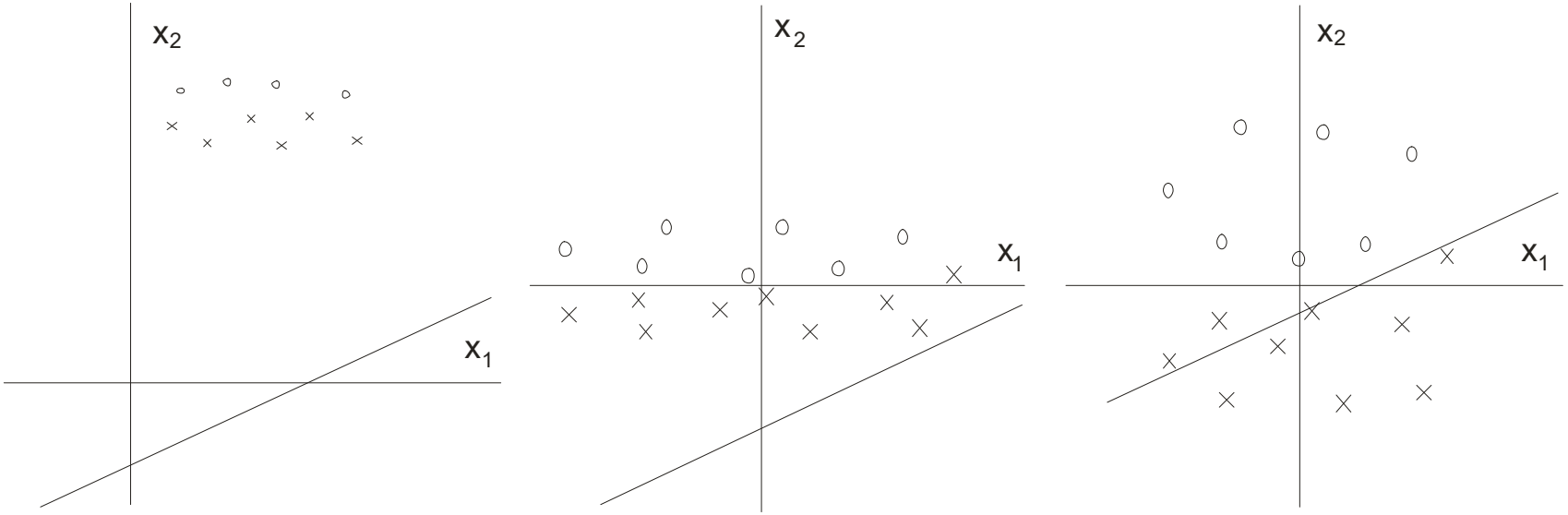
- Regularizace vah (weight-decay): do objektivní funkce přidáme výraz který penalizuje váhy s velkou kladnou či zápornou hondotou (viz regularizace logistické regrese)

Normalizace dat

bez normalizace

$$\tilde{x} = x - \mu$$

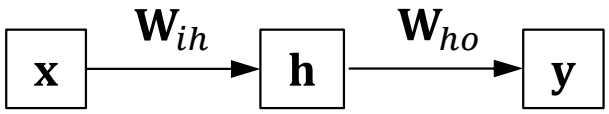
$$\tilde{x} = \frac{x - \mu}{\sigma}$$



- Transformujeme náhodné proměnné X, tak aby platilo:
  E[X] = 0; D[X] = 1
- Dynamický rozsah hodnot se přizpůsobí dynamickému rozsahu vah

Rekurentní neuronová síť

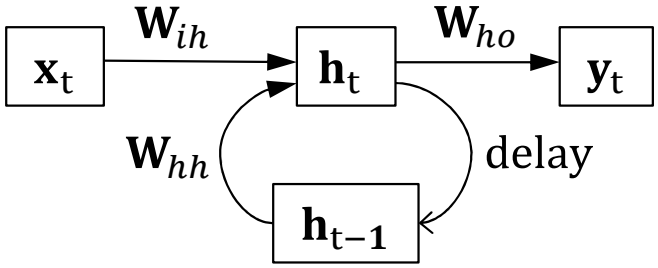
- Dopředná síť:



$$\mathbf{h} = \sigma(\mathbf{W}_{ih}\mathbf{x} + \mathbf{b}_h)$$

$$\mathbf{y} = \mathrm{softmax}(\mathbf{W}_{ho}\mathbf{h} + \mathbf{b}_o)$$

- Rekurentní neuronová sít:

