UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

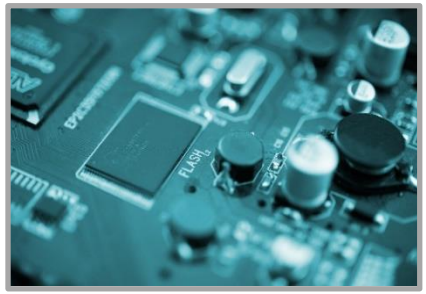
Umělá inteligence pro průmyslovou praxi

Kurz celoživotního vzdělávání

prof. Ing. Petr Doležel, Ph.D.

NÁRODNÍ PLÁN OBNOVY
Financováno Evropskou unií
NextGenerationEU

Umělá inteligence pro průmyslovou praxi



Umělá inteligence pro průmyslovou praxi

prof. Ing. Petr Doležel, Ph.D.

FEI, Univerzita Pardubice
Nám. Čs. legií 565
e-mail: petr.dolezel@upce.cz
tel.: 466 037 123

Úvod

Umělá inteligence

Expertní systémy (ES)

Fuzzy systémy (FS)

Evoluční algoritmy (EA)

Umělé neuronové sítě (NS)

Vysoká úroveň modelování

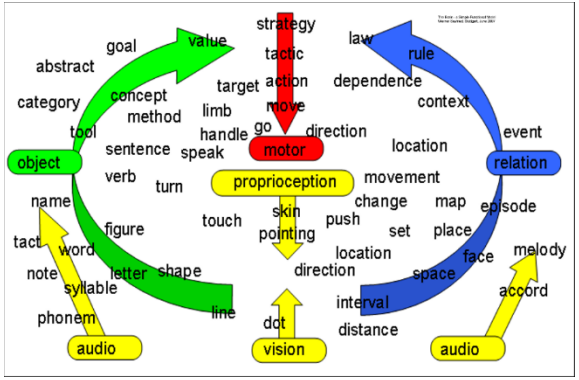
Nízká úroveň modelování

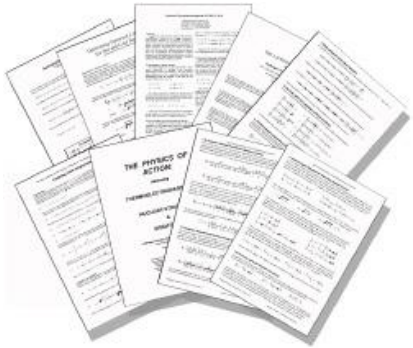
Mozek

Parametry mozku
Váha: 1200 - 1400 g
Objem: 140 x 160 x 90 mm
Plocha: 2200 - 2400 cm^2
Šedá kůra (cortex): tloušťka 1 - 4,5 mm

Hmotnosti mozku různých zvířat
velryba 7800 g
slon 6000 g
delfín 1500 g
gorila 540 g
kráva 460 g
ovce 140 g
pes 72 g
kočka 30 g
potkan 2 g
ještěrka 0,08 g

Mozek - funkce

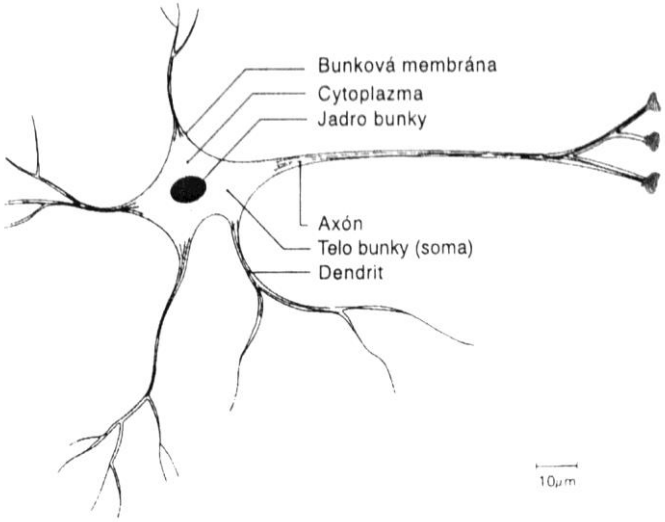




Mozek – některá historická fakta

335 let př.n.l.	Aristoteles	paměť, sny a pod.
400 let př.n.l.	Platon	v mozku věčná duše
130 n.l.	R.C. Galenus	nervy jako trubičky
1660	I. Newton	šíření vibrací
1771	L. Galvani	elektrický potenciál
19. století	*intensivní výzkum*	
1835	J. E. Purkyně	identifikace neuronu neuronová síť
1864	P. Brocca	lokalizace centra řeči

Biologický neuron



Soma tělo neuronu
rozměr µm až 10 µm

Dentrit vstup do neuronu
délka 3 mm

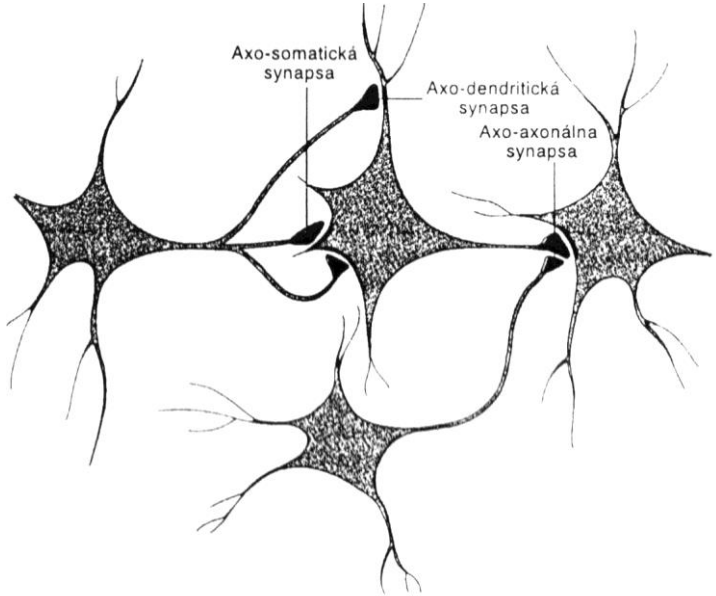
Axon výstup z neuronu
délka až v metrech

Synapse kontakt mezi neurony

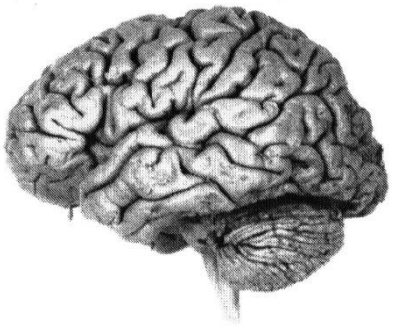
V mozku (v cortexu) je odhadem 10 miliard neuronů uložených zhruba v 6 vrstvách. Hustota je $(7 – 8).10^4$ neuronů na 1 mm^3. Každý neuron může mít s jiným neuronem 10 až 100 tisíc propojení. Celkový počet propojení je cca 60000 miliard.

Biologická neuronová síť

Propojení jednotlivých neuronů, synapse určují propustnost (váhu) daného spojení



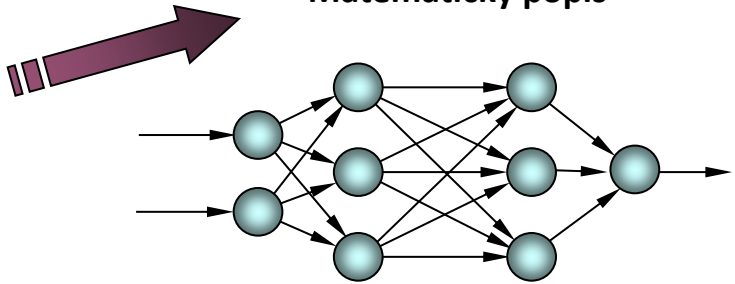
Umělá neuronová síť



MOZEK
Hardware lidského bytí
Základní procesor biologického systému

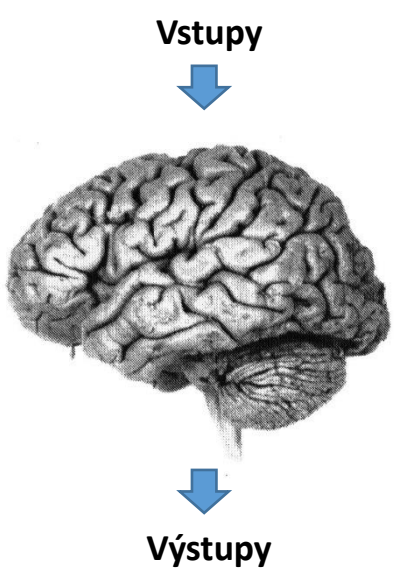
Biologická neuronová síť

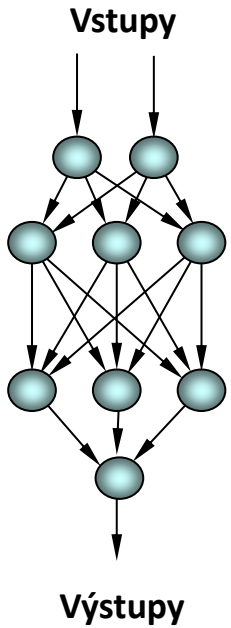
Matematický popis



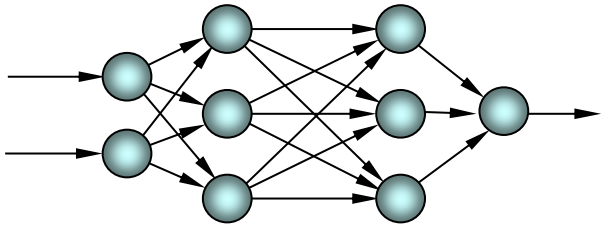
Umělá neuronová síť

Umělá neuronová síť

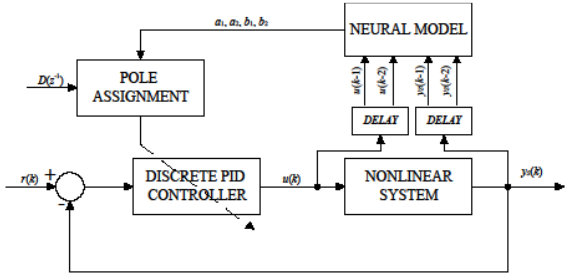


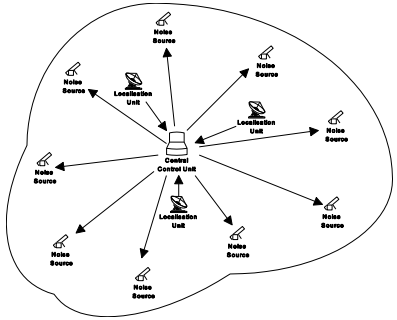


Použití umělé neuronové sítě

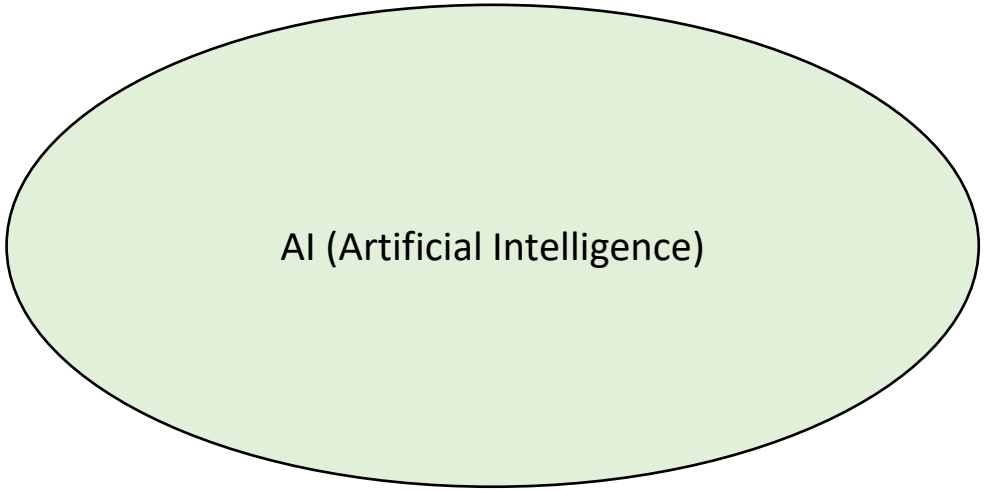


- Predikce
- Univerzální aproximace
- Rozpoznávání vzorů
- Rozhodování
- Zpracování signálů



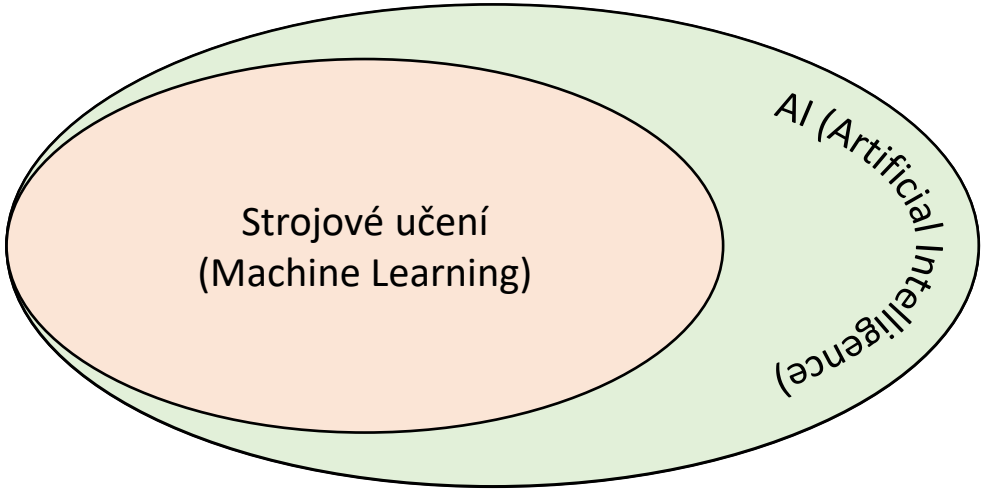


Co je AI?



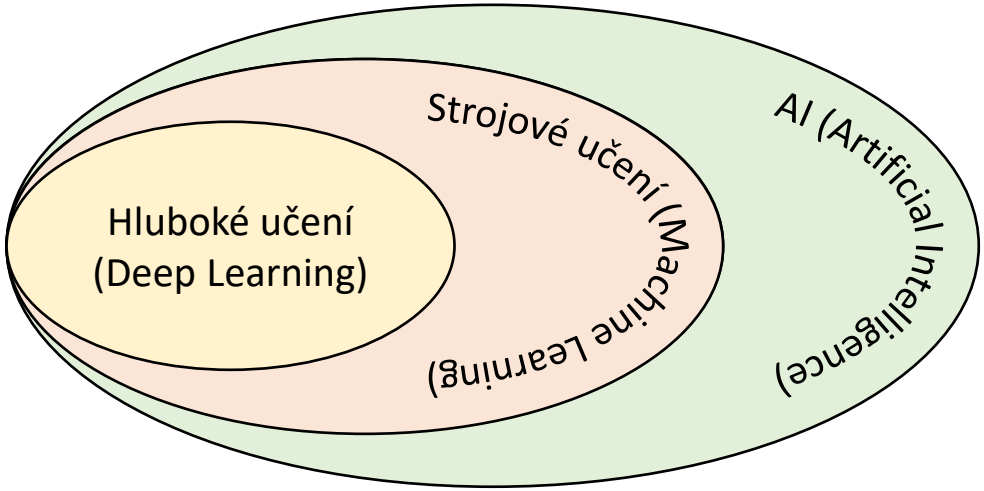
Schopnost strojů napodobovat kognitivní funkce, jako je učení a rozhodování.

Co je AI?



Podskupina AI zaměřená na trénování algoritmů na datech.

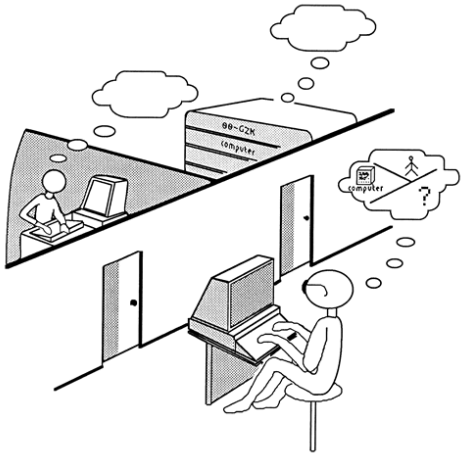
Co je AI?



Pokročilá forma strojového učení, využívající neuronové sítě.

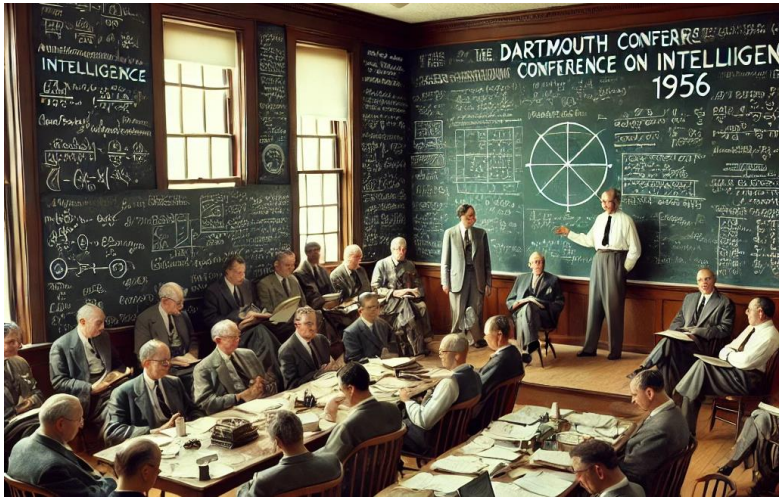
Vývoj, pokroky a „doby ledové“

- 1950s – Počátky AI
  - Alan Turing představuje svůj test jako způsob měření inteligence strojů.
  - První pokusy o programy simulující šachovou hru nebo logiku.



Vývoj, pokroky a „doby ledové“

- 1956 – Dartmouthská konference
  - AI se rodí jako obor: První zmínka o pojmu „Artificial Intelligence“.
  - Optimismus: Představa, že stroje budou brzy myslet jako lidé.



Vývoj, pokroky a „doby ledové“

- 1970s – První AI ZIMA (AI Winter)
  - Nedostatek výpočetního výkonu a neúspěchy v aplikacích.
  - Pokles financování, protože AI nesplnila přehnaná očekávání.

Vývoj, pokroky a „doby ledové“

- 1980s – Renesance AI díky expertním systémům
  - Vývoj expertních systémů (např. MYCIN v medicíně).
  - Komerční úspěchy vedly ke zvýšení financování.
  - Neuronové sítě se objevují jako potenciální řešení.

Vývoj, pokroky a „doby ledové“

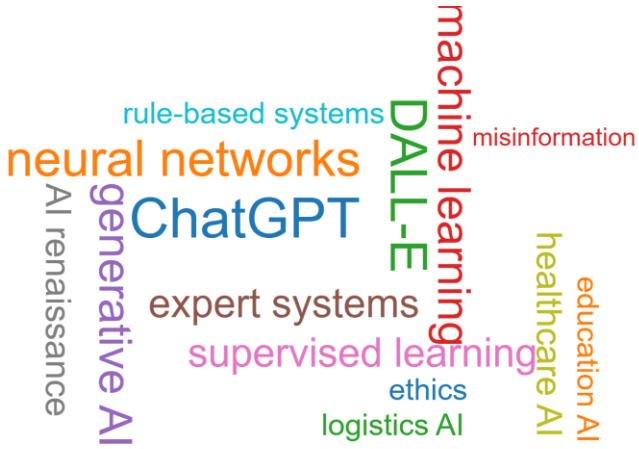
- 1990s – Druhá AI ZIMA (AI Winter 2)
  - Důvod: Expertní systémy byly drahé a rigidní.
  - Nadšení z AI znovu upadá.

Vývoj, pokroky a „doby ledové“

- 2000s – Návrat díky velkým datům
  - Hluboké učení (Deep Learning) začíná přinášet skutečné výsledky.
  - Využití masivního výpočetního výkonu (GPU).
  - První velké úspěchy ve strojovém vidění a rozpoznávání hlasu.
- 2010s – Zlatý věk AI
  - AI poráží lidské šampiony v hrách (šachy, Go – AlphaGo od DeepMind).
  - Rozmach personalizovaných služeb a aplikací (Google Assistant, Netflix).
  - Hluboké neuronové sítě dominují v přirozeném jazyce (GPT, BERT).