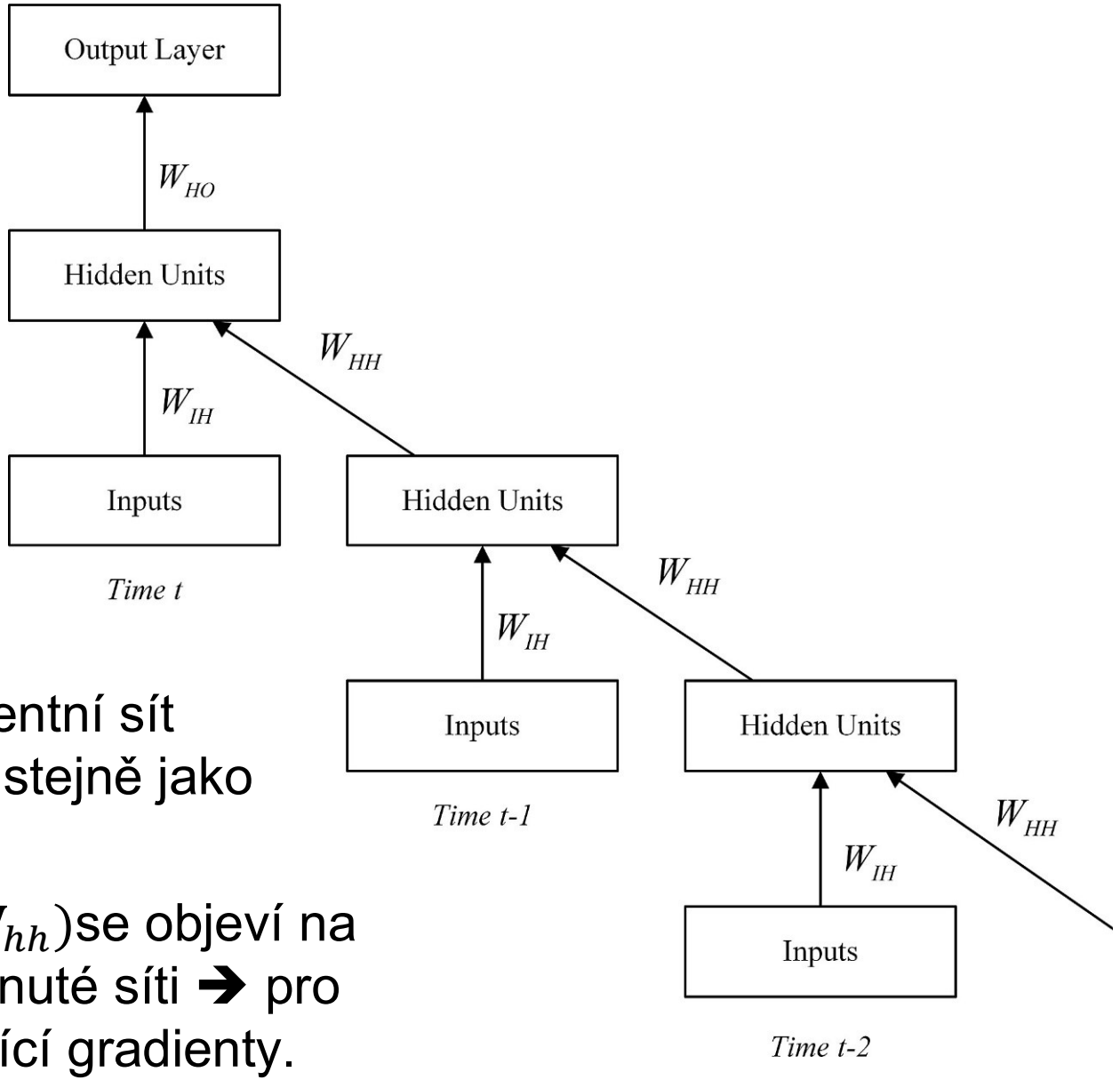


$$\mathbf{h}_t = \sigma(\mathbf{W}_{ih}\mathbf{x} + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

- Vhodná pro rozpoznávání sekvenčních dat (např. matice řečových příznaků)
- Ve „stavovém vektoru“ si pamatuje historii minulých vstupů
- Může nas zajímat $\mathbf{y}_t$ pro každé t ➔ mapování sekvence $[\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_T]$ na $[\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_T]$
- nebo jen $\mathbf{y}_T$ pro posledni vzor sekvence ➔ klasifikace celé sekvence

Zpětná propagace časem



- Pro trénování můžeme rekurentní sít rozvinout v čase a trénovat ji stejně jako doprednou neuronovou sít.
- Stejné „sdílené“ váhy ($\mathbf{W}_{ih}, \mathbf{W}_{hh}$)se objeví na různých mistech v takto rozvinuté síti ➔ pro trénování sečteme odpovídající gradienty.

Konvoluční neuronové sítě

- CNN je obvyklá architektura pro rozpoznávání obrázků
- Berou v úvahu to jak jednotlive vstupy neuronové sítě (pixely obrázku) leží vedle sebe.
- Sdílení vah pro různé pozice v obrázku ➔ invariance vůči translaci obrázku.

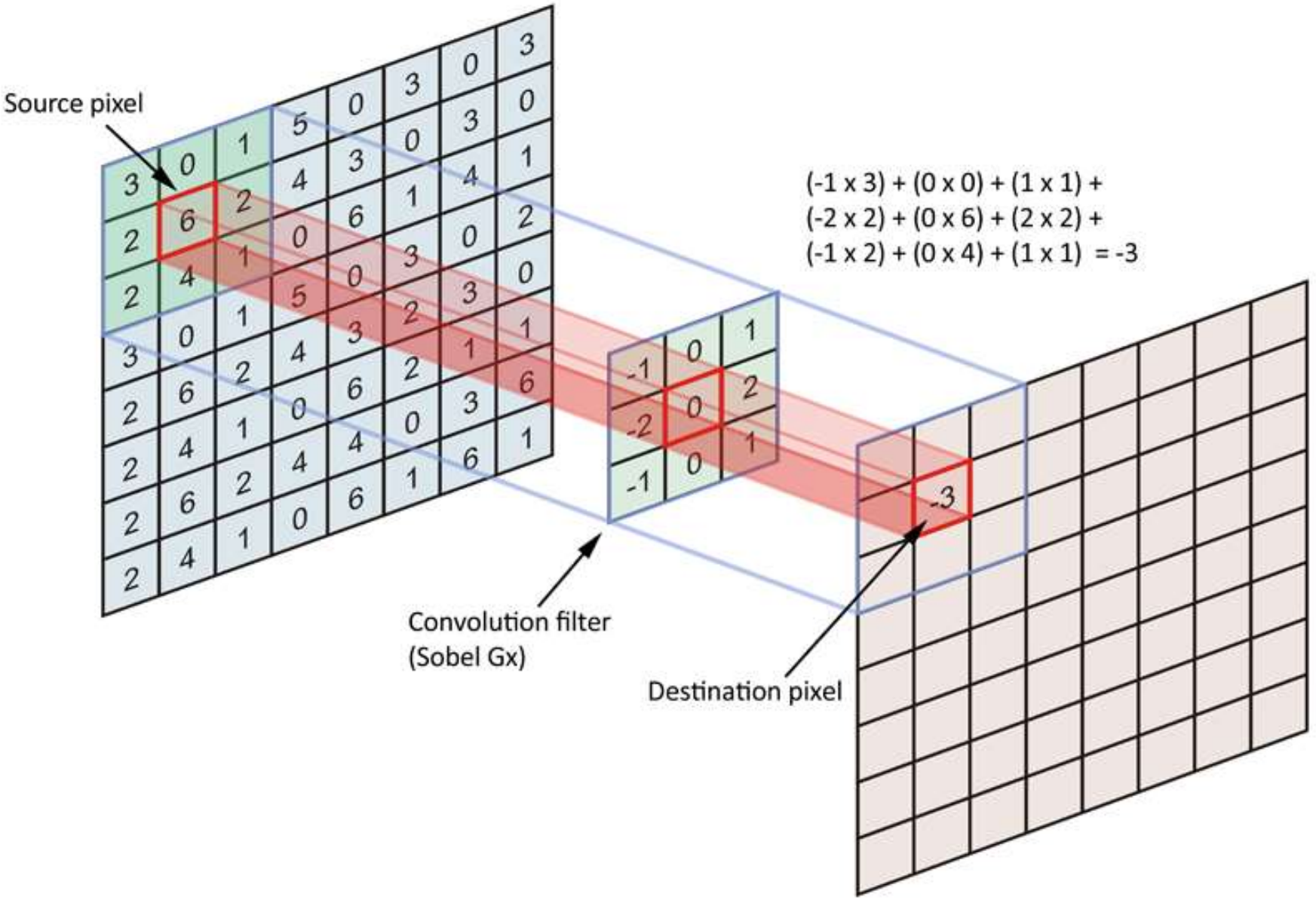
Diskrétní konvoluce

$$\mathbf{x}[i] * \mathbf{h}[i] = \sum_m \mathbf{x}[k]\mathbf{h}[i-k]$$

2d konvoluce

$$\mathbf{X}[i,j] * \mathbf{H}[i,j] = \sum_k \sum_l \mathbf{X}[k,l]\mathbf{H}[i-k, j-l]$$

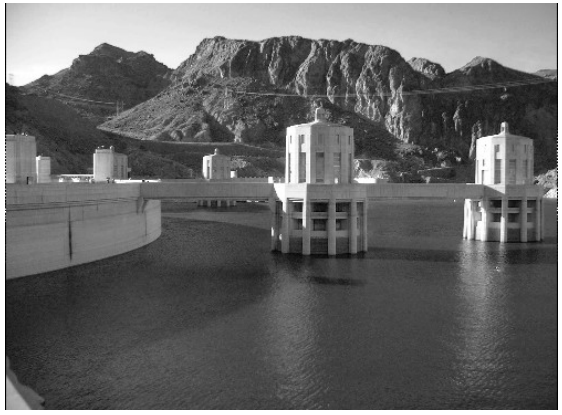
2d konvoluce



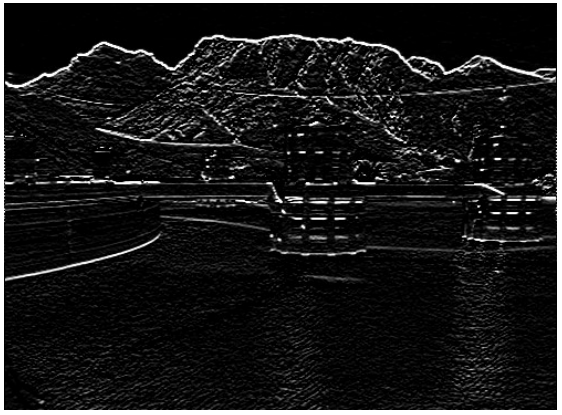
Konvoluční filtry

- Konvoluční vrstva aplikuje několik konvolučních filtru jejichž váhy se v rámci trénování CNN učíme.
- Výsledkem je několik hladin (replik) obrázku (tzv. feature maps)
- Na výsledné obrázky se jestě pixel po pixelu aplikuje nelinearita: např. sigmoida nebo *ReLU(a) = max(0,a)*