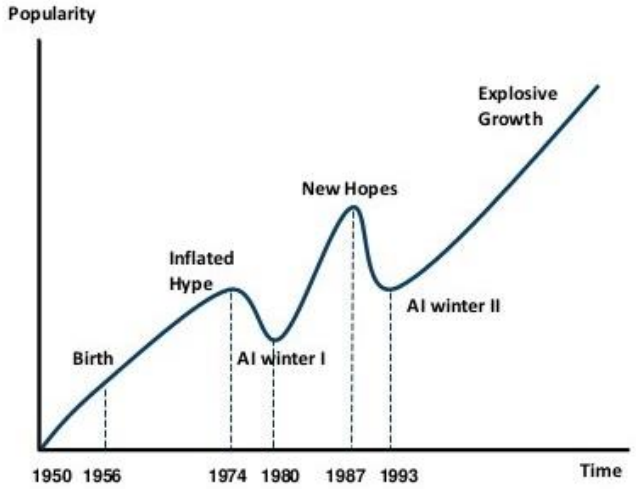
Vývoj, pokroky a „doby ledové“

- 2020s – Současnost a kam směřujeme
  - Vývoj generativních modelů (ChatGPT, DALL-E, MidJourney).
  - AI řeší problémy ve zdravotnictví, logistice, vzdělávání, ale přináší i výzvy (etika, dezinformace).



Důvody pro „doby ledové“ a pokroky AI

- Technologické limity: Nedostatek výpočetního výkonu a dat.
- Přehnaná očekávání: Nedosažené sliby vyvolaly pokles financování.



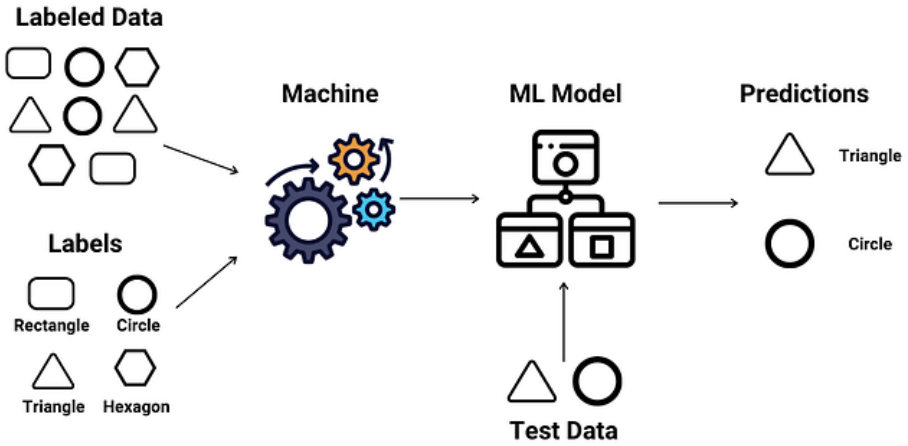
Timeline of AI Development

- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

Data?

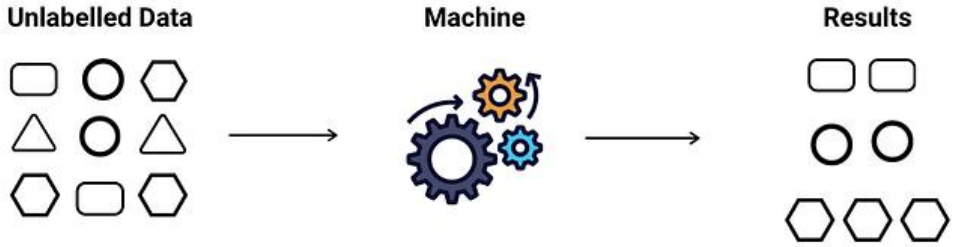
Typy učení v AI

- Supervised Learning (Učení s učitelem)
- Jak to funguje?
  - Model se trénuje na označených datech.



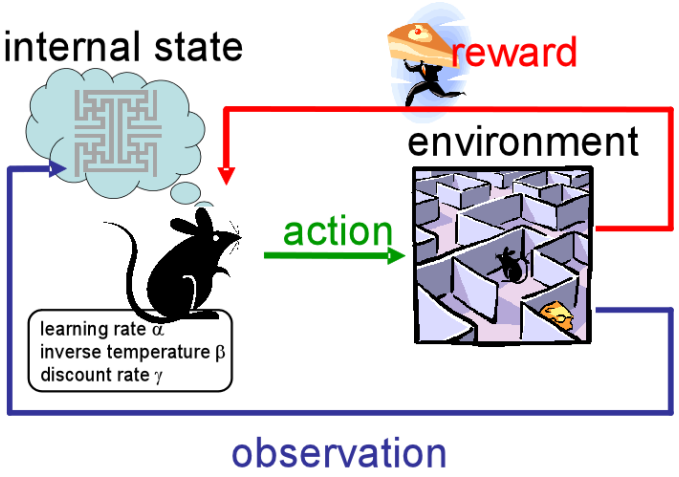
Typy učení v AI

- Unsupervised Learning (Učení bez učitele)
- Jak to funguje?
  - Model se snaží najít strukturu v neoznačených datech (např. shlukování dat).

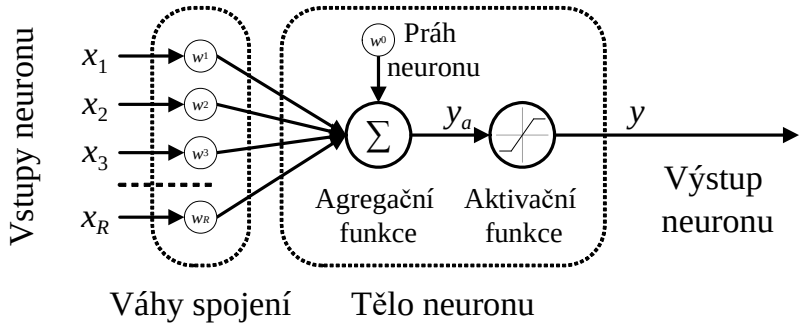


Typy učení v AI

- Reinforcement Learning (Posilované učení)
- Jak to funguje?
  - Model se učí na základě zpětné vazby (odměny a tresty) během interakce s prostředím.



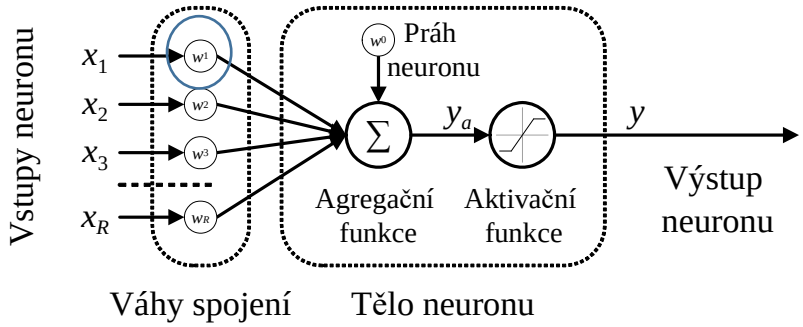
Umělý neuron



Vstupy $x_1, \ldots, x_i, \ldots, x_n$ **modelují dendrity,**
Váhy spojení $w_1, \ldots, w_i, \ldots, w_n$ **modelují synapse,**
Výstup y
simuluje činnost axonu.

Agregace vstupních signálů, prahování a následně jejich nelineární zobrazení představují model těla neuronu, tj. vyhodnocení celkového vstupního aktivačního potenciálu a jeho transformaci na výstupní signál.

Přenos signálu umělým neuronem



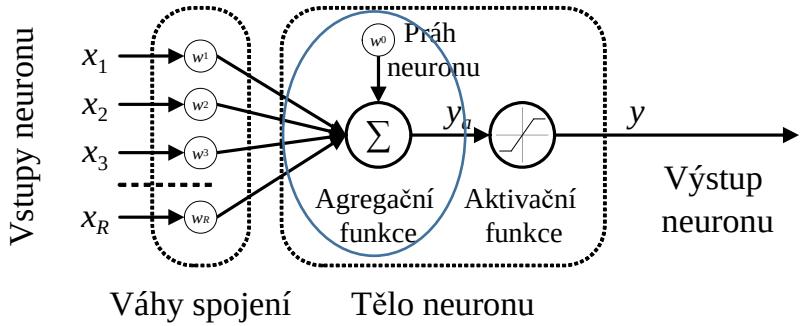
Konfluence - splynutí

$$z_i = x_i \oplus w_i$$

kde $\oplus$ je obecný operátor konfluence

Operátor konfluence $\oplus$ nahradíme prostým součinem hodnot x_i a w_i.

Přenos signálu umělým neuronem



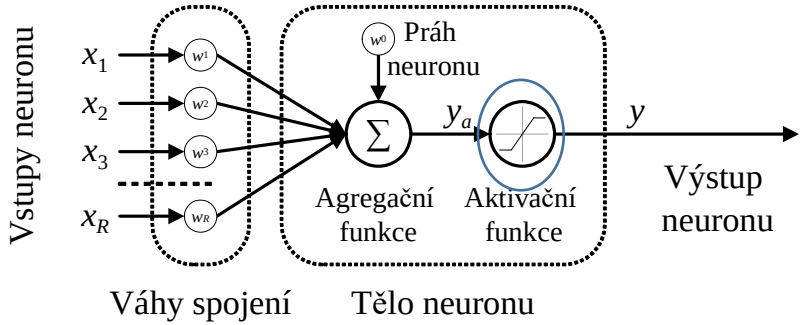
Agregace – seskupení a prahování

$$y_a = \overset{n}{\underset{i=0}{\mathrm{G}}} z_i$$

Operátor agregace G nahradíme prostým součtem (včetně prahu)

$$y_a = \sum_{i=1}^{n} x_i \cdot w_i + w_0$$

Přenos signálu umělým neuronem

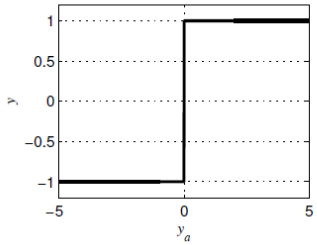


Aktivace – nelineární zobrazení vstupního potenciálu

$$y = \phi(y_a)$$

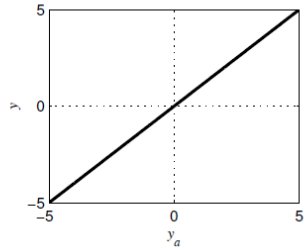
Aktivační funkce

Bipolární skoková funkce



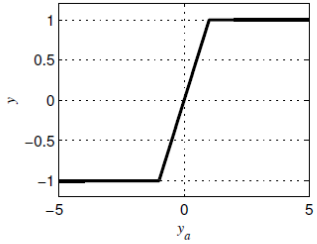
$$y = \begin{cases} 1, & y_a \geq 0 \\ -1, & y_a < 0 \end{cases}$$

Lineární funkce



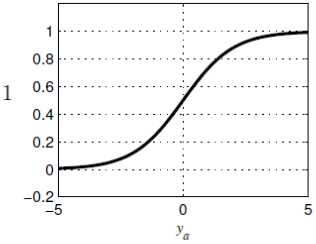
$$y = y_a$$

Lineární saturovaná funkce



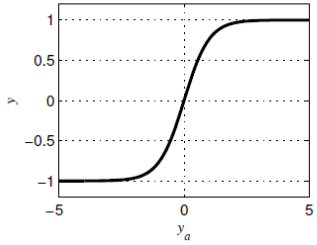
$$y = \begin{cases} 1, & y_a > 1 \\ y_a, & -1 \leq y_a \geq 1 \\ -1, & y_a < -1 \end{cases}$$

Sigmoidální funkce



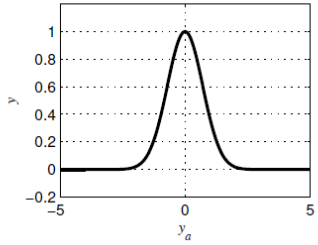
$$y = \frac{1}{1 + e^{-y_a}}$$

Hyperbolicko–tangenciální funkce



$$y = \tanh y_a$$

Gaussova funkce



$$y = e^{-y_a^2}$$

Učení umělé neuronové sítě

Učení

Je proces, kdy se síť přizpůsobuje (adaptuje) vnějšímu prostředí, které na ní působí prostřednictvím dat – vzorů získaných měřením (pozorováním) na objektu, jehož vlastnosti má v konečné fázi reprezentovat, případně na problému, který má následně řešit.

Učení se dělí na trénování, testování a validaci.

Cíl učení

Cílem učení neuronové sítě je nastavit parametry sítě tak, aby dávala požadované výsledky.

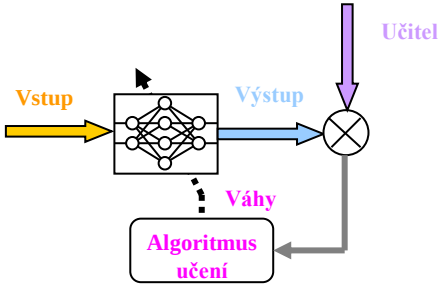
V biologických sítích jsou zkušenosti jsou uloženy v synapsích.
V umělých neuronových sítích jsou zkušenosti uloženy v jejich matematickém ekvivalentu - váhách.

Učení umělé neuronové sítě

Základní typy učení

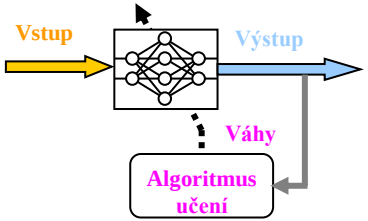
Učení s učitelem

Podobně jako v biologických sítích je zde využita zpětná vazba. Neuronové síti jsou předkládány příslušné vzory. Na základě aktuálního nastavení je zjištěn aktuální výsledek. Ten porovnáme s vyžadovaným výsledkem a určíme chybu. Poté je spočítána nutná korekce (dle typu neuronové sítě) a upraveny hodnoty vah, prahů, případně strmostí aktivačních funkcí, aby se snížila hodnota chyby. Toto se opakuje až do dosažení stanovené minimální chyby.

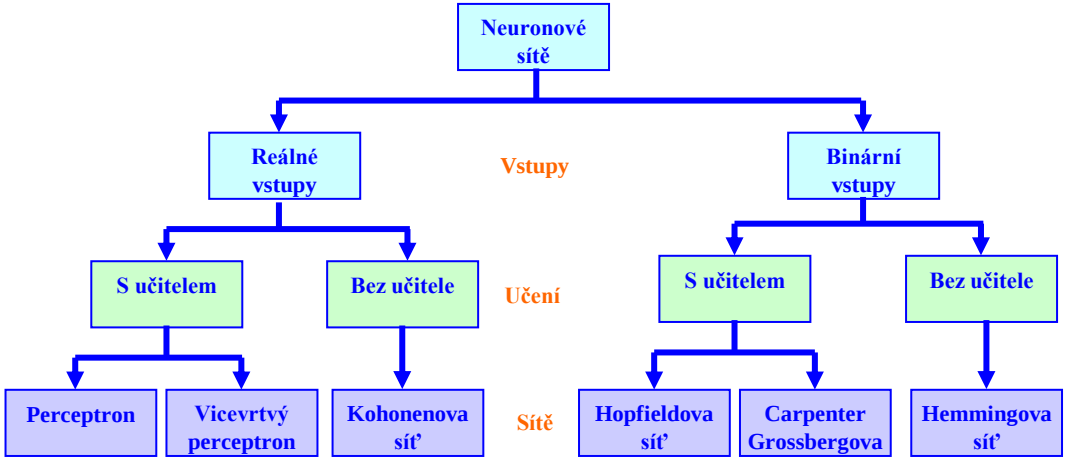


Učení bez učitele

Při učení bez učitele není vyhodnocován výstup. Při tomto učení je výstup dopředu neznámý. Síť dostává na vstup sadu vzorů, které si sama třídí. Buď si vzory třídí do skupin a reaguje na typického zástupce, nebo si přizpůsobí topologii vlastnostem vstupu.

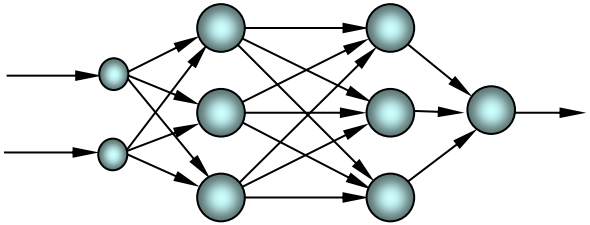


Topologie umělých neuronových sítí



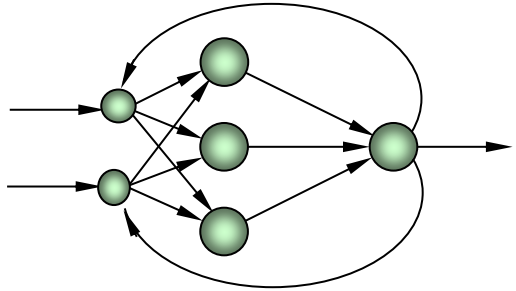
Topologie umělých neuronových sítí

Topologie umělých neuronových sítí



Dopředná neuronová síť

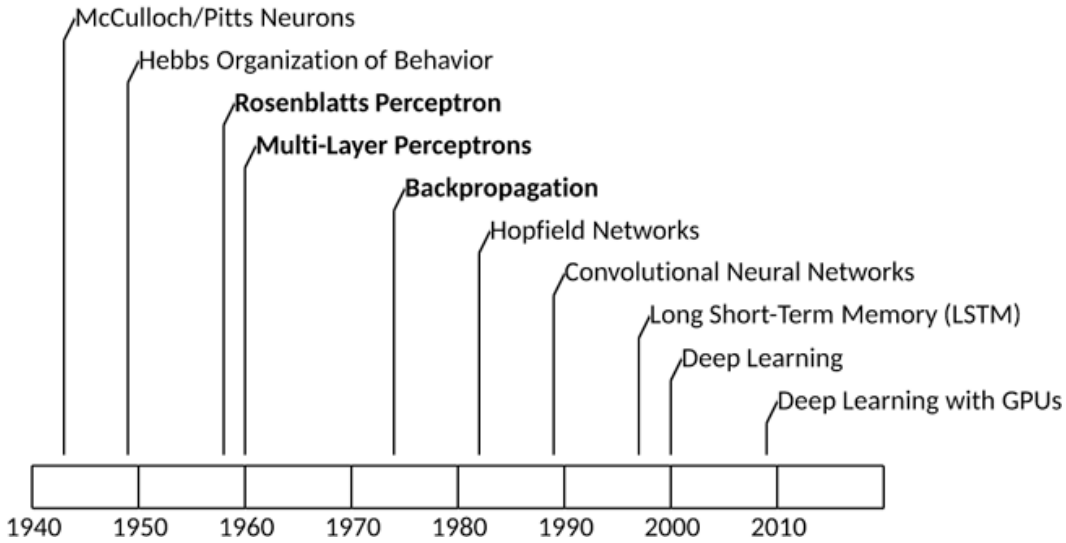
Rekurentní neuronová síť



UNS – některá historická fakta

1943	McCulloch, Pitts	matematický model neuronu – umělý neuron
1949	Hebb	pravidlo pro učení, adaptace vah (synapsí)
1951	Minsky a kol.	Neuropočítač Snark
1957	Rosenblatt	Perceptron, učení
1958	Rosenblatt	Neuropočítač Mark I Perceptron
1959	Windrow, Hoff	Model neuronu ADALINE (ADaptive LInear Neuron)
1962	Windrow, Hoff	Neuronová síť MADALINE (Many ADALINE)
1969	Minsky	Kritika perceptronu, neschopnost modelovat funkci XOR
70. léta		útlum
	Grossberg, Kohonen, Anderson	tichý výzkum
1982, 84	Hopfield	symetrické neuronové sítě
1986	Rumelhart, Hinton, McClelland	Algoritmus zpětného šíření chyby, Backpropagation
1987	Kohonen	samoorganizující se síť (Kohonenova mapa)
1987	Konference v San Diego	1700 účastníků zakládá International Neural Network Society (INNS)

UNS – některá historická fakta



Lineární algebra

Základy vektorů

- Co je vektor?
  - Uspořádaná množina čísel reprezentující bod, směr nebo sílu v prostoru.
- Reprezentace
  - Sloupcový $\begin{pmatrix} 3 \\ 4 \end{pmatrix}$
  - Řádkový $(3 \quad 4)$

Vektory jsou prvky lineárních prostorů a slouží jako základní stavební bloky pro reprezentaci dat i parametrů v neuronových sítích.

Základy vektorů

- Základní operace
  - Sčítání a odčítání

    $(a_1, a_2, \dots, a_n) + (b_1, b_2, \dots, b_n) = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$

    $\boldsymbol{a} = (5, 1, 3), \boldsymbol{b} = (2, 2, 3), \boldsymbol{a} + \boldsymbol{b} = (7, 3, 6)$

  - Skalární násobení

    $k \cdot (a_1, a_2, \dots, a_n) = (k \cdot a_1, k \cdot a_2, \dots, k \cdot a_n)$

    $3 \cdot (2, 1, 4) = (6, 3, 12)$

Základy vektorů

- Základní operace
  - Skalární (dot) součin

$$\boldsymbol{a} = (a_1, a_2, \dots, a_n), \quad \boldsymbol{b} = (b_1, b_2, \dots, b_n)$$
$$\boldsymbol{a} \cdot \boldsymbol{b} = (a_1 \cdot b_1, a_2 \cdot b_2, \dots, a_n \cdot b_n)$$
$$\boldsymbol{a} = (5, 1, 3), \boldsymbol{b} = (2, 2, 3), \boldsymbol{a} \cdot \boldsymbol{b} = (10, 2, 9)$$

  - Norma vektoru - Norma vektoru udává jeho délku nebo velikost.

    Součet absolutních hodnot jednotlivých složek (L1 norma)

    Euklidovská (L2 norma) $\|a\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$

    Více informací o normách s vizualizací

Základy matic

- Co je matice?
  - Uspořádaná tabulka čísel uspořádaných do řádků a sloupců.
  - Definice: Matice $\boldsymbol{A}$ o rozměrech $m \times n$ má m řádků a n sloupců.
- Reprezentace
  - Matice 2×3 $\begin{pmatrix} 1 & 1 & 4 \\ 2 & 5 & 1 \end{pmatrix}$
  - Matice 3×2 $\begin{pmatrix} 2 & 1 \\ 1 & 2 \\ 1 & 5 \end{pmatrix}$

Indexace prvků: Prvek a_{ij} je umístěn na i-tém řádku a j-tém sloupci.

Základy matic

- Základní operace
  - Sčítání a odečítání
    - Operace se provádí po jednotlivých prvcích.
    - Matice musí mít stejné rozměry.

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 4 \\ 2 & 5 & 1 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 2 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \mathbf{A} + \mathbf{B} = \begin{pmatrix} 3 & 3 & 5 \\ 3 & 6 & 3 \end{pmatrix}$$

  - Skalární násobení

$$3 \cdot \begin{pmatrix} 2 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 6 & 3 \\ 3 & 6 \\ 3 & 9 \end{pmatrix}$$

Základy matic

- Základní operace
  - Násobení matic
    - Součin matic $\boldsymbol{A}$ a $\boldsymbol{B}$ (označme $\boldsymbol{A}$ jako $m \times n$ a $\boldsymbol{B}$ jako $n \times p$) je matice $\boldsymbol{C}$ o rozměrech $m \times p$, kde prvek nové matice je

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

    - Počet sloupců matice $\boldsymbol{A}$ musí odpovídat počtu řádků matice $\boldsymbol{B}$.

Základy matic

- Základní operace
  - Násobení matic
    - Počet sloupců matice $\boldsymbol{A}$ musí odpovídat počtu řádků matice $\boldsymbol{B}$.

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Základy matic

- Základní operace
  - Transpozice
    - Operace, která převrací matici přes její hlavní diagonálu.
    - Označuje se A^T .

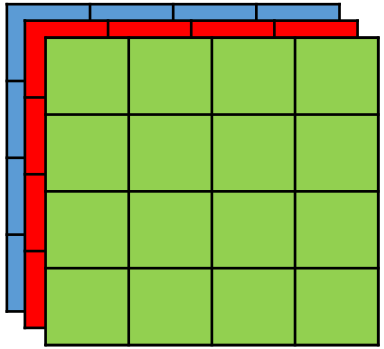
$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \quad \Rightarrow \quad A^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

Základy matic

- Základní operace
  - Inverze
    - Inverzní matice A^{-1} je taková matice, že: $A \cdot A^{-1} = A^{-1} \cdot A = I$
    - Matice A musí být čtvercová.
    - Matice A musí být regulární.

Základy tenzorů

- Co je tenzor?
  - Tenzor je obecnější pojem než matice.
  - Umožňuje reprezentaci dat s více než dvěma rozměry.
  - Rozšiřuje koncepty:
    - Skalár (hodnost 0)
    - Vektor (hodnost 1)
    - Matice (hodnost 2)
    - Tenzor (hodnost ≥ 3)
  - Umožňuje modelovat a zpracovávat data s komplexní strukturou, což je klíčové v moderním strojovém učení.



Základy tenzorů

- Příklady tenzorů
  - Obrazy
    - Obrázek lze reprezentovat jako 3D tenzor: výška, šířka, počet kanálů (např. RGB).
    - Obrázek o rozměrech 256 × 256 pixelů s 3 kanály → tvar tenzoru: [256, 256, 3].
  - Dávky dat (batches)
    - Při trénování neuronových sítí se data často zpracovávají po dávkách.
    - Batch obsahující 32 obrázků (každý s tvarem [256, 256, 3])
      → tvar tenzoru: [32, 256, 256, 3].

Základy tenzorů

- Základní operace
  - Reshaping (přeskupení)
    - Přetvoření tvaru tenzoru bez změny jeho dat.
    - Umožňuje adaptovat data na požadavky konkrétního modelu.
    - Příklad: Z tenzoru tvaru [28, 28, 1] (černobílý obraz) na tenzor tvaru [784] (serializovaný obraz = v jednom vektoru).

Základy tenzorů

- Základní operace
  - Kontrakce (složené násobení)
    - Zobecnění maticového násobení na vyšší dimenze.
    - Probíhá sčítání přes jeden nebo více indexů tenzoru.
  - Sčítání a skalární násobení
    - Provádí se element-wise (po prvcích) stejně jako u matic nebo vektorů.

NumPy

Použití knihovny NumPy

- Základy knihovny NumPy
  - Instalace knihovny – *pip install numpy*
  - Import knihovny ve skriptu – *import numpy as np*
  - Tvorba polí (tenzorů)

```
arr_1d = np.array([1, 2, 3, 4])
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
```

  - Speciální funkce

```
zeros = np.zeros((3, 3))
ones = np.ones((2, 4))
rng = np.arange(0, 10, 2)       # [0, 2, 4, 6, 8]
linsp = np.linspace(0, 1, 5)    # [0, 0.25, 0.5, 0.75, 1.0]
```

Použití knihovny NumPy

- Základy knihovny NumPy
  - Základní vlastnosti

```
print(arr_2d.shape)
print(arr_2d.dtype)
print(arr_2d.ndim)
```

  - Indexování a slicing

```
arr = np.array([10, 11, 12, 13, 14, 15])
print(arr[0])     # první prvek
print(arr[1:4])  # prvky od indexu 1 do 3 (4 se nebere)
print(arr[-1])    # poslední prvek
```

  - Obdobně v maticích a tenzorech

Použití knihovny NumPy

- Základy knihovny NumPy
  - Manipulace s poli – reshape a ravel

```
arr = np.arange(12)
mat = arr.reshape((3, 4))
print(mat
flat = mat.ravel()
print(flat)
```

Použití knihovny NumPy

- Základy knihovny NumPy
  - Manipulace s poli – skládání polí – concatenate, hstack, vstack

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c_concat = np.concatenate((a, b))
c_hstack = np.hstack((a, b))      # pro 1D to bude podobné
print(c_concat)  # [1, 2, 3, 4, 5, 6]
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
AB_v = np.vstack((A, B))        # Vertikální spojení (pod sebe)
AB_h = np.hstack((A, B))        # Horizontální spojení (vedle sebe)
```

Použití knihovny Matplotlib

- Grafické zobrazení
  - Instalace knihovny – *pip install matplotlib*
  - Import knihovny ve skriptu – *import matplotlib.pyplot as plt*
  - Základní graf

```
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
plt.plot(x, y, label='sin(x)‘)
plt.title('Základní sinusovka‘)
plt.xlabel('x‘)
plt.ylabel('sin(x)‘)
plt.legend()
plt.show()
```

Použití knihovny Matplotlib

- Grafické zobrazení
  - Scatter

```
x = np.random.randn(100)
y = np.random.randn(100)
plt.scatter(x, y, c='red', alpha=0.5, label='Náhodné body')
plt.title('Scatter plot ukázka')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Použití knihovny Matplotlib

- Grafické zobrazení
  - Histogram

```
data = np.random.randn(500)
plt.hist(data, bins=20, edgecolor='black', alpha=0.7)
plt.title('Histogram rozložení')
plt.xlabel('Hodnota')
plt.ylabel('Frekvence')
plt.show()
```

Použití knihovny Matplotlib

- Grafické zobrazení
  - Heatmap (imshow/matshow)

```
mat_data = np.random.rand(10, 10)

plt.imshow(mat_data, cmap='viridis')
plt.colorbar(label='Hodnota')
plt.title('Heatmap 10x10')
plt.show()

plt.matshow(mat_data, cmap='viridis')
plt.colorbar(label='Hodnota')
plt.show()
```

Použití knihovny Matplotlib

- Grafické zobrazení
  - Vizualizace vícerozměrných dat (tenzorů) – řez dat

```
# Tenzor 3D: 5 "vrstev" (např. 5 obrázků 10x10)
tensor_3d = np.random.rand(5, 10, 10)

fig, axes = plt.subplots(1, 5, figsize=(15, 3))
for i in range(5):
    axes[i].imshow(tensor_3d[i], cmap='viridis')
    axes[i].set_title(f"Vrstva {i}")
    axes[i].axis('off')
plt.show()
```

Použití knihovny Matplotlib

- Grafické zobrazení
  - Vizualizace vícerozměrných dat (tenzorů) – průměrování

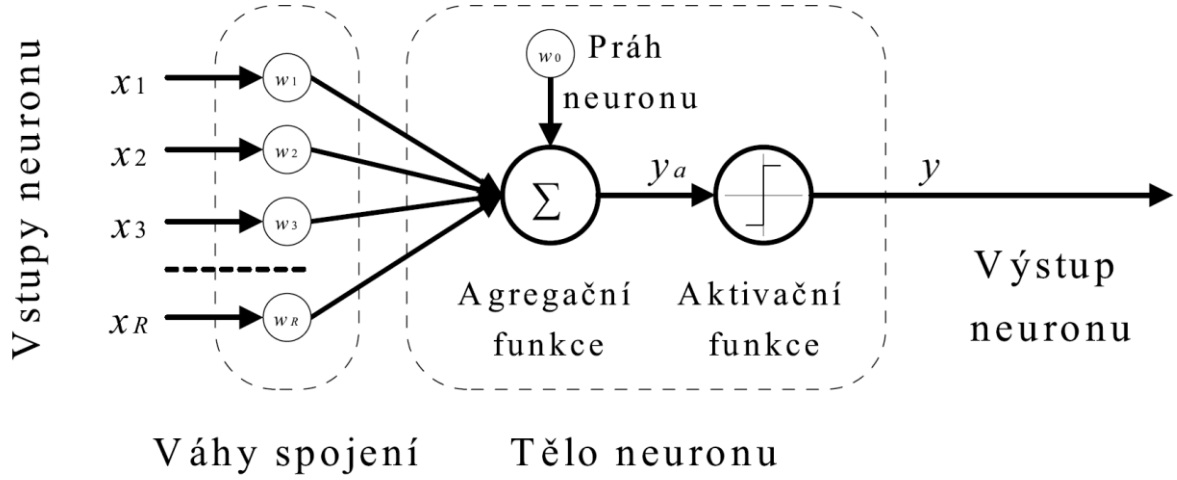
```
# Tenzor 3D: 5 "vrstev" (např. 5 obrázků 10x10)
tensor_3d = np.random.rand(5, 10, 10)

mean_image = tensor_3d.mean(axis=0)  # průměr přes 5 vrstev => výsledkem je 2D (10x10)
plt.imshow(mean_image, cmap='viridis')
plt.title('Průměr 3D dat přes 1. dimenzi')
plt.colorbar()
plt.show()
```

Perceptron

Model perceptronu

- Výkonným prvkem perceptronu je model neuronu s lineárně váženou agregační funkcí, kde původne byla uvažována pouze skoková aktivační funkce



Model perceptronu

- Agregační funkce

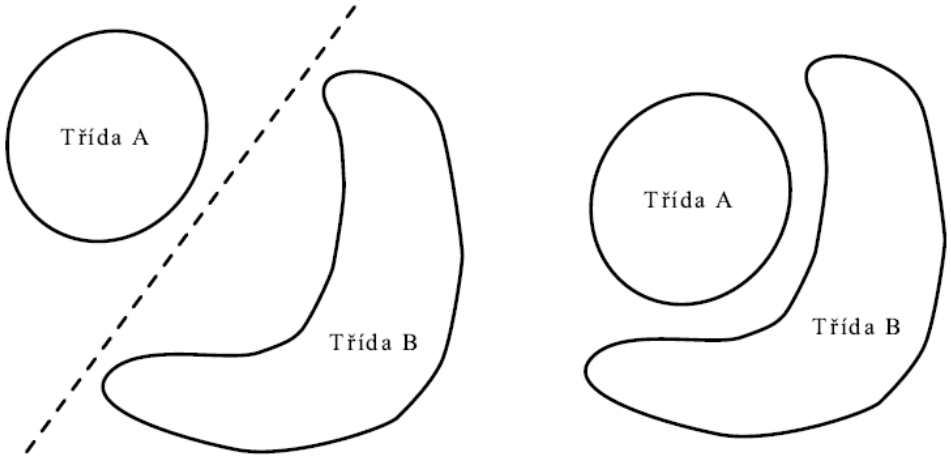
$$y_a = \sum_i x_i \cdot w_i + w_0$$

- Aktivační funkce

$$y = \begin{cases} 1 & \text{pro } y_a > 0 \\ -1 & \text{pro } y_a \leq 0 \end{cases}$$

Využití perceptronu

- Prostřednictvím perceptronu lze řešit pouze problémy, jejichž řešení jsou lineárně separovatelná. Tj. jsme schopni separovat data pouze do dvou skupin, data navíc musejí být oddělitelná nadrovinou.



Odezva perceptronu

- V případě perceptronu je na konkrétní množinu vstupu odpovědí jediná skalární hodnota *y*.

Pro odezvu platí: $y = \phi(y_a)$,

kde ϕ je aktivační funkce

y_a je potenciál neuronu

Pro výpočet potenciálu platí $y_a = \sum_{i=0}^{R} w_i \cdot x_i$

Pro vztah výpočtu potenciálu neuronu lze použít maticový zápis: $y_a = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$

Hebbův zákon učení

- Pokud je hodnota vstupu do neuronu synchronní s očekávaným výstupem, pak se váha spojení mezi příslušným vstupem a neuronem posiluje, pokud je asynchronní (hodnoty nejsou shodné), váha se oslabuje.
- Matematicky toto lze v případě bipolárních vstupů a aktivační funkce zapsat vztahem:

$$w_i = w_i + x_i \cdot t,$$

kde t je očekávaný výstup z neuronu

x_i je vstup

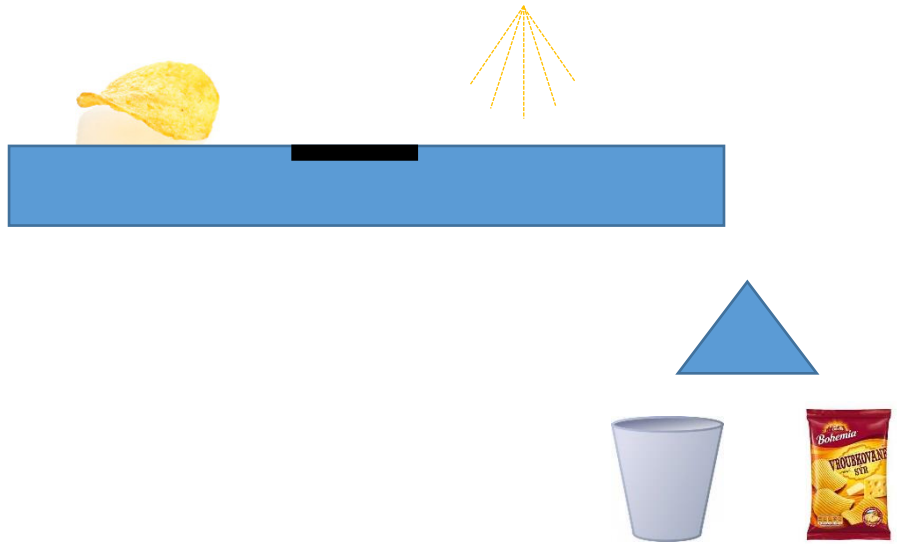
w_i je váha příslušného vstupu

Algoritmus chybového učení perceptronu

- Náhodně nastav váhy neuronu $\mathbf{w}$, zvol koeficient rychlosti učení α.
- Dokud není splněna podmínka zastavení:
  - Pro každý vstup z trénovací množiny:
    - Spočítej výstupní hodnotu neuronu y.
    - Spočítej chybu na výstupu $e = (t - y)$.
    - Adaptuj práh neuronu dle vztahu $w_i = w_0 + \alpha \cdot e$
    - Adaptuj váhy neuronu dle vztahu $w_i = w_i + \alpha \cdot x_i \cdot e$

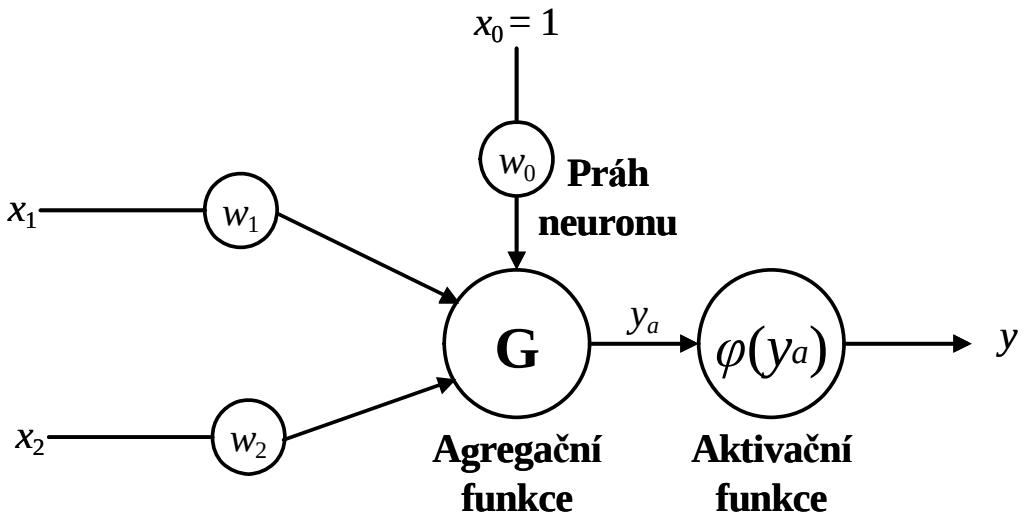
Vzorový příklad – třídění chipsů

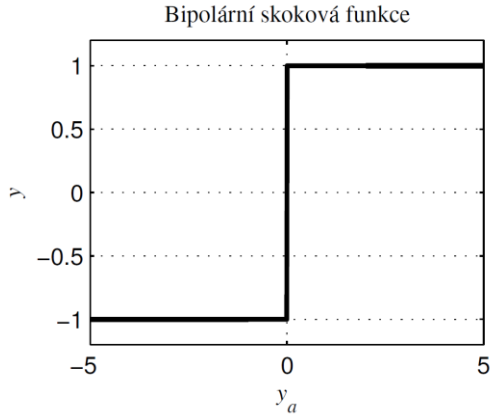
Chceme navrhnout systém pro třídění chipsů. Buď půjde brambůrek do sáčku, nebo do koše.