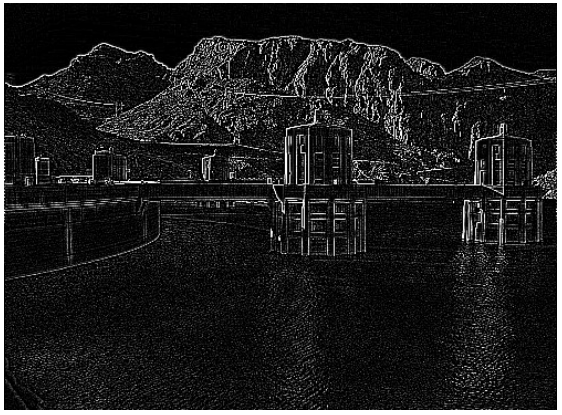
0	0	0
0	1	0
0	0	0



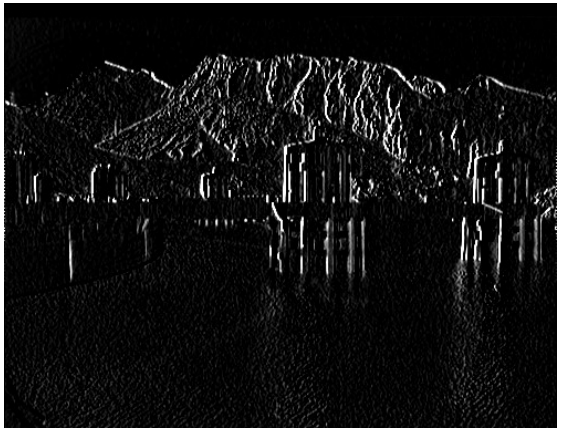
1	2	1
0	0	0
-1	-2	-1



-1	-1	-1
-1	8	-1
-1	-1	-1

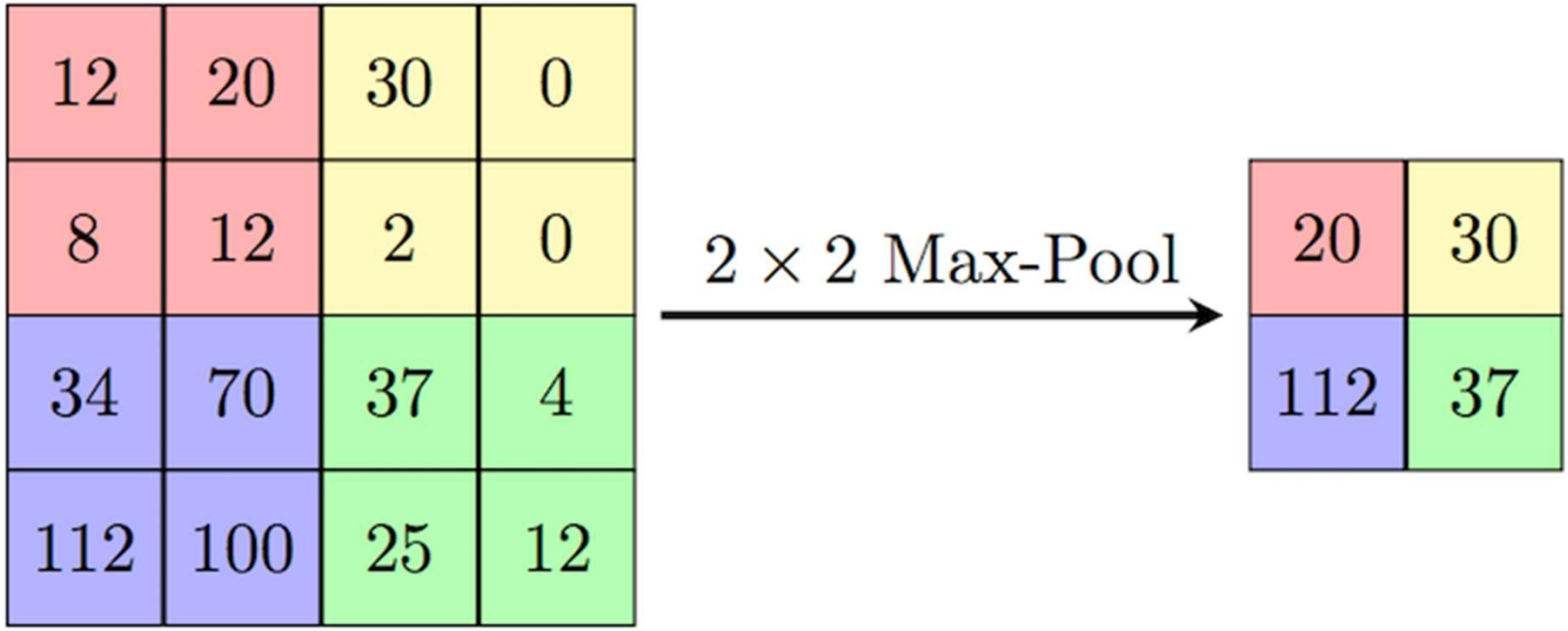


-1	0	1
-2	0	2
-1	0	1



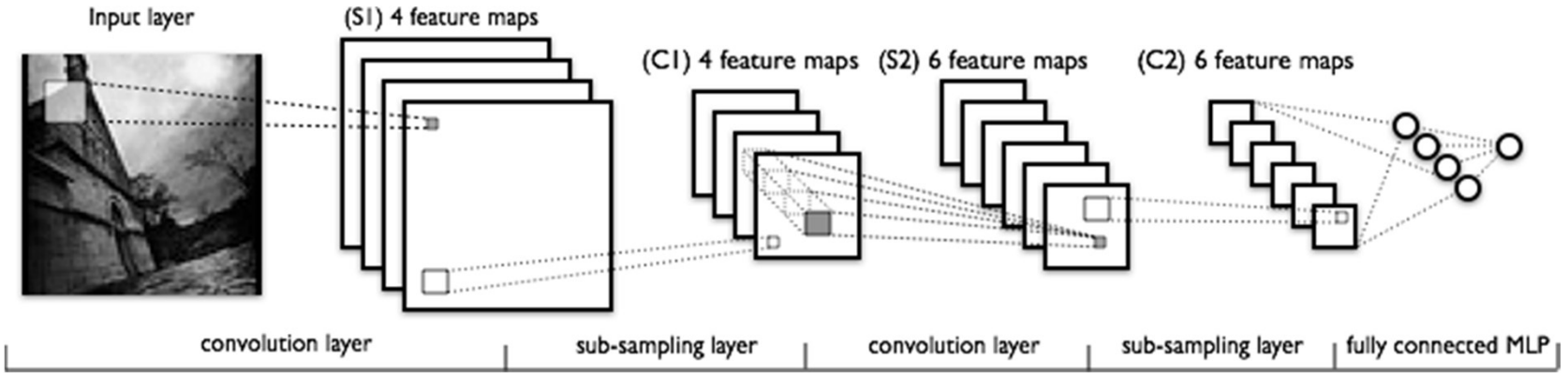
Pooling

- Pooling vrstva provede podvzorkování obrázku
- Např Max-pooling nahradí každou čtverici pizeků pouze jejich maximalní hodnotou ➔ zredukuje onrázek na poloviční velikost



Kompletní CNN

- CNN střídá konvolučni a pooling vrstvy
- Na konci se většinou objevi obvyklá plně propojená hladina



- Konvoluční filtry v pozdějších vrstvách mají tvar 3D matic tak aby operovaly nad všemi “feature maps”. Filtry se ale pohybuji jen ve dvou dimenzich:

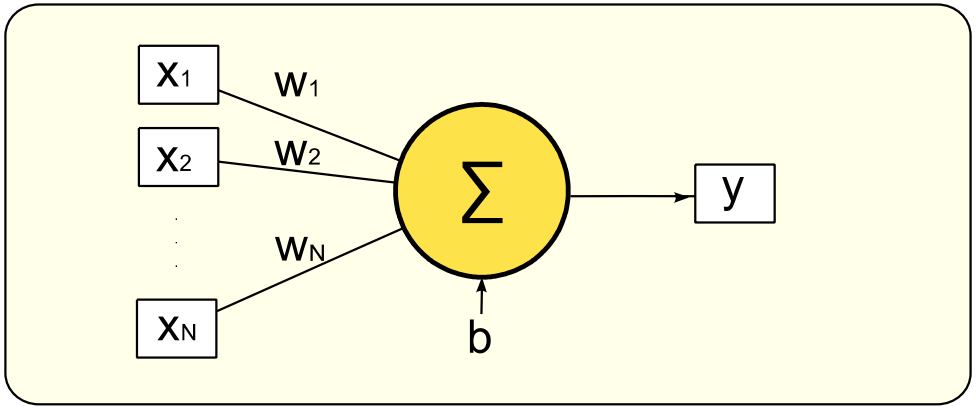
$$\mathrm{conv}(\mathbf{X}, \mathbf{H})_{i,j} = \sigma\left(\sum_k \sum_l \sum_m \mathbf{X}[k,l,m]\mathbf{H}[i-k, j-l, m]\right)$$

- Pro barevné obrázky operuje i prvni konvoliční vrstva nad třemi barevnými hladinami (RGB feature maps)