Vzorový příklad – třídění chipsů

Chceme navrhnout systém pro třídění chipsů. Topologie daného perceptronu:





Vzorový příklad – třídění chipsů

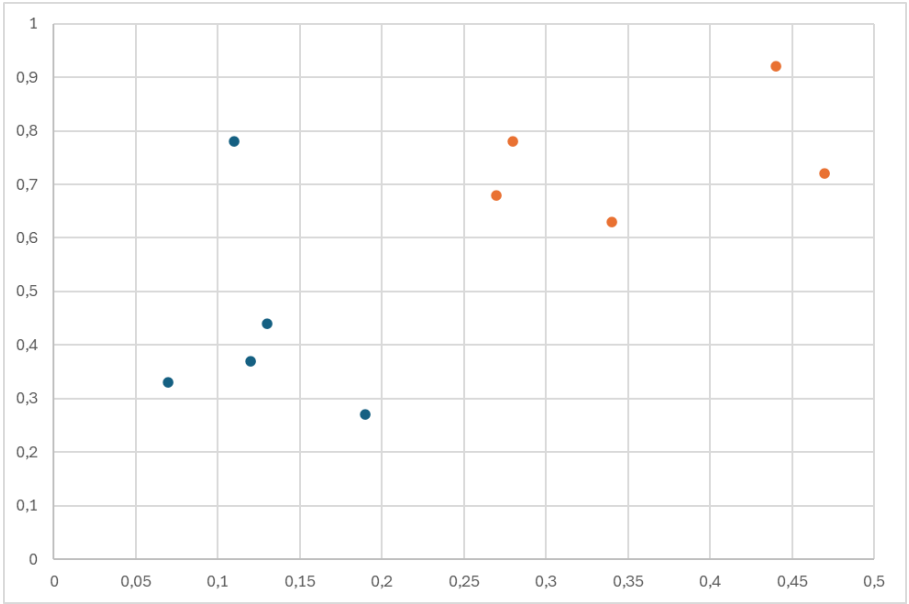
Jedná se o typický příklad učení s učitelem -> tvorba datasetu.

Hmotnost	Intenzita odraženého světla	Výsledek
0,11 g	0,78	Vyřadit
0,27 g	0,68	Ponechat
0,07 g	0,33	Vyřadit
0,12 g	0,37	Vyřadit
0,34 g	0,63	Ponechat
0,47 g	0,72	Ponechat
0,28 g	0,78	Ponechat
0,44 g	0,92	Ponechat
0,13 g	0,44	Vyřadit
0,19 g	0,27	Vyřadit

Vzorový příklad – třídění chipsů

Jedná se o typický příklad učení s učitelem -> tvorba datasetu.

Hmotnost	Intenzita světla	Výsledek
0,11 g	0,78	Vyřadit
0,27 g	0,68	Ponechat
0,07 g	0,33	Vyřadit
0,12 g	0,37	Vyřadit
0,34 g	0,63	Ponechat
0,47 g	0,72	Ponechat
0,28 g	0,78	Ponechat
0,44 g	0,92	Ponechat
0,13 g	0,44	Vyřadit
0,19 g	0,27	Vyřadit



Vzorový příklad – třídění chipsů

Transformace a normalizace trénovací množiny.

Hmotnost	Intenzita odraženého světla	Výsledek
-0,8	0,5692	-1
0	0,2615	1
-1	-0,8154	-1
-0,75	-0,6923	-1
0,35	0,1077	1
1	0,3846	1
0,05	0,5692	1
0,85	1	1
-0,7	-0,4769	-1
-0,4	-1	-1

Hebbův zákon učení

- Pokud je hodnota vstupu do neuronu synchronní s očekávaným výstupem, pak se váha spojení mezi příslušným vstupem a neuronem posiluje, pokud je asynchronní (hodnoty nejsou shodné), váha se oslabuje.
- Matematicky toto lze v případě bipolárních vstupů a aktivační funkce zapsat vztahem:

$$w_i = w_i + x_i \cdot t,$$

kde t je očekávaný výstup z neuronu

x_i je vstup

w_i je váha příslušného vstupu

Algoritmus chybového učení perceptronu

- Náhodně nastav váhy neuronu $\mathbf{w}$, zvol koeficient rychlosti učení α.
- Dokud není splněna podmínka zastavení:
  - Pro každý vstup z trénovací množiny:
    - Spočítej výstupní hodnotu neuronu y.
    - Spočítej chybu na výstupu $e = (t - y)$.
    - Adaptuj práh neuronu dle vztahu $w_i = w_0 + \alpha \cdot e$
    - Adaptuj váhy neuronu dle vztahu $w_i = w_i + \alpha \cdot x_i \cdot e$

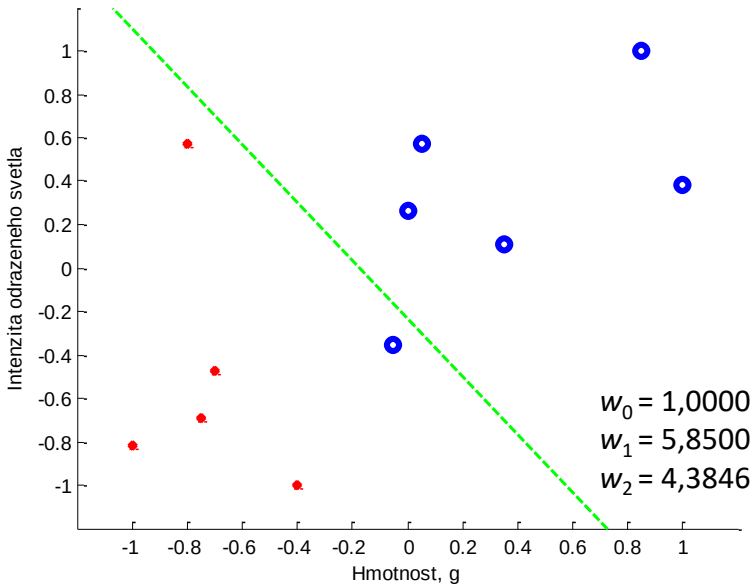
Vzorový příklad – třídění chipsů

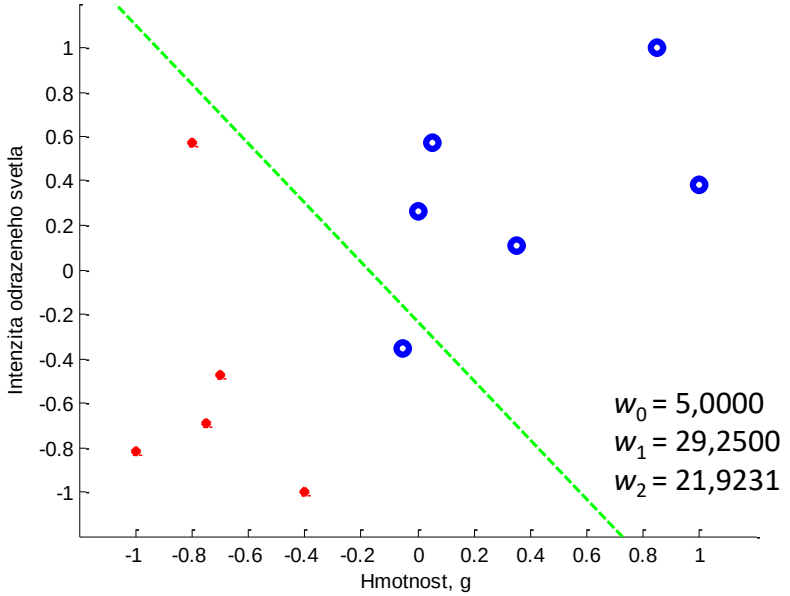
Trénování pomocí Hebbova zákona učení.

Hebbovo učení				
	$w_i =$	w_i	$+ x_i t$	= …
První vzor	$w_0 =$	0	+ 1 (-1)	= -1
	$w_1 =$	0	+ (-0,8)(-1)	= 0,8
	$w_2 =$	0	+ (0,5692)(-1)	= -0,5692
Druhý vzor	$w_0 =$	-1	+ (1)(1)	= 0
	$w_1 =$	0,8	+ (0)(1)	= 0,8
	$w_2 =$	-0,5692	+ (0,2615)(1)	= -0,3077
Třetí vzor	$w_0 =$	0	+ (1)(-1)	= -1
	$w_1 =$	0,8	+ (-1)(-1)	= 1,8
	$w_2 =$	-0,3077	+ (-0,8154)(-1)	= 0,5077
Čtvrtý vzor	$w_0 =$	-1	+ (1)(-1)	= -2
	$w_1 =$	1,8	+ (-0,75)(-1)	= 2,55
	$w_2 =$	0,5077	+ (-0,6923)(-1)	= 1,2

Vzorový příklad – třídění chipsů

Trénování pomocí Hebbova zákona učení. Počet epoch (1 vs. 5).





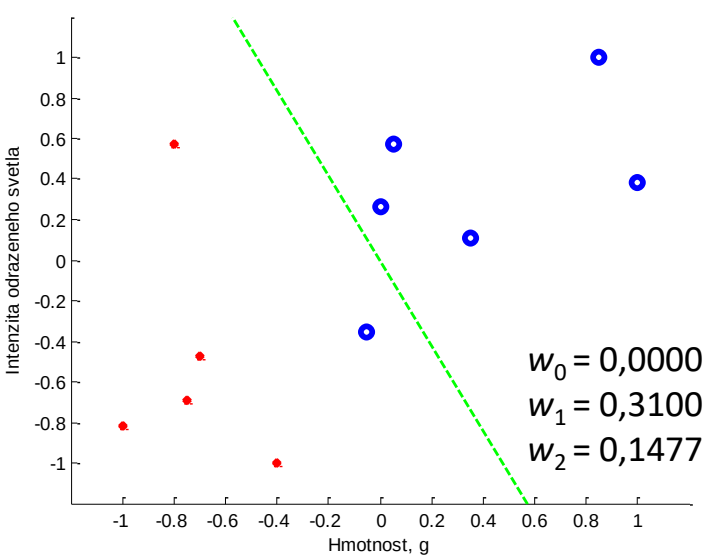
Vzorový příklad – třídění chipsů

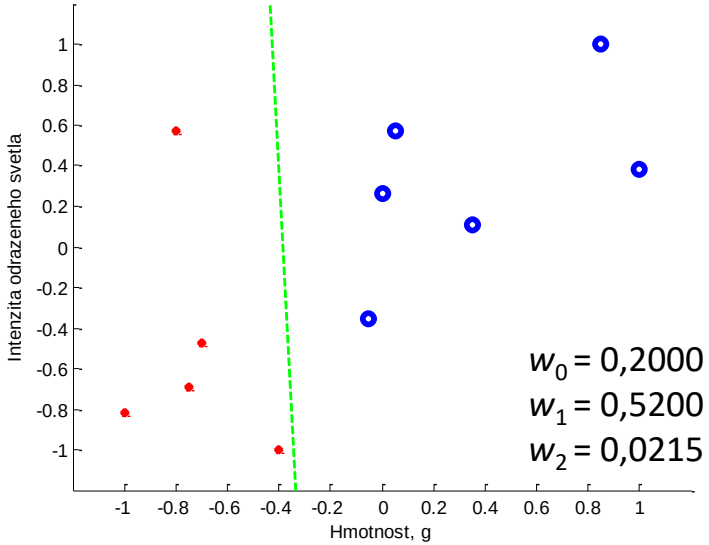
Trénování pomocí chybového učení perceptronu.

Chybové učení				
	$w_i =$	w_i	$+ \alpha x (t - y)$	= …
První vzor	$w_0 =$	0	+ 0,1(-1 - 1)	= -0,2
	$w_1 =$	0	+ 0,1 (-0,8) (-1 - 1)	= 0,16
	$w_2 =$	0	+ 0,1 (0,5692) (-1 - 1)	= -0,1138
Druhý vzor	$w_0 =$	-0,2	+ …	= …
	$w_1 =$	0,16	+ …	= …
	$w_2 =$	-0,1138	+ …	= …

Vzorový příklad – třídění chipsů

Trénování pomocí chybového učení perceptronu. Počet epoch (1 vs. 5).





Vzorový příklad – kvalita sušenek

Hmotnost sušenky (g): Ideální rozmezí je mezi 15 a 17 g.
Průměr sušenky (mm): Optimální hodnota se pohybuje mezi 50 a 55 mm.

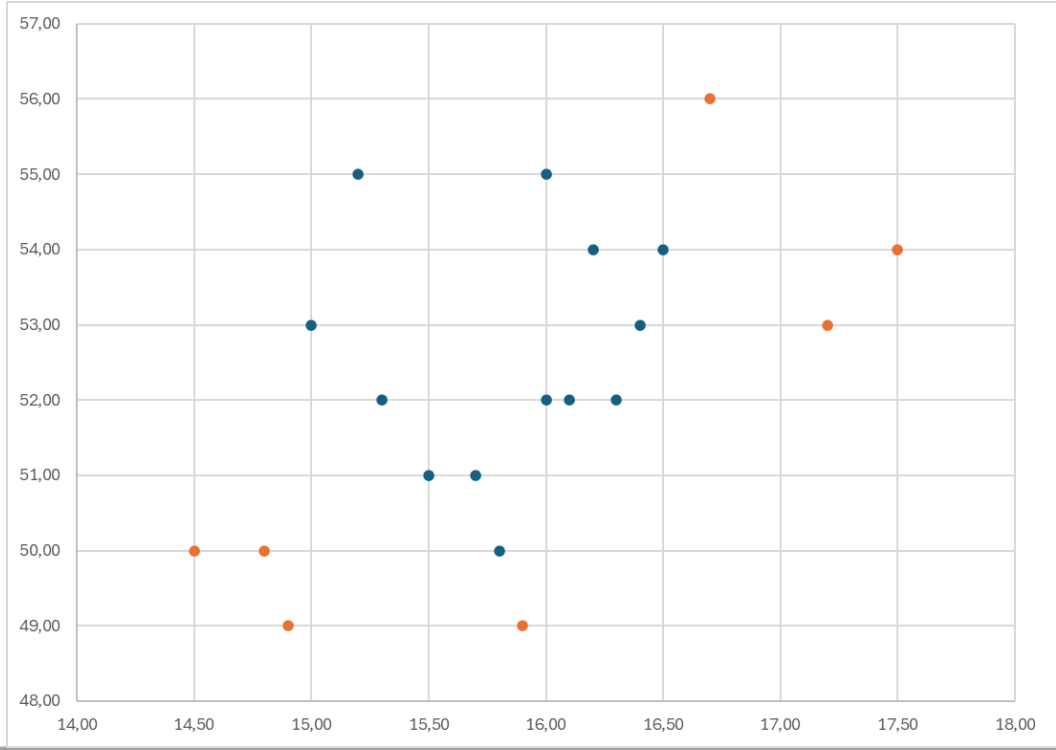
Kvalita = 1: Sušenka má obě hodnoty v optimálním rozmezí.
Kvalita = 0: Alespoň jeden parametr je mimo dané rozpětí.

Hmotnost sušenky (g)	Průměr sušenky (mm)	Kvalita
16.0	52	1
15.5	51	1
14.8	50	0
16.5	54	1
17.2	53	0

Vzorový příklad – kvalita sušenek

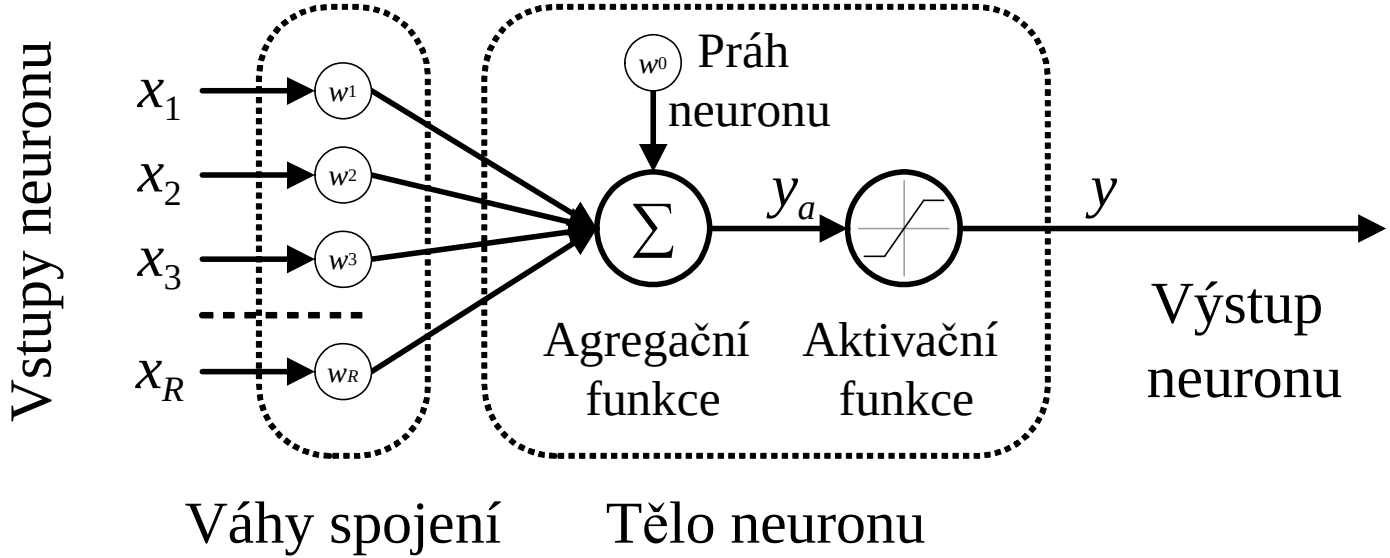
Hmotnost sušenky (g): 15 a 17 g.
Průměr sušenky (mm): 50 a 55 mm.

Co s tím?