Support Vector Machines

Support Vector Machines

- SVM lineární klasifikátor s specifickou objektivní funkci: maximum margin

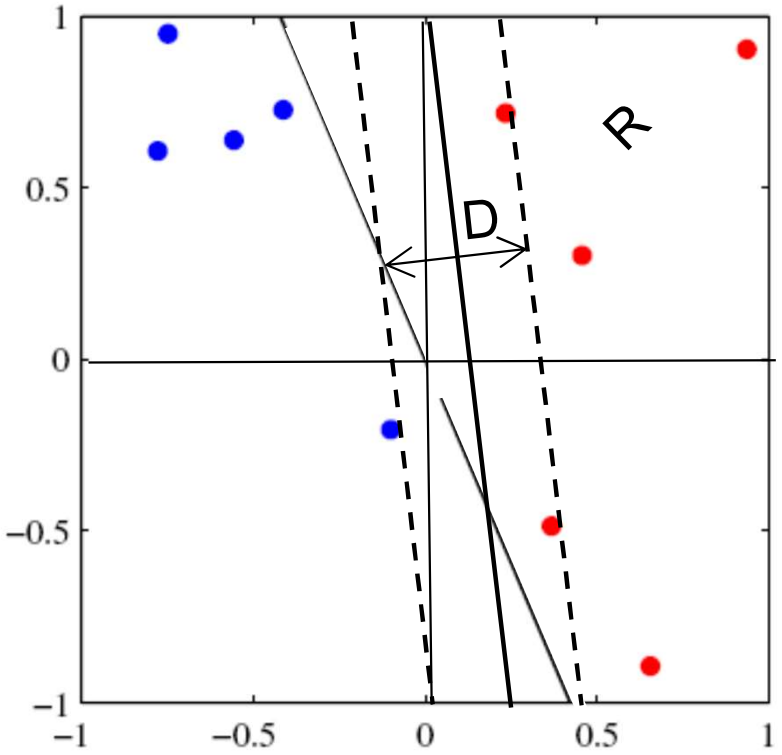


$$y = \sum_{n=1}^{N} w_n x_n + b = \mathbf{w}^T \mathbf{x} + b$$

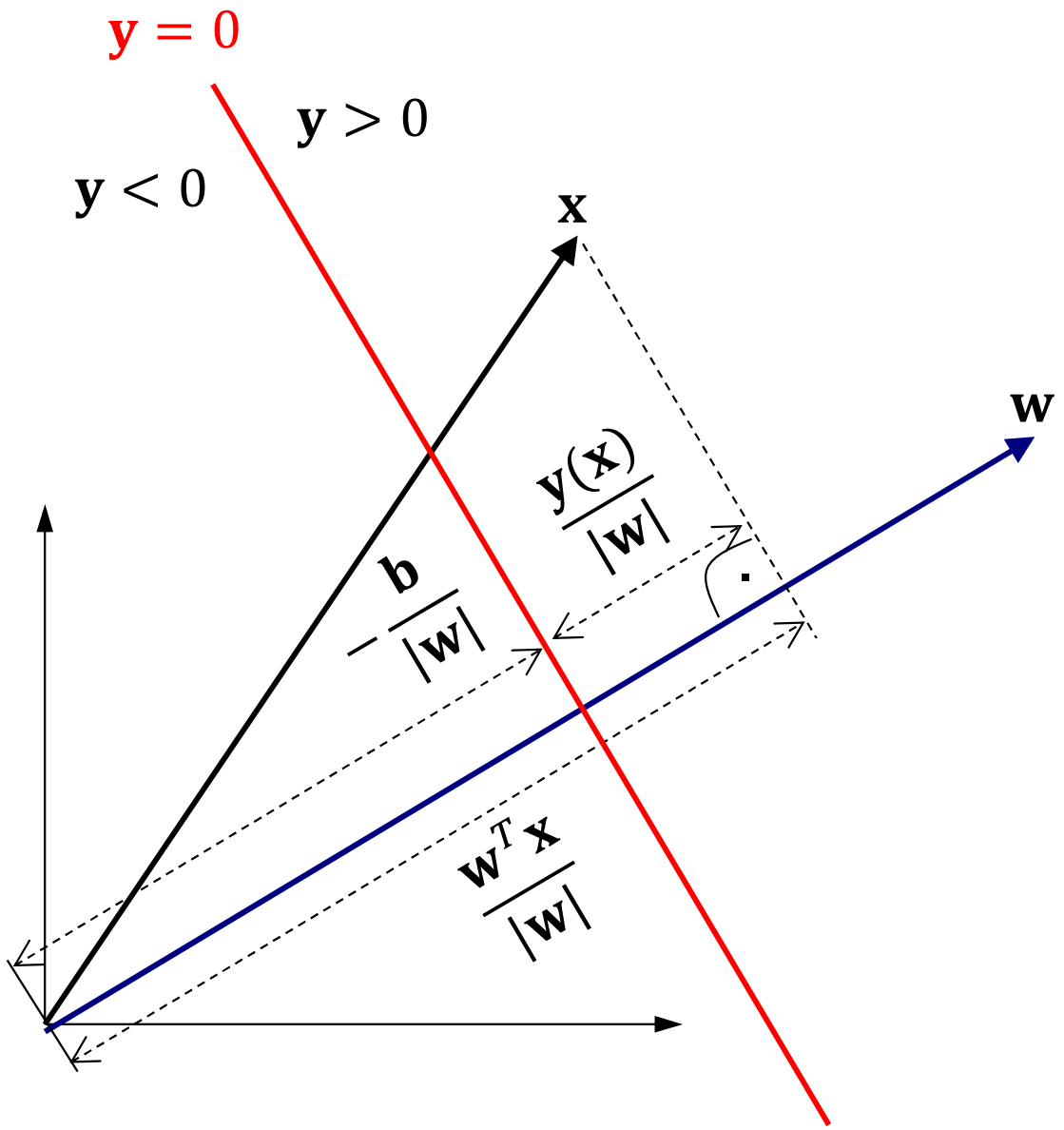
- V základní variantě zvládne pouze dvě nepřekrývající se lineárně separovatelné třídy

SVM kritérium

- Maximalizuje mezeru (margin) mezi dvěmi shluky dat



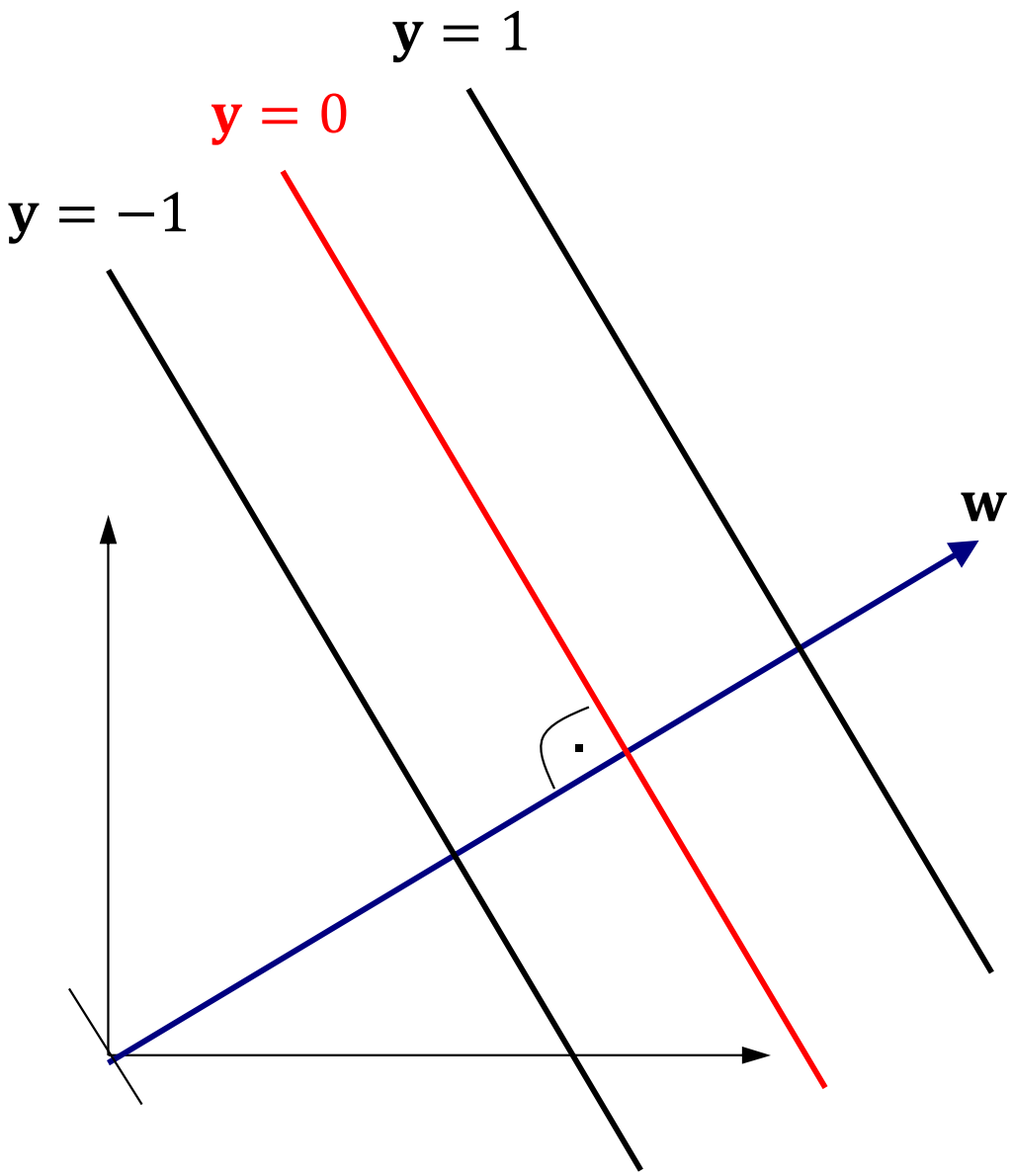
Lineární klasifikátor



$$\mathbf{y}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + \mathbf{b}$$

$$\frac{\mathbf{w}^T \mathbf{x}}{|\mathbf{w}|} = -\frac{\mathbf{b}}{|\mathbf{w}|} + \frac{\mathbf{y}(\mathbf{x})}{|\mathbf{w}|}$$

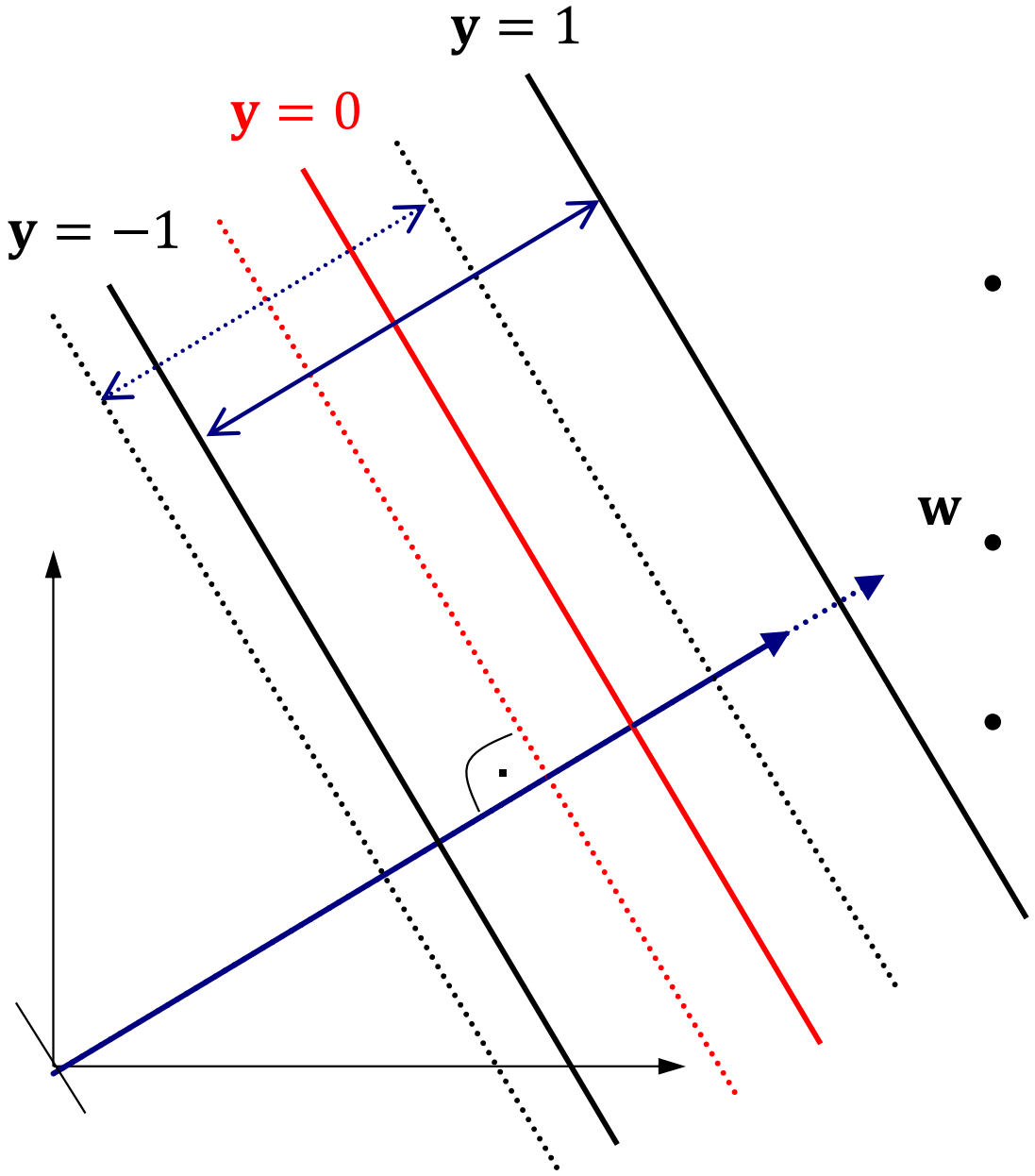
Lineární klasifikátor



$$y = \mathbf{w}^T \mathbf{x} + b$$

- Rozhodněme, že margin bude dán prostorem mezi přímkami y=-1 a y=1
- Co se stane, když zkrátíme vektor **w**?