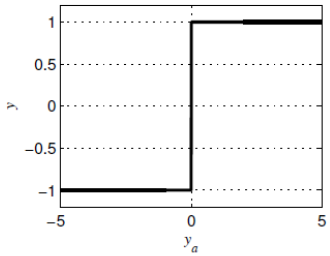


Formální model neuronu

- Oproti samotnému perceptronu mohou být uvažovány i jiné aktivační funkce

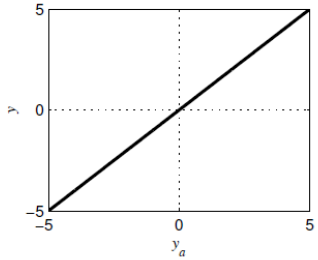


Bipolární skoková funkce



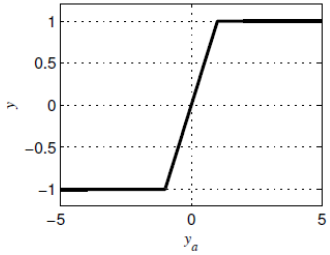
$$y = \begin{cases} 1, & y_a \geq 0 \\ -1, & y_a < 0 \end{cases}$$

Lineární funkce



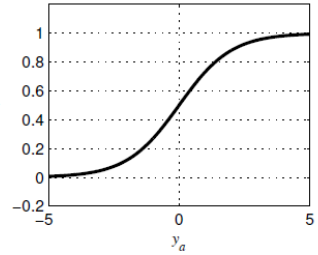
$$y = y_a$$

Lineární saturovaná funkce



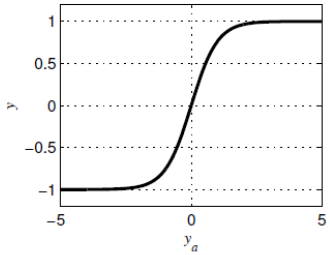
$$y = \begin{cases} 1, & y_a > 1 \\ y_a, & -1 \leq y_a \geq 1 \\ -1, & y_a < -1 \end{cases}$$

Sigmoidální funkce



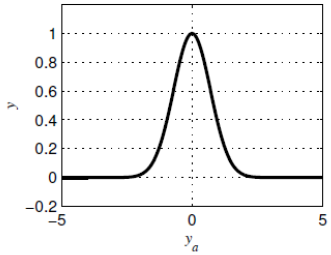
$$y = \frac{1}{1 + e^{-y_a}}$$

Hyperbolicko–tangenciální funkce

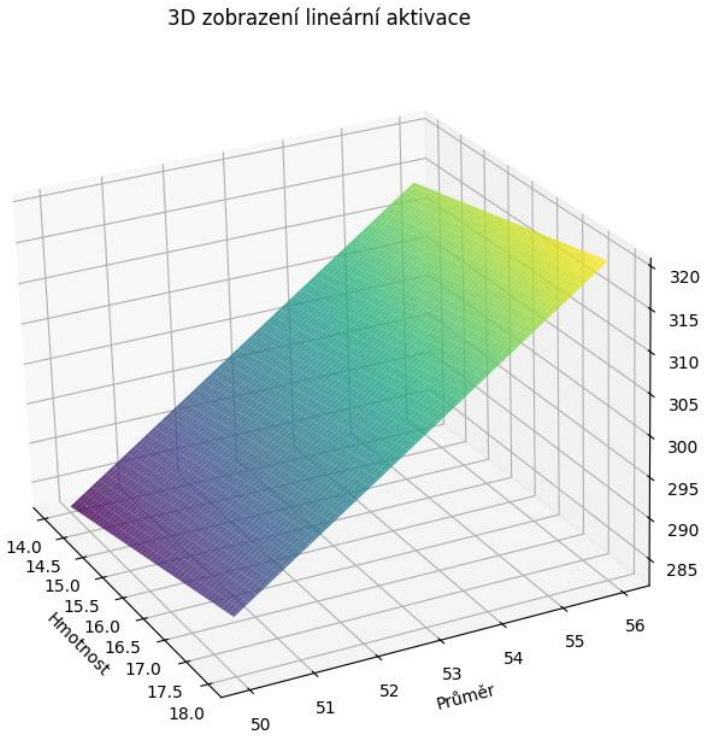


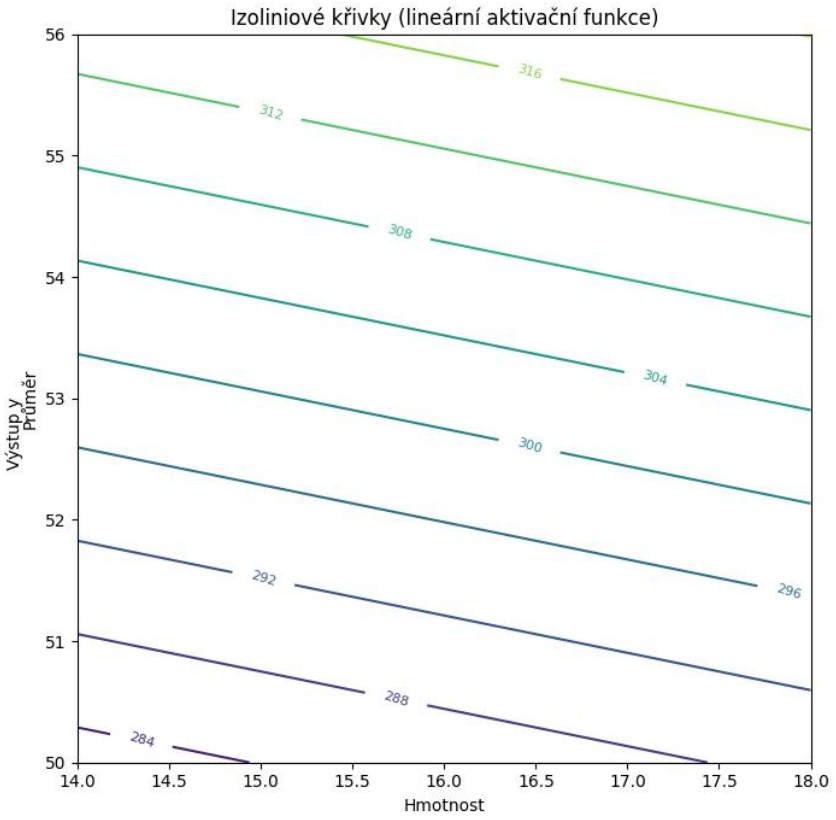
$$y = \tanh y_a$$

Gaussova funkce

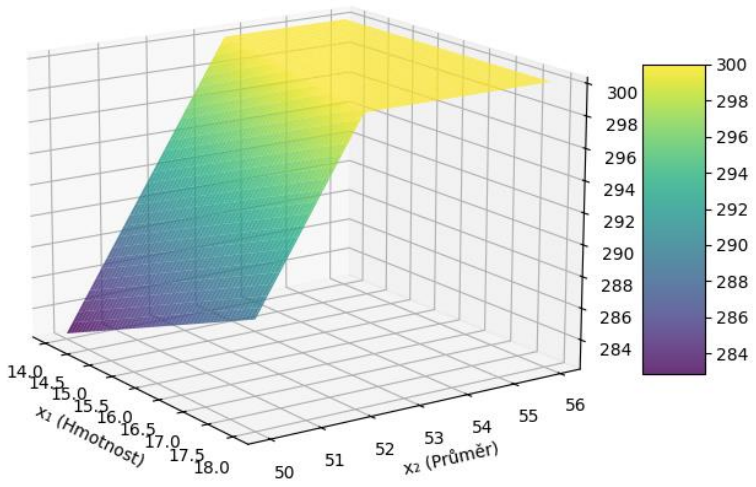


$$y = e^{-y_a^2}$$

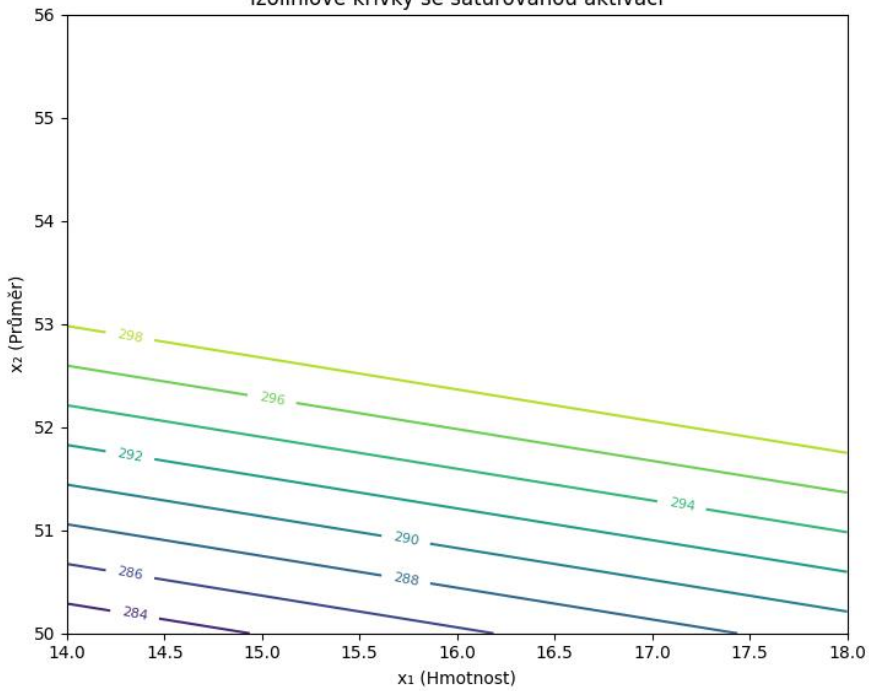

3D zobrazení lineární aktivace
Hmotnost
Průměr
Výstup y
14.0
14.5
15.0
15.5
16.0
16.5
17.0
17.5
18.0
50
51
52
53
54
55
56
285
290
295
300
305
310
315
320

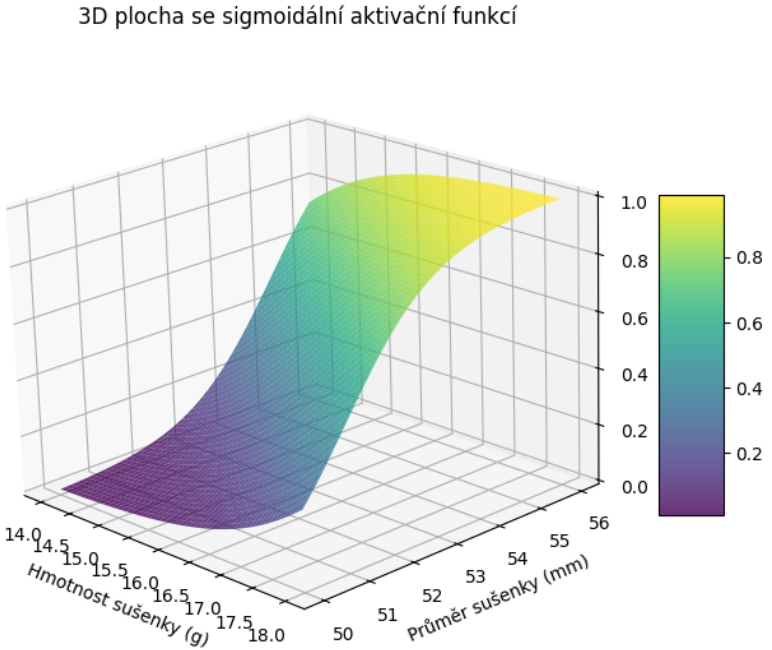

Izoliniové křivky (lineární aktivační funkce)
Hmotnost
Průměr
316
312
308
304
300
296
292
288
284

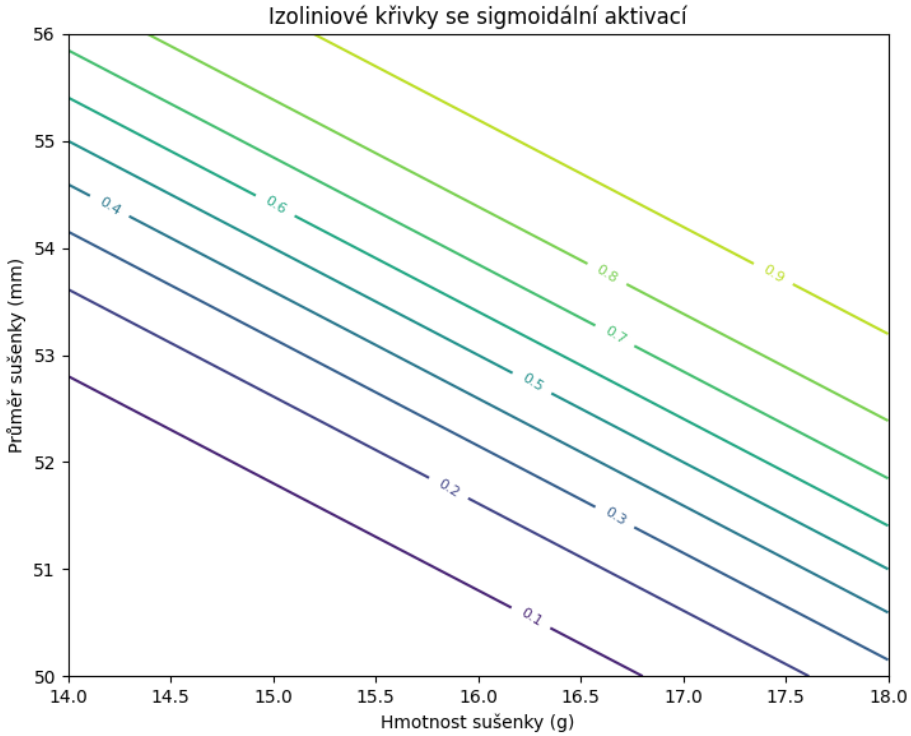
3D plocha se saturovanou lineární aktivační funkcí



Izoliniové křivky se saturovanou aktivací




3D plocha se sigmoidální aktivační funkcí
Hmotnost sušenky (g)
Průměr sušenky (mm)
14.0
14.5
15.0
15.5
16.0
16.5
17.0
17.5
18.0
50
51
52
53
54
55
56
0.0
0.2
0.4
0.6
0.8
1.0


Izoliniové křivky se sigmoidální aktivací
Průměr sušenky (mm)
Hmotnost sušenky (g)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

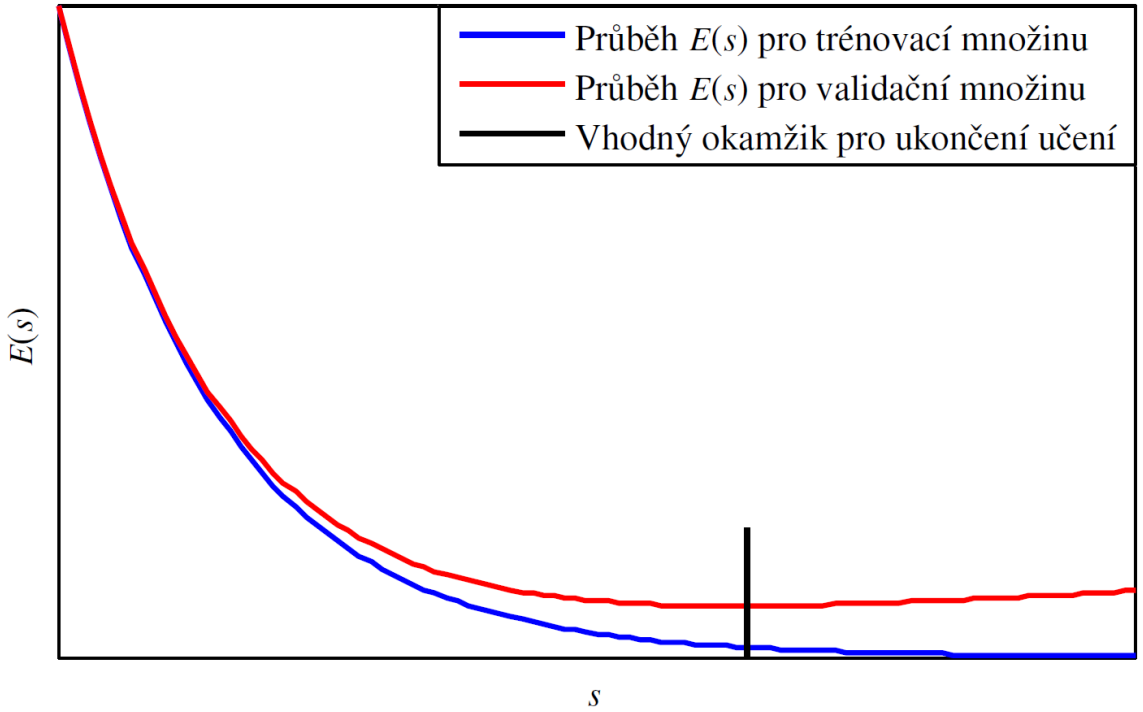
Rozdílnost akt. funkcí – chování výstupu

- Lineární aktivační funkce
  - Výstup: Neomezená reálná hodnota (lineární kombinace vstupů).
  - Interpretace: reprezentuje "sílu aktivace".
- Saturovaná lineární aktivační funkce
  - Výstup: Lineární kombinace vstupů oříznutá do předem daného intervalu.
  - Interpretace: Reprezentuje "sílu aktivace" s omezeným rozsahem.
- Sigmoidální aktivační funkce
  - Výstup: Hodnota v intervalu (např.) (0,1)(0,1).
  - Interpretace: Výstup lze interpretovat jako pravděpodobnost nebo jistotu přiřazení k dané třídě;

Celkový postup algoritmu učení

- Rozdělení dat na testovací, trénovací a validační množinu
- Nastavení vah a prahů sítě, nastavení koeficientu učení α
- Pro každou dvojici: vzor - očekávaný výstup,
  - Spočítání odezvy sítě
  - Spočítání chyby a aktualizace vah a prahů
  - Test ukončení algoritmu
    - Maximální počet epoch
    - Změna hodnoty účelové funkce dále neklesá
    - Dosaženo kapacity sítě – výkon na validační množině se zhoršuje → přetrénování

Proč provádět validaci?



Chyby u FFNN

$$e = \sum_{k=1}^{Q} (t_k - y_k)^2$$

$$E = \frac{1}{N} \sum_{i=1}^{N} e_i$$

N – počet vzorů
Q – počet výstupů

Varianty učení ANN

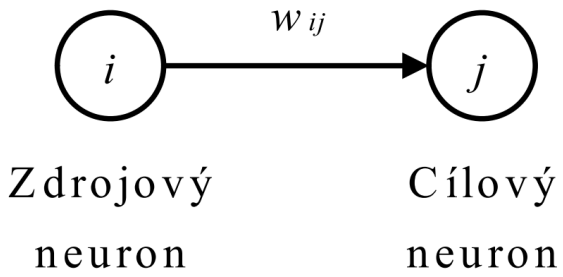
- Online
  - Po každém vzoru dojde k aktualizaci hodnot vah a prahů.
- Offline
  - Změny vah a prahů se sčítají v dočasné proměnné a jejich aktualizace se provede až na konci epochy trénování (po celé epoše).
- Dávkové (batch)
  - Kombinuje předchozí varianty, kde se aktualizace provádí po dávkách vzorků (většinou po 64, 32, 16, 8, 4 vzorech).
  - Kombinuje výhody a nevýhody obou variant (kompromis rychlosti a paměťové náročnosti)

Umělá neuronová síť

Artificial Neural Network (ANN)

Obecné schéma ANN

- Váhy spojení mezi jednotlivými neurony jsou označeny symbolem w_{ij}.
- Příslušné indexy charakterizují spojení od i-tého zdrojového neuronu k j-tému cílovému neuronu.

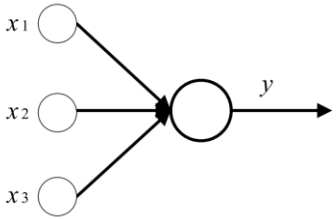


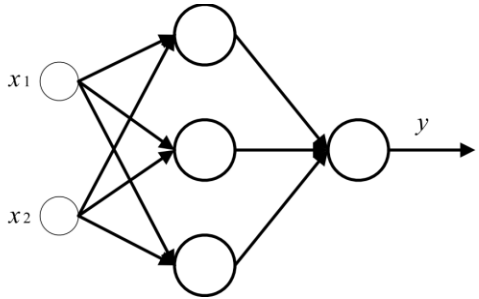
Topologie ANN

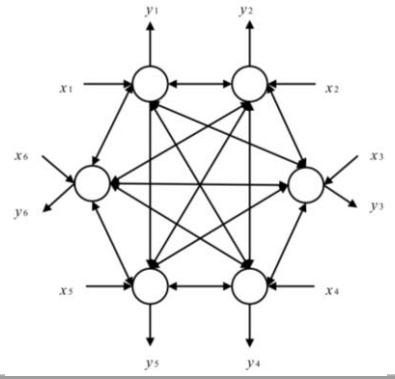
- Topologie (geometrie, struktura, architektura) neuronové sítě popisuje umístění jednotlivých neuronů v síti a jejich propojení.
- Společným rysem je vrstevnatá struktura, kde rozlišujeme:
  - vstupní vrstvu,
  - samotnou vrstvu nebo vrstvy skryté,
  - výstupní vrstvu.
- Topologie je pak definována počtem neuronů v jednotlivých vrstvách, počtem vrstev a vlastními propojeními jednotlivých neuronů/vrstev.

Topologie ANN

- Podle toku signálu ANN dělíme na
  - dopředné, v nichž se signál šíří po orientovaných spojeních jedním směrem (od vstupní vrstvy k výstupní vrstvě);
  - zpětnovazební (rekurentní), u nichž existují mezi neurony, nebo vrstvami zpětné vazby. Je třeba poznamenat, že v těchto sítích je někdy těžko definovat vstupní a výstupní vrstvy.