Lineární klasifikátor



$$y = \mathbf{w}^T \mathbf{x} + b$$

- Rozhodněme, že „margin“ bude dán prostorem mezi přímkami y=-1 a y=1
- Co se stane, když zkrátíme vektor **w**? *Margin se zvětší!*
- Budeme hledat řešeni, kde **w** je co nejkratší a kde „margin“ odděluje data obou tříd.

Trénování SVM

- Minimalizujeme:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

- S podmínkami:

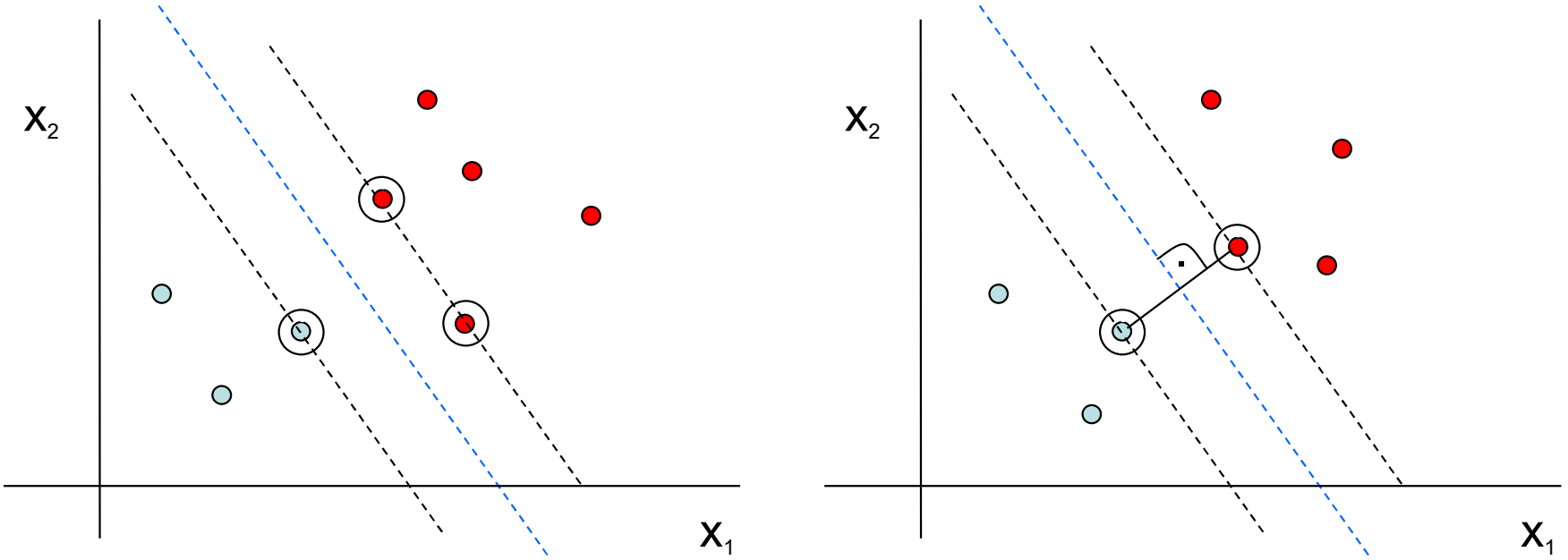
$$t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \qquad n \in \{1,2,\ldots,N\}$$

- *N* je počet trénovacích vektorů a $t_n \in \{-1,1\}$ udávají třídy pro jednotlivá trénovací data.

- Jedná se o problém tzv. kvadratického programování, což je speciální případ optimalizace s omezením (constrained optimization).

Co jsou to Support Vectors?

- Podpůrné vektory (support vectors) leží na okraji prázdné oblasti a přímo ovlivňují řešení
- Kdyby se ostatní data vypustila z trénovacího setu, výsledek by se nezměnil



Řešení

- Normálový vektor w definující rozhodovací hranici lze sestavit lineární kombinaci podpůrných vektoru:

$$\mathbf{w} = \sum_{i \in S} t_i \alpha_i \mathbf{x}_i \qquad \alpha_i > 0$$

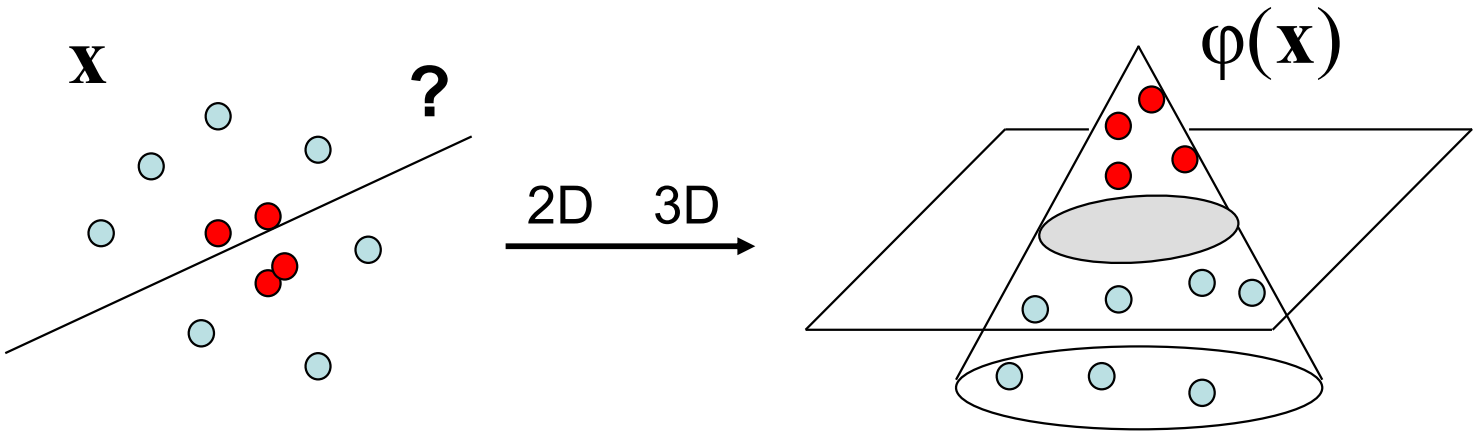
- Tato reprezentace umožňuje klasifikaci bez explicitního vyjádření vektoru **w**:

$$y = \mathbf{w}^T \mathbf{x} + b = \sum_{i \in S} t_i \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

- Podobně i vstupem do trénovacíhi mohou být skalární součiny mezi trénovacími daty ➔ možnost použití následujícího „**kernel triku**“.

Lineárně neseparovatelná úloha

- Může být řešena nelineárním mapováním dat $\varphi(\mathbf{x})$ do nového prostoru (s více dimenzemi)



- Pro SVM potřebujeme v novem prostoru spočítat skalární součin mezi jakýmikoli dvěma mapovanými vektory.
- To lze často udělat efektivněji bez explicitního mapovaní do noveho prostoru pomoci tzv. jádrové funkce (kernel function)

$$\mathrm{K}(\mathbf{x},\mathbf{y}) = \varphi(\mathbf{x})^T\ \varphi(\mathbf{y})$$

Příklady jader

- Polynomiální jádro: $\mathrm{K}(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^d$
  - Přiklad pro jednorozměrná data a *d*=2

$$\mathrm{K}(x,y) = (1+xy)^2 = 1+2xy+x^2y^2 = \varphi(x)^T\,\varphi(y)$$

  - Což odpovídá skalárnímu součinu po mapování:

$$\varphi(x)^T = \left(1, \sqrt{2}x, x^2\right)$$

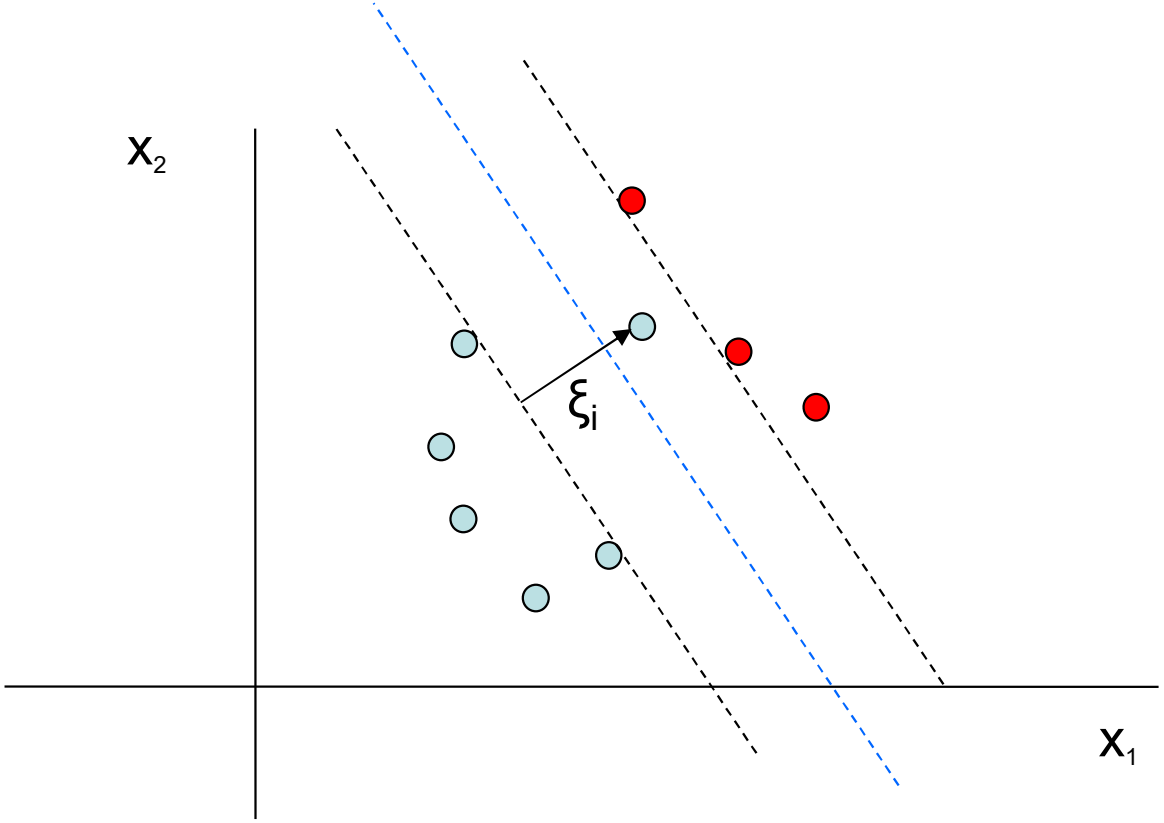
- Radiální bázove funkce:

$$\mathrm{K}(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})\right)$$

  - Odpovídají skalárnímu součinu po projekci do jistého nekonečně dimensionálního prostoru ➔ vždy separovatelné třídy .

Překrývající se třídy

- Pro překrývající třídy uvedené řešení selže.
- Zavádějí se proměnné (slack variables), které oslabí omezující podmínky



Překrývající se třídy II

- Minimalizujeme

$$\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \xi_n$$

- S podmínkami:

$$t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \qquad n = 1,\ldots,N$$

$$\xi_n \geq 0$$

- První výraz maximalizuje mezeru (margin) mezi třídami a druhý penalizuje vzory porušující tento margin.
- *C* řídí kompromis mezi oběma výrazy. C→∞ odpovídá originální variantě pro separovatelná data.

Vlastnosti a použití SVM

- Výstup SVM nemá pravděpodobnostní interpretaci.
  - nepredikuje pravděpodobnost tříd pro daný vstup
  - produkuje ale měkké skóre, které lze přibližně na „pravděpodobnost třídy“ převést.
  - Objektivní funkce má blíže k *maximalizaci počtu správně rozpoznaných* než k *maximalizaci pravděpodobnost že vše je rozpoznáno dobře* (objektivní funkce logistické regrese).

- Často používaný klasifikátor pro problémy se dvěma třídami.
  - Rozšíření na více tříd je možné, ale ne tak přímočaré jako u pravděpodobnostních klasifikátorů.

- Hlavní výhoda oproti např. logistické regresi je možnost implicitní nelineární transformace vstupů pomoci jader.
  - Společná vlastnost všech jádrových metod (Kernel methods).

Software

- Existuje velni dobrá knihovna LibSVM

  http://www.csie.ntu.edu.tw/~cjlin/libsvm/

- Funkce v Matlabu:
  - svmtrain, svmclassify