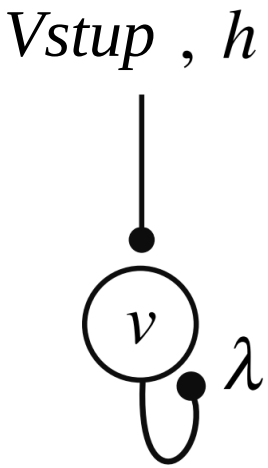
Rekurentní ANN

- Nejjednodušší rekurentní ANN je jeden neuron se zpětnou vazbou.
- Nechť je neuron systém s dynamickým přenosem:

$$\tau_n \frac{dv}{dt} = -v + h + \lambda v$$

$$\tau_n \frac{dv}{dt} = -\underbrace{(1-\lambda)}v + h$$

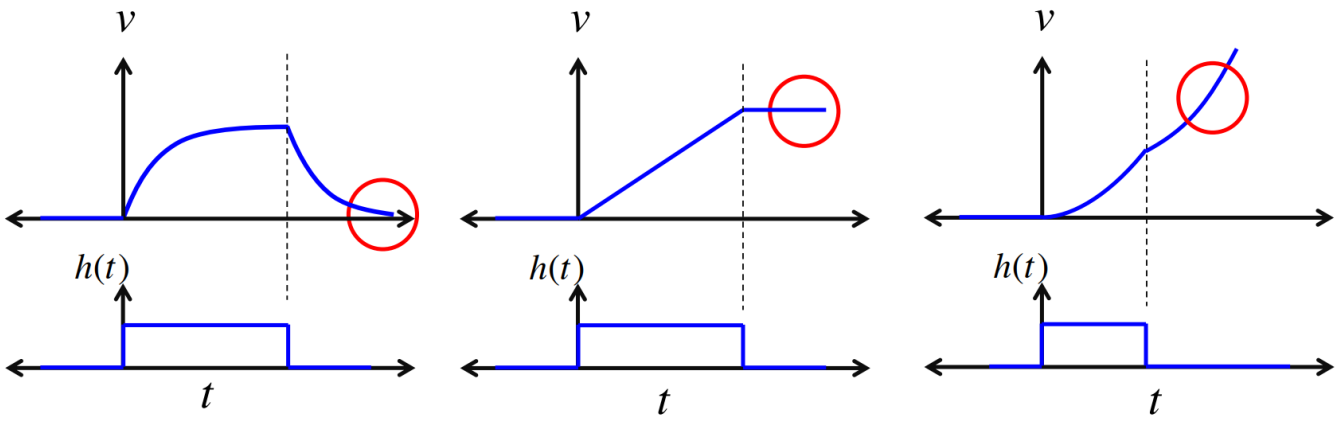
$\lambda < 1$ $\lambda = 1$ $\lambda > 1$



Rekurentní ANN

$$\tau_n \frac{dv}{dt} = -(1-\lambda)v + h$$

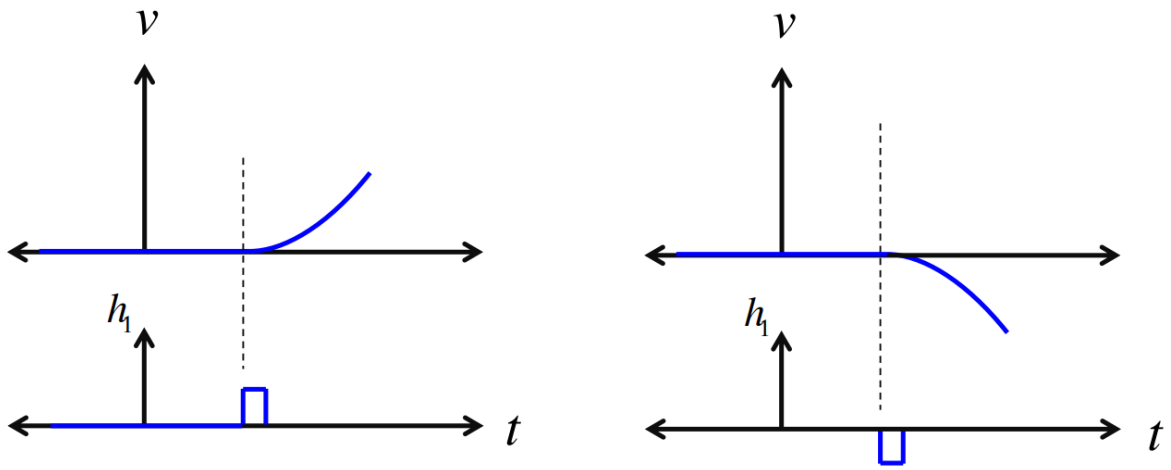
$\lambda < 1$	$\lambda = 1$	$\lambda > 1$
Exponenciální relaxace	Integrace vstupu	Exponenciální růst



Paměť!

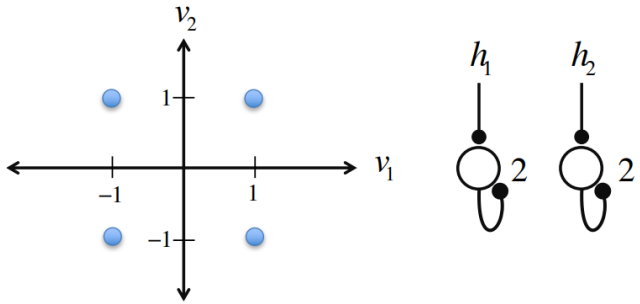
Rekurentní ANN

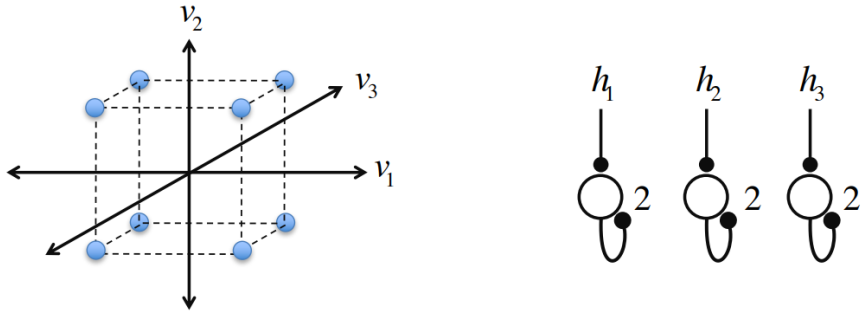
- Atraktor – konečný stav, do kterého dynamický systém časem směřuje.
- Pro $\lambda > 1$ je neuron po přivedení vstupu „přitahován“ do atraktoru.



Rekurentní ANN

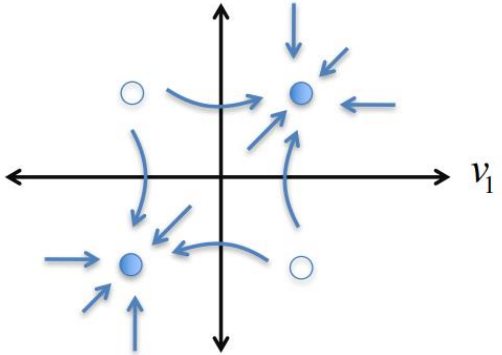
- Pokud výstup neuronu omezíme bipolární aktivační funkcí, tak se nutně ustálí v dané hodnotě výstupu (typicky -1, 1 nebo 0, 1).
- Z jakéhokoli stavu se pak takový neuron ustálí v atraktorech.
- Pro více takových neuronů platí ustálení v nějakém stavu z jejich stavového vektoru (2^n možných stavů).

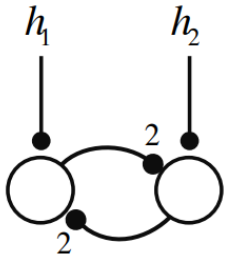


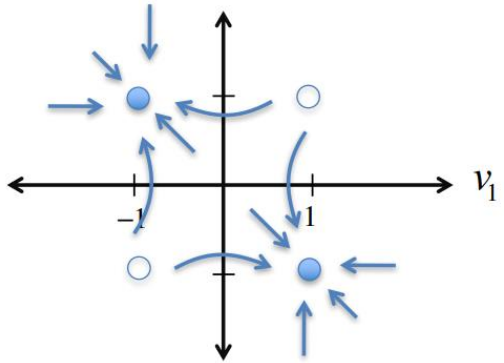


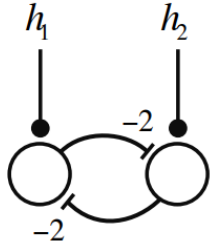
Hopfieldova síť

- Když dané neurony spojíme vzájemně, tak se z některých dříve stabilních stavů stanou stavy nestabilní.
- Podle spojení pak konvergují do vzájemně komplementárních množin z původního stavového prostoru.



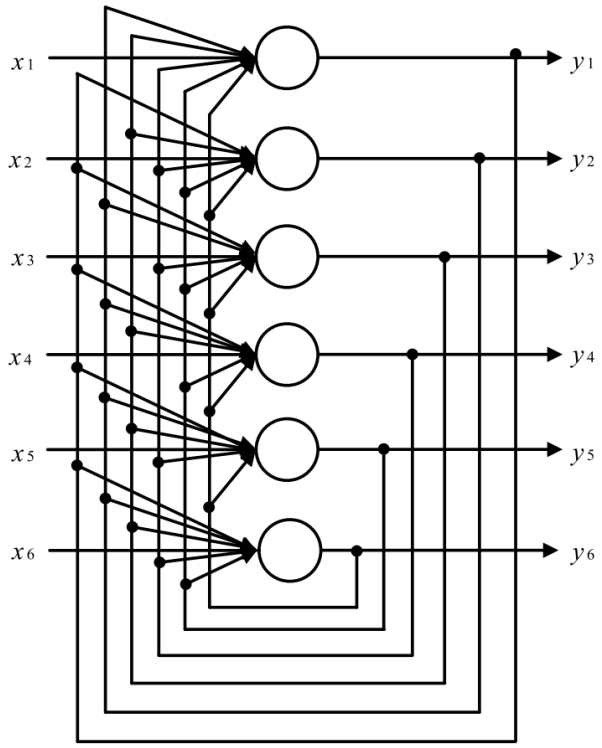






Hopfieldova síť

- Je rekurentní neuronovou sítí s neurony v jedné vrstvě a úplnými váženými spojeními mezi všemi neurony vyjma sebe sama.
- Výstup je závislý nejen na vstupech do sítě, ale také na aktuálních stavech jednotlivých neuronů.
- Samostatné neurony Hopfieldovy sítě nemají práh.



Diskrétní Hopfieldova síť

- Pracuje jako autoasociativní paměť.
- Na základě neúplných informací si tedy dokáže vybavit vzor uložený v paměti.

- Agregační funkce je dána vztahem: $y_a = \sum_{i=1}^{n} x_i \cdot w_i$

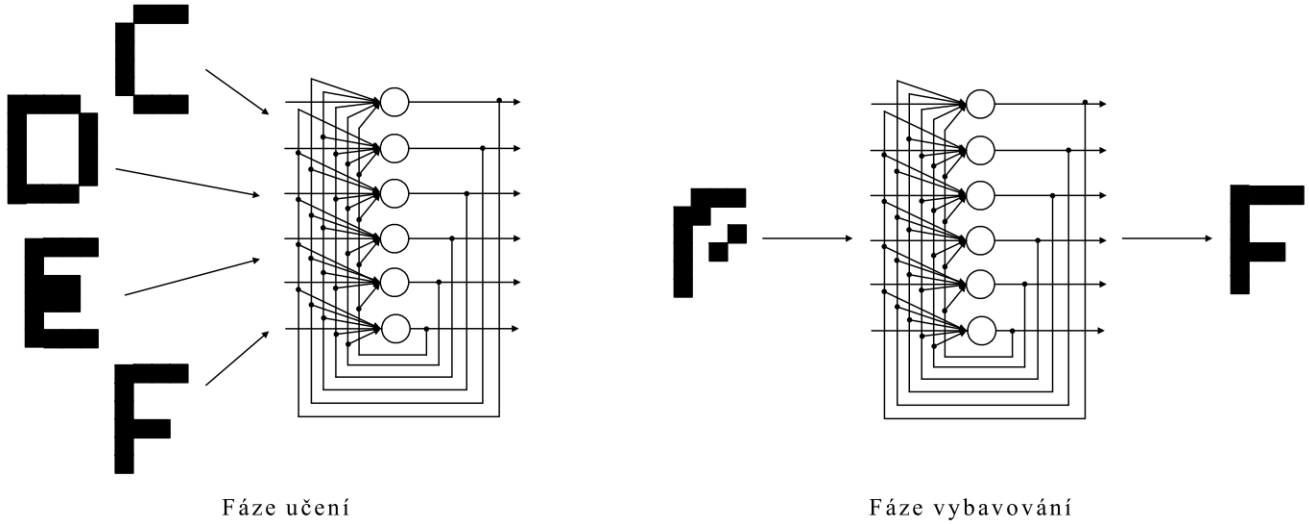
- Pro aktivační funkci platí: $y = \begin{cases} 1 & \text{pro} \quad y_a \geq 0 \\ -1 & \text{pro} \quad y_a < 0 \end{cases}$

Diskrétní Hopfieldova síť

- Hodnoty vstupů i výstupů jsou bipolární.
- Každý neuron tedy vypočte svůj potenciál a aktivuje svůj výstup podle bipolární aktivační funkce.
- Kapacita sítě je nízká, musí být splněno $S < 0.138n$
- Automaticky si pamatuje inverzní vzory
- n … kolika hodnotami je popsán jeden vzor
- S … kolik je vzorů k zapamatování

Diskrétní Hopfieldova síť

- Rozlišujeme mezi fází učení a fází vybavování



Fáze učení

Fáze vybavování

Diskrétní Hopfieldova síť

- V případě, že na vstup je poslána kombinace, která není blízká žádnému ze vzorů, mohou nastat následující situace:
  - síť konverguje ke vzoru, který nebyl součástí paměti (tzv. fantom);
  - síť nekonverguje (osciluje mezi dvěma stavy);
  - síť konverguje k některému ze vzorů uložených v paměti (pokud se vstup nachází v oblasti atrakce některého ze vzorů).
- Aplikace Hopfieldovy sítě je omezena nejen nízkou kapacitou paměti, ale také určitou nejistotou výsledku.

Diskrétní Hopfieldova síť

- Odezva Hopfieldovy sítě jako autoasociativní paměti (vybavování sítě) probíhá iterativním porovnáváním vstupů sítě (v první iteraci) a následně předchozích výstupů (v dalších iteracích) s uloženými vzory podle Hammingovy metriky a za odpověď se bere vzor v paměti, který má tuto vzdálenost nejkratší.
- Síť konverguje do stabilního stavu, který se už dále nemění.
- Hopfieldova síť jednoznačně definovaná jejími vahami, které je zvykem značít symbolem w_{ij} , kde *i* je index neuronu, ze kterého signál vystupuje a *j* je index neuronu, do kterého signál vstupuje.

Učení Hopfieldovy sítě

- Aplikuje se Hebbův zákon učení
- Matice vah **w** je symetrická matice *n* x *n*, kde *n* je počet vstupů do sítě, pro každý prvek matice **w** platí:

$$w_{ij} = \begin{cases} \sum_{k=1}^{s} x_{ki} x_{kj} & pro \quad i \neq j \\ 0 & pro \quad i = j, \end{cases}$$

kde *S je* počet vzorů trénovací množiny

Vybavování Hopfieldovy sítě

1) $y_i(0) = x_i \qquad i = 1 \ldots n$

2) $y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$

3) Krok 2) se opakuje do ustálení

Příklad 1

- Dva vzory k zapamatování

$$\mathbf{x}_1 = [1 \quad 1 \quad 1 \quad -1]$$

$$\mathbf{x}_2 = [1 \quad 1 \quad -1 \quad -1]$$

$$w_{ij} = \begin{cases} \sum_{k=1}^{s} x_{ki} x_{kj} & pro \quad i \neq j \\ 0 & pro \quad i = j, \end{cases}$$

$$w_{12} = x_{11} \cdot x_{12} + x_{21} \cdot x_{22} = 2$$

$$w_{13} = x_{11} \cdot x_{13} + x_{21} \cdot x_{23} = 0$$

$$atd...$$

$$\mathbf{w} = \begin{bmatrix} 0 & x & x & x \\ x & 0 & x & x \\ x & x & 0 & x \\ x & x & x & 0 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

Příklad 1

- Vybavování – 1. iterace

$$\mathbf{test} = \begin{bmatrix} -1 & 1 & -1 & -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

$$y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$$

$$y_1 = f\left(0\cdot(-1) + 2\cdot 1 + 0\cdot(-1) + (-2)\cdot(-1)\right) = f(4) = 1$$

$$y_2 = f\left(2\cdot(-1) + 0\cdot 1 + 0\cdot(-1) + (-2)\cdot(-1)\right) = f(0) = 1$$

$$y_3 = f\left(0\cdot(-1) + 0\cdot 1 + 0\cdot(-1) + 0\cdot(-1)\right) = f(0) = 1$$

$$y_4 = f\left((-2)\cdot(-1) + (-2)\cdot 1 + 0\cdot(-1) + 0\cdot(-1)\right) = f(0) = 1$$

Příklad 1

- Vybavování – 2. iterace

$$y = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

$$y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$$

$$y_1 = f(0\cdot 1 + 2\cdot 1 + 0\cdot 1 + (-2)\cdot 1) = f(0) = 1$$

$$y_2 = f(2\cdot 1 + 0\cdot 1 + 0\cdot 1 + (-2)\cdot 1) = f(0) = 1$$

$$y_3 = f(0\cdot 1 + 0\cdot 1 + 0\cdot 1 + 0\cdot 1) = f(0) = 1$$

$$y_4 = f((-2)\cdot 1 + (-2)\cdot 1 + 0\cdot 1 + 0\cdot 1) = f(-4) = -1$$

Příklad 1

- Vybavování – 3. iterace

$$y = \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

$$y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$$

$$y_1 = f\left(0\cdot 1 + 2\cdot 1 + 0\cdot 1 + (-2)\cdot(-1)\right) = f(4) = 1$$

$$y_2 = f\left(2\cdot 1 + 0\cdot 1 + 0\cdot 1 + (-2)\cdot(-1)\right) = f(4) = 1$$

$$y_3 = f\left(0\cdot 1 + 0\cdot 1 + 0\cdot 1 + 0\cdot(-1)\right) = f(0) = 1$$

$$y_4 = f\left((-2)\cdot 1 + (-2)\cdot 1 + 0\cdot 1 + 0\cdot(-1)\right) = f(-4) = -1$$

Diskrétní Hopfieldova síť

- V případě, že na vstup je poslána kombinace, která není blízká žádnému ze vzorů, mohou nastat následující situace:
  - síť konverguje ke vzoru, který nebyl součástí paměti (tzv. fantom);
  - síť nekonverguje (osciluje mezi dvěma stavy);
  - síť konverguje k některému ze vzorů uložených v paměti (pokud se vstup nachází v oblasti atrakce některého ze vzorů).
- Aplikace Hopfieldovy sítě je omezena nejen nízkou kapacitou paměti, ale také určitou nejistotou výsledku.

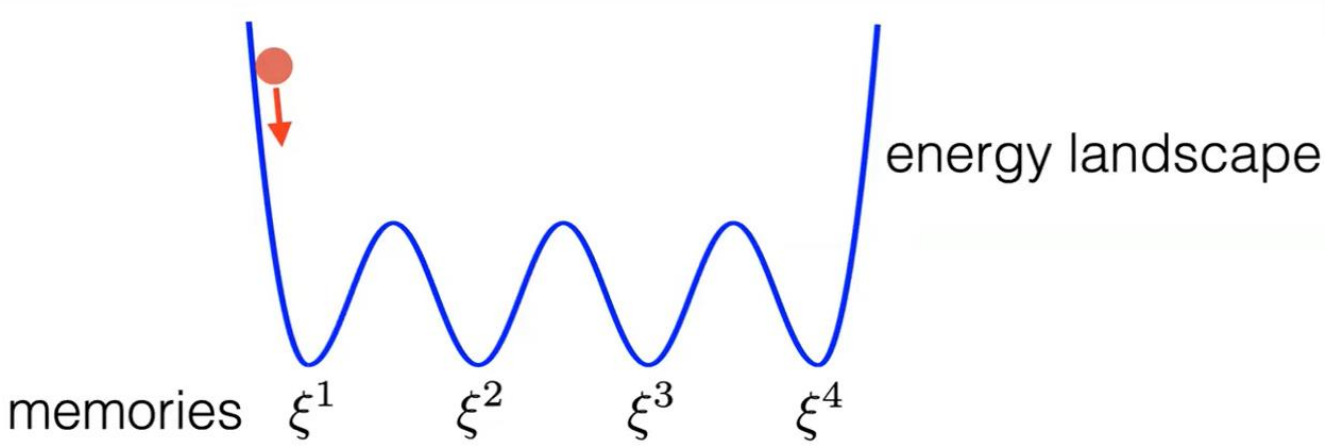
Hopfieldova síť

$$E = -\frac{1}{2}\sum_{i,j} T_{i,j} S_i S_j - \sum_i \theta_i S_i$$

- První část rovnice představuje interakci mezi páry neuronů, kde váhové koeficienty určují, jak moc je interakce mezi dvěma neurony *i* a *j* silná a zda je tato interakce excitační (positivní váha) nebo inhibiční (negativní váha).
- Druhá část rovnice zahrnuje prahy aktivace, které umožňují neuronům změnit svůj stav i bez vstupu od ostatních neuronů.

Hopfieldova síť

- Energetická funkce má lokální minima v místech, kde jsou uloženy vzpomínky (stavy naučené při trénování), ale také „fantomy“



Hopfieldova síť

- Při tvorbě Hopfieldovy sítě pro řešení optimalizačního problému je důležité postupovat systematicky, aby bylo zajištěno, že síť efektivně modeluje a řeší daný problém.
- **Rozpoznání a specifikace problému**: Definice optimalizačního problému k řešení, včetně všech omezení a cílů.
- **Reprezentace Hopfieldovy sítě**
  - Neurony: Každý neuron v síti reprezentuje část řešení problému. Musí být určeno, jak bude struktura problému mapována na neurony v síti.
  - Stav neuronů: Stavy neuronů (obvykle binární, tj. 0 nebo 1) reprezentují, zda je daná část řešení zahrnuta nebo ne.

Hopfieldova síť

- **Energetická funkce**
  - Definice energetické funkce: Energetická funkce je klíčem k modelování optimalizačního problému jako procesu minimalizace energetického stavu sítě. Tato funkce by měla odrážet cíl optimalizace a omezení problému.
  - Penalizace nesplnění omezení: Energetická funkce by měla zahrnovat členy, které penalizují řešení nesplňující omezení problému.

- **Dynamika sítě**
  - Aktualizace neuronů: Definice pravidel pro aktualizaci stavů neuronů s cílem najít konfiguraci minimalizující energetickou funkci. Aktualizace může být asynchronní nebo synchronní.
  - Konvergence: Zavedení mechanismů zajišťujících, že síť konverguje k stabilnímu stavu, který reprezentuje optimální nebo suboptimální řešení.

Hopfieldova síť

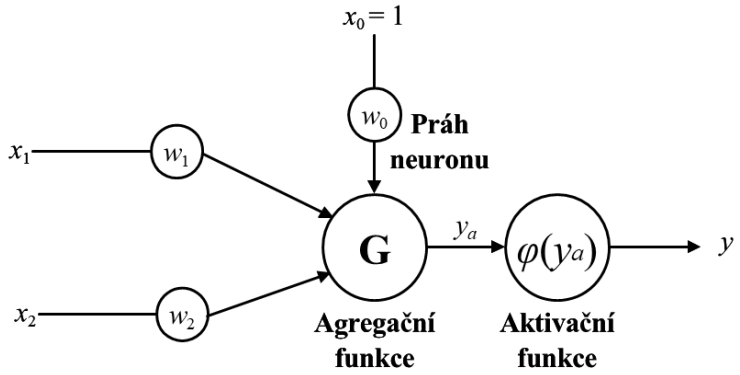
- **Dekódování a validace řešení**
  - Dekódování řešení: Po dosažení stabilního stavu je nutné interpretovat stavy neuronů jako řešení optimalizačního problému.
  - Ověření řešení: Kontrola, zda dekódované řešení splňuje všechna omezení a je optimální nebo přijatelně suboptimální.

- **Experimenty a ladění sítě**
  - Adjustace parametrů: Experimenty s různými hodnotami parametrů sítě (např. váhy spojení, parametry penalizace, prahy aktualizace) pro zlepšení výkonu a kvality řešení.
  - Testování na různých instancích problému: Testování sítě na různých datech nebo instancích problému pro ověření robustnosti a univerzálnosti sítě.

Řešení příkladů

Odezva perceptronu

- Výpočet odezvy je výpočet výstupu perceptronu (y) na vstupní vektor

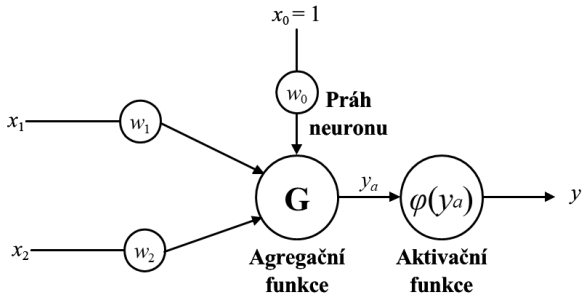


$$\mathbf{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Vypočítejte odezvu daného perceptronu s bipolární skokovou aktivační funkcí na vstup [x_1, x_2]

Odezva perceptronu

- Výpočet odezvy je výpočet výstupu na vstupní vektor



$$\mathbf{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}$$

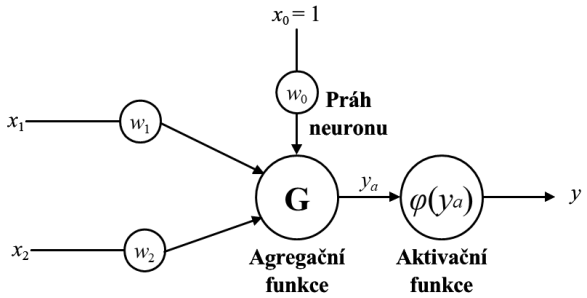
1. Vypočítejte odezvu daného perceptronu s bipolární skokovou aktivační funkcí na vstup [-2, 2]

$$y_a = 1 \cdot -0{,}5 + -2 \cdot 0{,}5 + 2 \cdot -0{,}5 = -0{,}5 - 1 - 1 = -2{,}5$$

$$y_a = -2{,}5 \leq 0 \rightarrow \boldsymbol{y = -1}$$

Odezva perceptronu

- Výpočet odezvy je výpočet výstupu na vstupní vektor



$$\mathbf{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$$

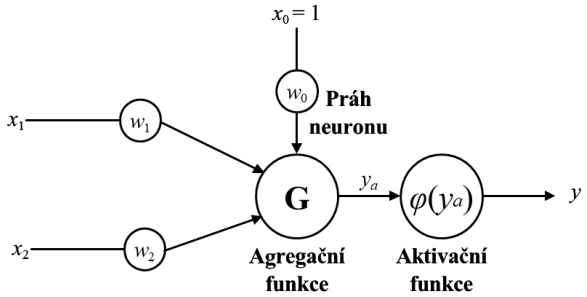
2. Vypočítejte odezvu daného perceptronu s bipolární skokovou aktivační funkcí na vstup [-1, -1]

$$y_a = 1 \cdot 0 + -1 \cdot 1 + -1 \cdot -1 = 0 - 1 + 1 = 0$$

$$y_a = 0 \leq 0 \rightarrow \boldsymbol{y = -1}$$

Odezva perceptronu

- Výpočet odezvy je výpočet výstupu na vstupní vektor



$$\mathbf{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix}$$

3. Vypočítejte odezvu daného perceptronu s bipolární skokovou aktivační funkcí na vstup [1, 2]

$$y_a = 1 \cdot -1 + 1 \cdot -1 + 2 \cdot 2 = -1 - 1 + 4 = 2$$

$$y_a = 2 > 0 \rightarrow \boldsymbol{y = 1}$$

Hebbův zákon učení

- Pokud je hodnota vstupu do neuronu synchronní s očekávaným výstupem, pak se váha spojení mezi příslušným vstupem a neuronem posiluje, pokud je asynchronní (hodnoty nejsou shodné), váha se oslabuje.
- Matematicky toto lze v případě bipolárních vstupů a aktivační funkce zapsat vztahem:

$$w_i = w_i + x_i \cdot t,$$

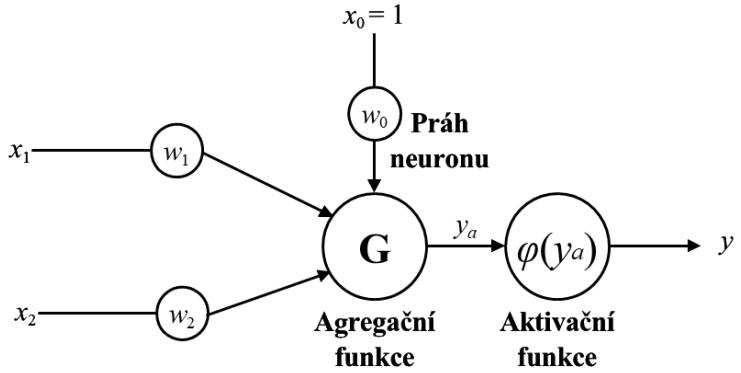
kde t je očekávaný výstup z neuronu

x_i je vstup

w_i je váha příslušného vstupu

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzor trénovací množiny. Trénování proveďte s pomocí Hebbova zákona učení.



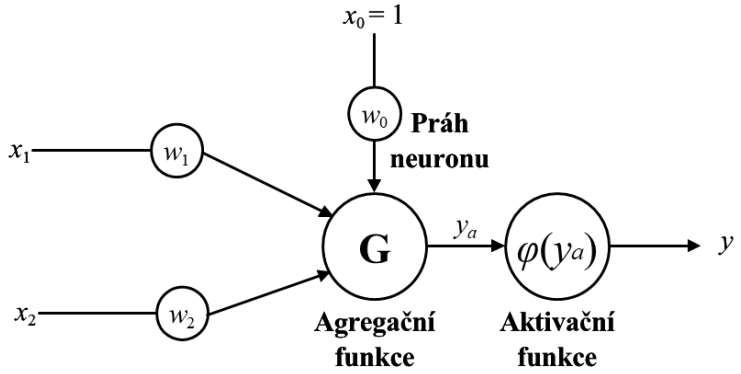
vz1: $\boldsymbol{x} = [x_1, x_2], y = t$

$$\mathbf{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

1. Lze rovnou provést trénování aplikací přičtením výrazu $x_i \cdot t$ k aktuální váze w_i

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzor trénovací množiny. Trénování proveďte s pomocí Hebbova zákona učení.



vz1: $\boldsymbol{x} = [1, 1], t = 1$

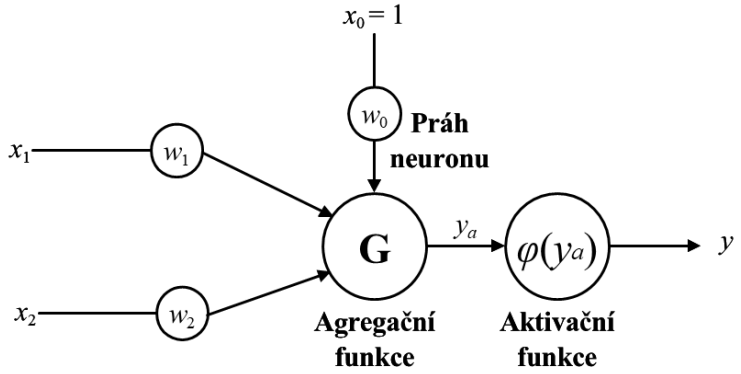
$$\mathbf{W} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}$$

1. Můžeme rovnou provést trénování

$$\mathbf{W} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}, \mathbf{W_n} = \begin{bmatrix} -0{,}5 + 1 \cdot 1 \\ 0{,}5 + 1 \cdot 1 \\ -0{,}5 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 0{,}5 \\ 1{,}5 \\ 0{,}5 \end{bmatrix} \leftarrow \text{Výsledné váhy}$$

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzor trénovací množiny. Trénování proveďte s pomocí Hebbova zákona učení.



vz1: $\boldsymbol{x} = [-2, 1], t = -1$

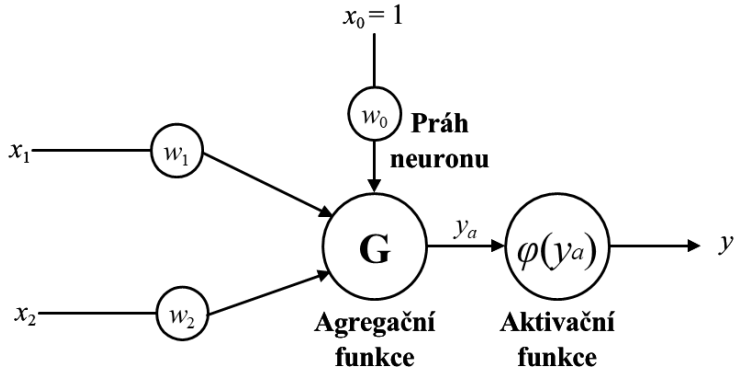
$$\mathbf{W} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

1. Můžeme rovnou provést trénování

$$\mathbf{W} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \mathbf{W_n} = \begin{bmatrix} -1 + 1 \cdot -1 \\ 0 + -2 \cdot -1 \\ 1 + 1 \cdot -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 0 \end{bmatrix}$$ ← Výsledné váhy

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzory trénovací množiny. Trénování proveďte s pomocí Hebbova zákona učení.



vz1: $\boldsymbol{x} = [-1, 1], t = -1$

vz2: $\boldsymbol{x} = [2, -1], t = 1$

$$\mathbf{W} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

1. Můžeme rovnou provést trénování

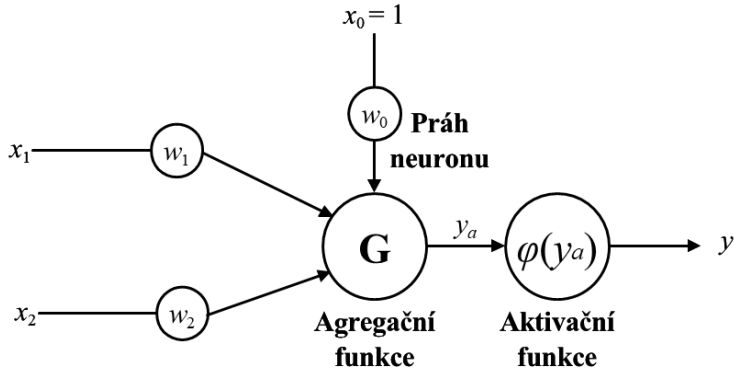
$$\mathbf{W} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \mathbf{W_{n_1}} = \begin{bmatrix} -1 + 1 \cdot -1 \\ 0 + -1 \cdot -1 \\ 1 + 1 \cdot -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} \rightarrow \mathbf{W_{n_2}} = \begin{bmatrix} -2 + 1 \cdot 1 \\ 1 + 2 \cdot 1 \\ 0 + -1 \cdot 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix}$$ ← Výsledné váhy

Algoritmus chybového učení perceptronu

- Náhodně nastav váhy neuronu $\mathbf{w}$, zvol koeficient rychlosti učení α.
- Dokud není splněna podmínka zastavení:
  - Pro každý vstup z trénovací množiny:
    - Spočítej výstupní hodnotu neuronu y.
    - Spočítej chybu na výstupu $e = (t - y)$.
    - Adaptuj práh neuronu dle vztahu $w_i = w_0 + \alpha \cdot e$
    - Adaptuj váhy neuronu dle vztahu $w_i = w_i + \alpha \cdot x_i \cdot e$

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzor trénovací množiny. Trénování proveďte s pomocí chybového učení při $\alpha = 1$.



vz1: $\boldsymbol{x} = [1, 1], t = 1$

$$\mathbf{W} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}$$

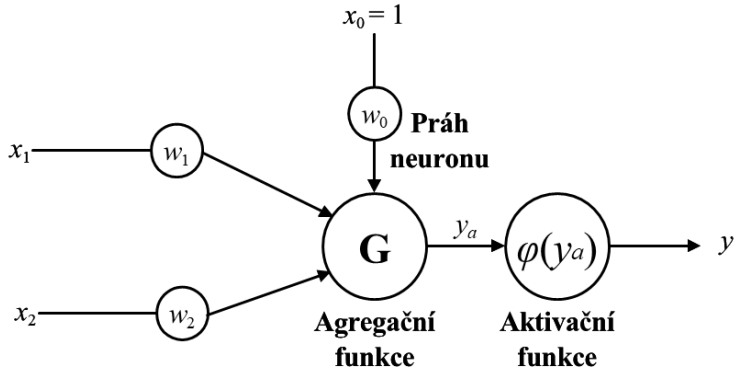
1. Výpočet odezvy perceptronu na vstup [1, 1]

$$y_a = 1 \cdot -0{,}5 + 1 \cdot 0{,}5 + 1 \cdot -0{,}5 = -0{,}5 + 0{,}5 - 0{,}5 = -0{,}5$$

$$y_a = -0{,}5 < 0 \rightarrow \boldsymbol{y = -1}$$

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzor trénovací množiny. Trénování proveďte s pomocí chybového učení při $\alpha = 1$.



vz1: $\boldsymbol{x} = [1, 1], t = 1$

$$\mathbf{W} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}$$

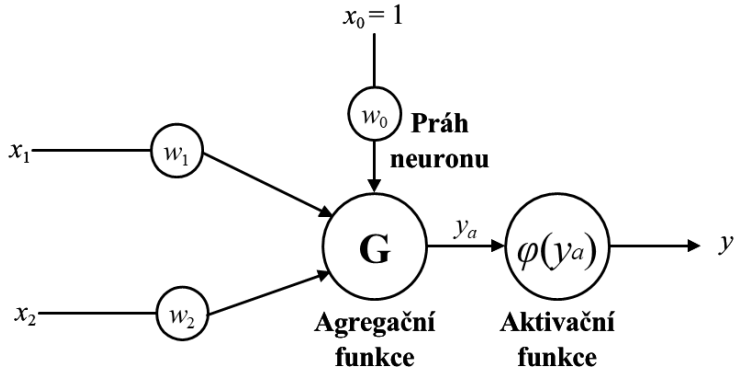
2. Výpočet chyby perceptronu dle daného vzoru: vstup [1, 1], očekávaný výstup 1

Odezva (viz krok 1) $y = -1$

Chyba $e = t - y = \big(1 - (-1)\big) = 2$

Trénování perceptronu

- Mějme model perceptronu s bipolární skokovou aktivační funkcí a následující vzor trénovací množiny. Trénování proveďte s pomocí chybového učení při $\alpha = 1$.



vz1: $\boldsymbol{x} = [1, 1], t = 1$

$$\mathbf{W} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}$$

3. Aktualizace vah dle $w_i = w_i + \alpha \cdot x_i \cdot e$ Chyba (viz krok 2) $e = 2$

$$\mathbf{W} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}, \mathbf{W_n} = \begin{bmatrix} -0{,}5 + 1 \cdot 2 \\ 0{,}5 + 1 \cdot 2 \\ -0{,}5 + 1 \cdot 2 \end{bmatrix} = \begin{bmatrix} -1{,}5 \\ 2{,}5 \\ -1{,}5 \end{bmatrix}$$ ← Výsledné váhy

Samoorganizační mapy

Samoorganizační mapy

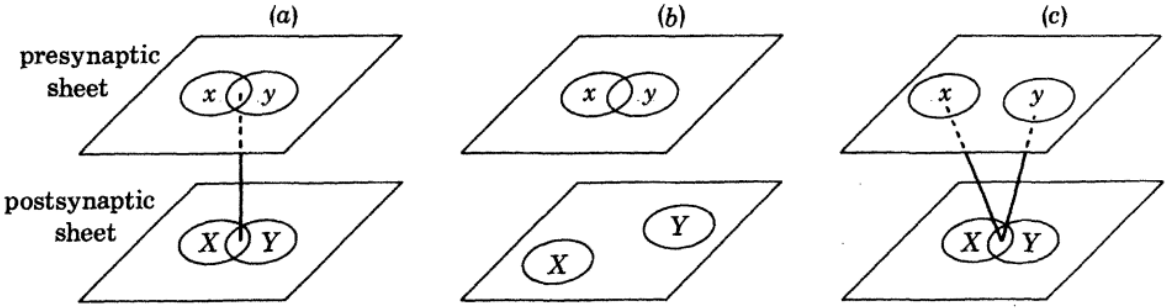
- Využívají kompetitivní učení.
- Jednotlivé výstupní neurony mezi sebou „soutěží“ o aktivitu.
- V aktuální časový okamžik je aktivní pouze jeden výstupní neuron.

Tato skupina neuronový sítí vychází ze 2 publikací:

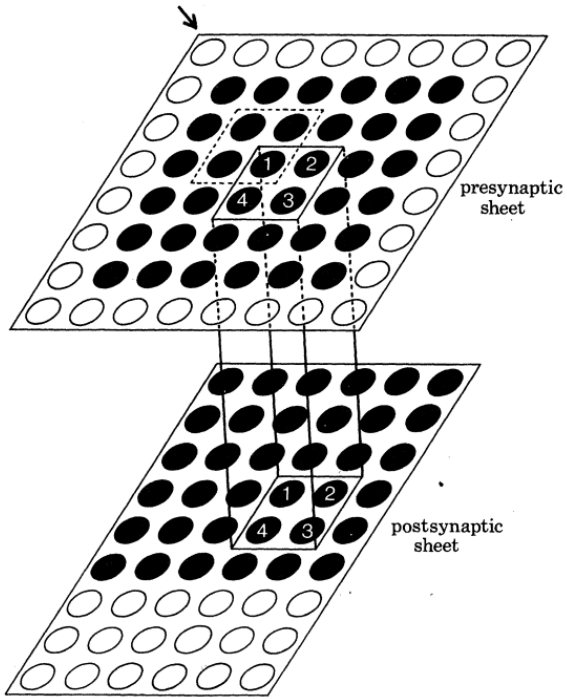
- Willshaw, D.J., Von Der Malsburg, C. How patterned neural connections can be set up by self organization (1976) Proceedings of the Royal Society of London - Biological Sciences, 194 (1117), pp. 431-445. Cited 497 times.
- Kohonen, T. Self-organized formation of topologically correct feature maps (1982) Biological Cybernetics, 43 (1), pp. 59-69. Cited 6008 times.

Willshaw

- Definoval základy pro samoorganizační mapy na základě pozorování biologických procesů.
  - Budeme hovořit o dvou dvourozměrných listech prvků, které mají představovat presynaptický a postsynaptický list nervových buněk.
  - Presynaptické elementy jsou schopny vyslat axony a vytvořit synapse s elementy postsynaptického listu, čímž se vytvoří mapování mezi oběma listy.



Willshaw



Kohonenova samoorganizační mapa

- Jedná se o neznámější architekturou samoorganizačních map.
- Kromě vstupních terminálů má tato architektura pouze jednu vrstvu.
- Ta je označována jako kompetiční a bývá uspořádána do netriviální struktury (zpravidla nějakého typu dvojrozměrné mřížky).
- Struktura určuje, které uzly spolu sousedí, což je důležité při učení.

Kohonenova samoorganizační mapa

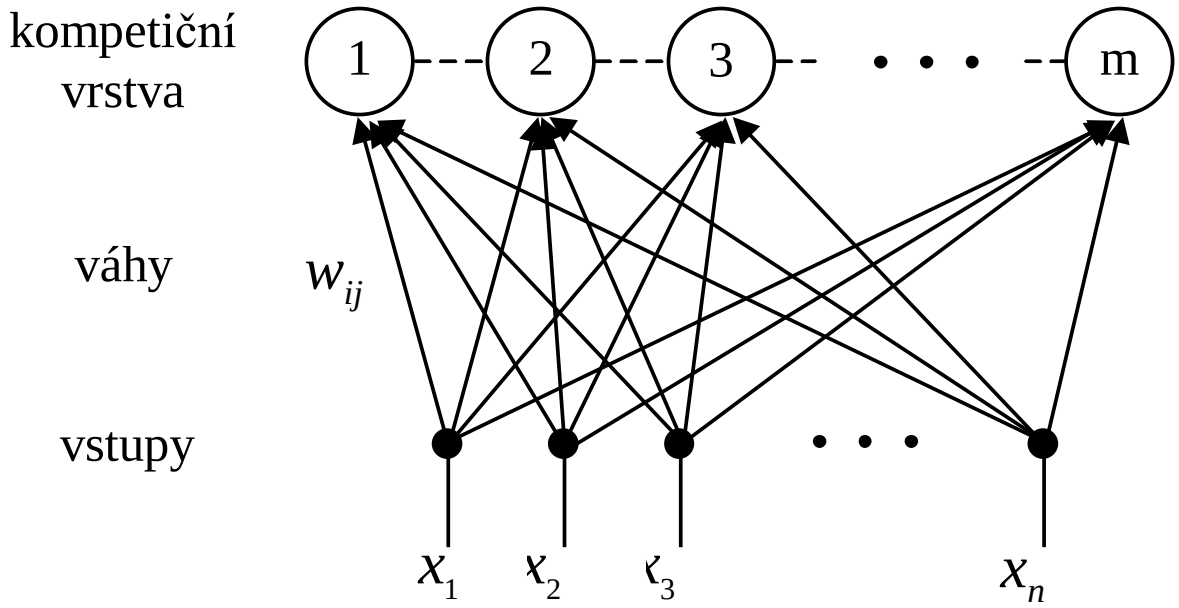
Vlastnosti:

- Provádí shlukovou analýzu vstupních dat.
- Snahou nalézt prostorovou reprezentaci složitých datových struktur.

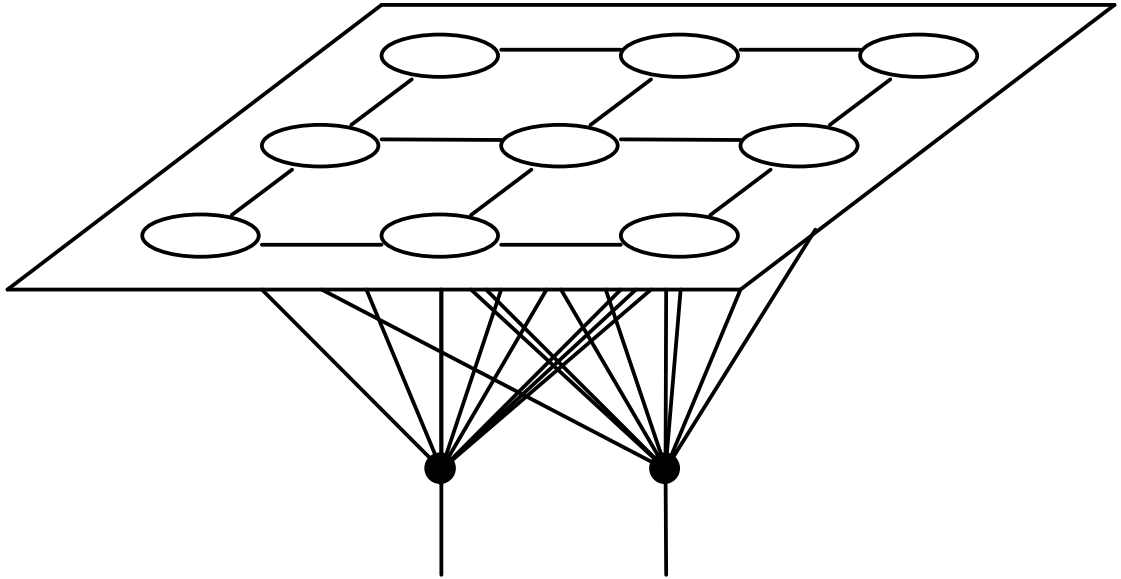
Topologie:

- Dvouvrstvá síť (vstupní vrstva a kompetiční vrstva) s úplným propojením mezi vrstvami.
- Výstupní neurony jsou navíc uspořádány do nějaké topologické struktury (dvourozměrná mřížka, jednorozměrná řada).

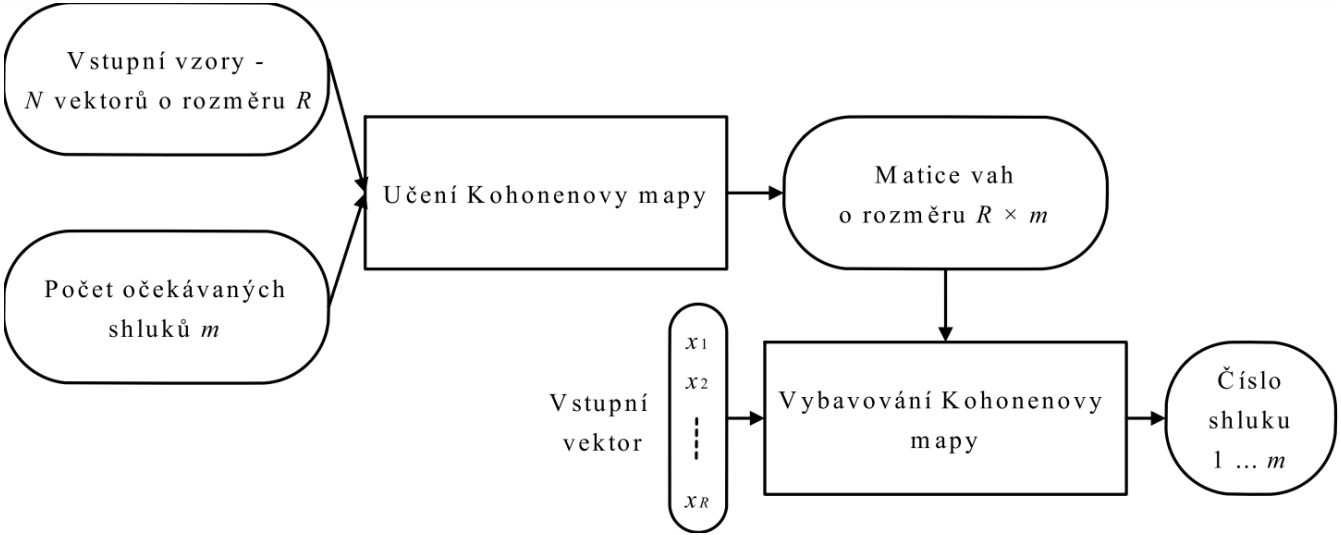
Topologie Kohonenovy mapy



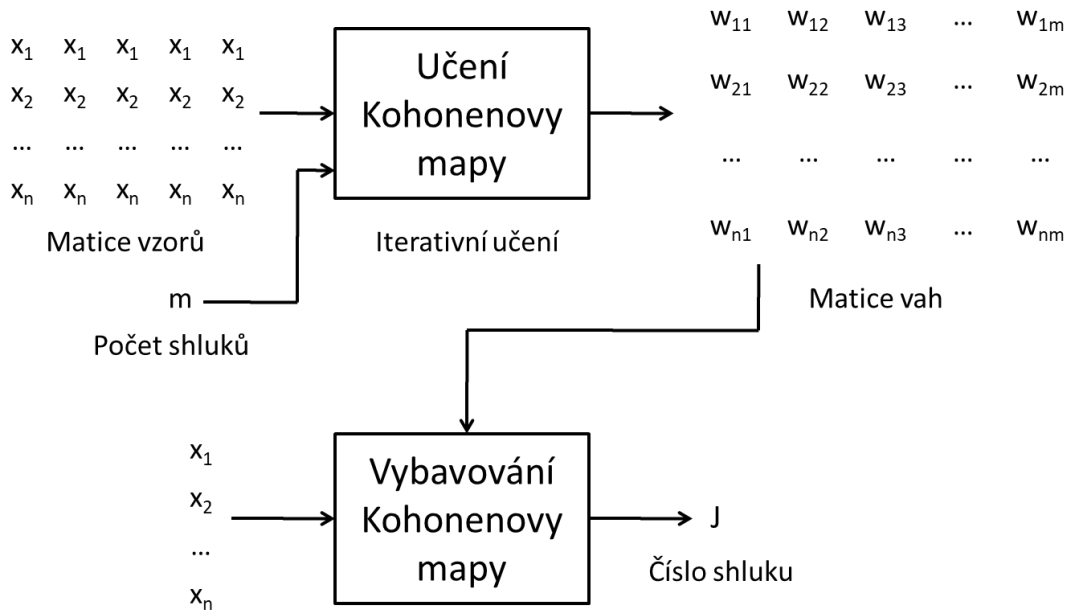
Topologie Kohonenovy mapy



Životní cyklus Kohonenovy mapy



Životní cyklus Kohonenovy mapy



Učení Kohonenovy mapy

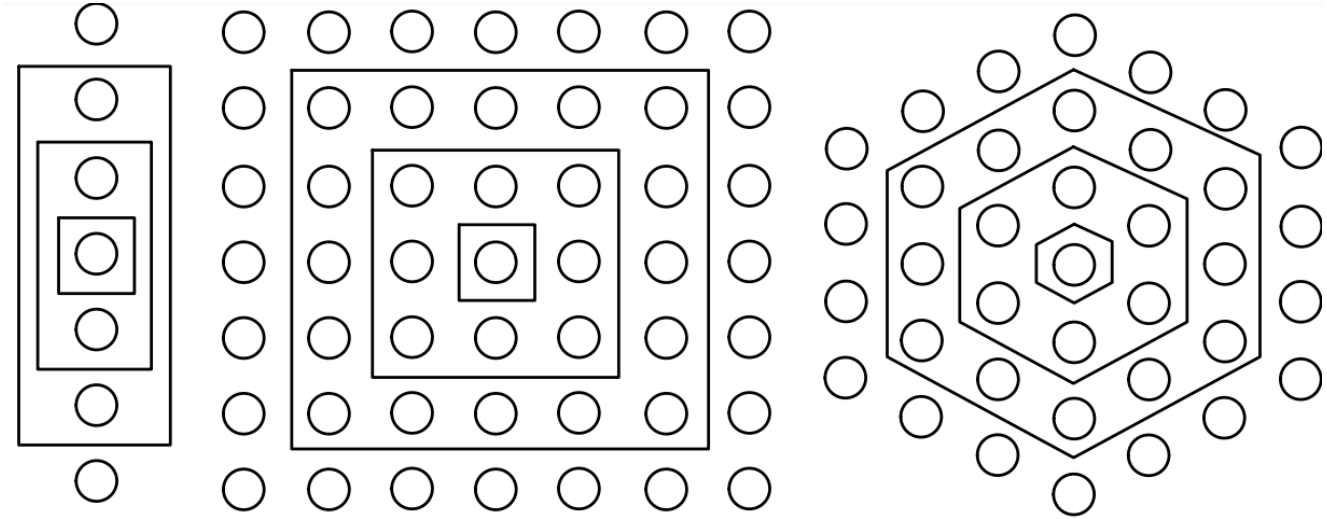
Vzhledem k tomu, že spolu neurony výstupní vrstvy „soutěží“, tak je třeba zvolit jaké kritérium definuje vítěze.

U Kohononovy mapy se jako toto kritérium volí vzdálenost vah neuronu od vstupních hodnot.

Nicméně by i okolní neurony měli být aktuálním výstupem ovlivněny, a proto se definuje okolí aktivního neuronu.

Pro učení je klíčová struktura Kohonenovy mapy spolu s okolím neuronu.

Mřížky a okolí neuronu



Jednorozměrná řada

Dvojrozměrná mřížka

Šesterečná mřížka

Učení Kohonenovy mapy

- Během učení se Kohonenově mapě postupně předkládají hodnoty vstupních dat a pro každý vstupní vektor vybere neuron J, který je prostorově nejblíže vstupnímu vektoru.
- Následně se upraví váhy tohoto neuronu a také všech neuronů v jeho okolí směrem ke vstupnímu vektoru, dle vztahu:

$$w_{ij}(k) = \begin{cases} w_{ij}(k-1) + \alpha\left[x_i(k) - w_{ij}(k-1)\right], & \text{pro } j \in \varrho(J), \\ w_{ij}(k-1), & \text{jinak}, \end{cases}$$

Učení Kohonenovy mapy

$$w_{ij}(k) = \begin{cases} w_{ij}(k-1) + \alpha\left[x_i(k) - w_{ij}(k-1)\right], & \text{pro } j \in \varrho(J), \\ w_{ij}(k-1), & \text{jinak}, \end{cases}$$

- Velikost okolí přitom není konstantní, ale během učení se snižuje.
- Počáteční hodnota okolí se volí tak, aby pokrývala asi polovinu všech neuronů v kompetiční vrstvě.
- Doporučuje volit počáteční koeficient rychlosti učení α blízký 1 a postupně ho během učení snižovat až na hodnotu blízkou nule.
- Smyslem vztahu a učení je, aby vítězný neuron, který nejlépe reprezentuje aktuální vstup, ještě více zlepšil svou relativní pozici vůči němu.

Učení Kohonenovy mapy

1. Inicializuj váhy neuronu malými náhodnými čísly, definuj rychlostní konstantu α, poloměr topologického sousedství R.
2. Dokud není splněno kritérium pro zastavení:
   - 2.1 Pro každý vstup $\mathbf{x} = [x_1, x_2, \cdots x_n]^T$, z trénovacích dat
     - 2.1.1 pro každé $j = 1, \ldots, m$ spočti:

       $$D(j) = \sum_{i=1}^{n} \left(w_{ij} - x_i \right)^2$$

     - 2.1.2 Najdi index J takový, že $D(J)$ je minimální.
     - 2.1.3 Aktualizuj váhy všech neuronů, které jsou v topologickém sousedství neuronu J vztahem:

       $$w_{ij}(k+1) = w_{ij}(k) + \alpha \left(x_i - w_{ij}(k) \right)$$

   - 2.2 Aktualizuj parametr rychlosti učení a zmenši poloměr topologického sousedství R.
   - 2.3 Otestuj podmínku ukončení.

Vybavování Kohonenovy mapy

- Jakmile je Kohonenova mapa správně adaptována, je možné ji využít ke shlukové analýze:
  - Předložíme-li síti vstupní vektor, soutěží jednotlivé neurony o to, který z nich je vstupu nejblíže.
  - Tento neuron se pak aktivuje (výstup roven jedné), zatímco ostatní neurony zůstávají pasivní (výstup roven nule). Každý neuron tak reprezentuje určitou množinu vstupních dat, které jsou mu blízké.
- Algoritmus: Pro vstup $\mathbf{x} = [x_1, x_2, \cdots x_n]^{\mathrm{T}}$,

  pro každé $j = 1, \ldots, m$ spočti:

  $$D(j) = \sum_{i=1}^{n} \left(w_{ij} - x_i \right)^2$$

  Najdi index J takový, že $D(J)$ je minimální.

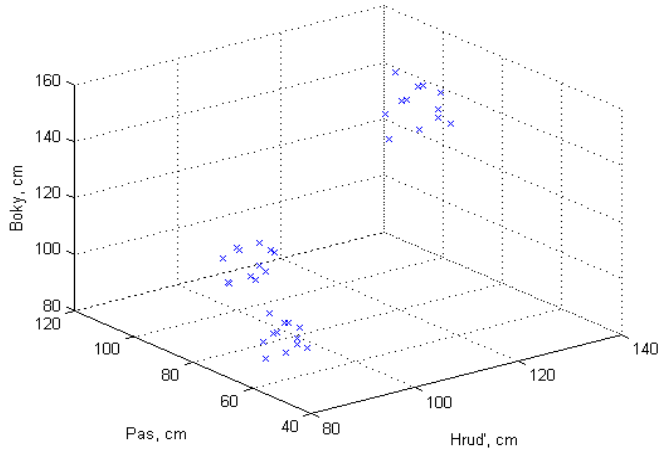
  Výstupem je číslo aktivovaného neuronu J

Příklad

Míry lidí (boky, pas, hruď)

Zastoupeny modelky, muži, ženy (stáří 55-65 let)

	1	2	3
1	90	135	135
2	62	93	88.5000
3	92	135	135

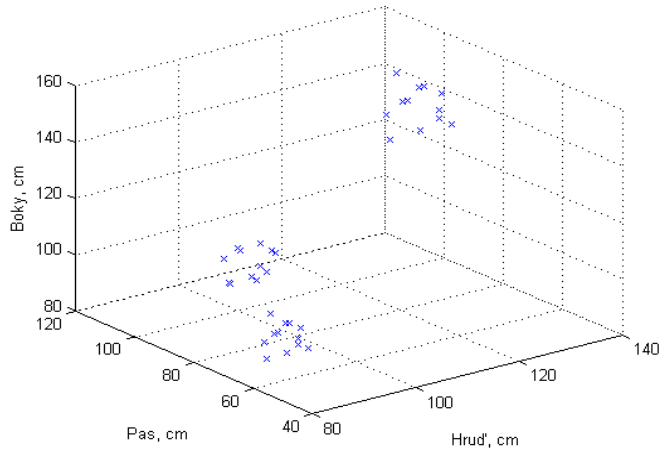


Příklad

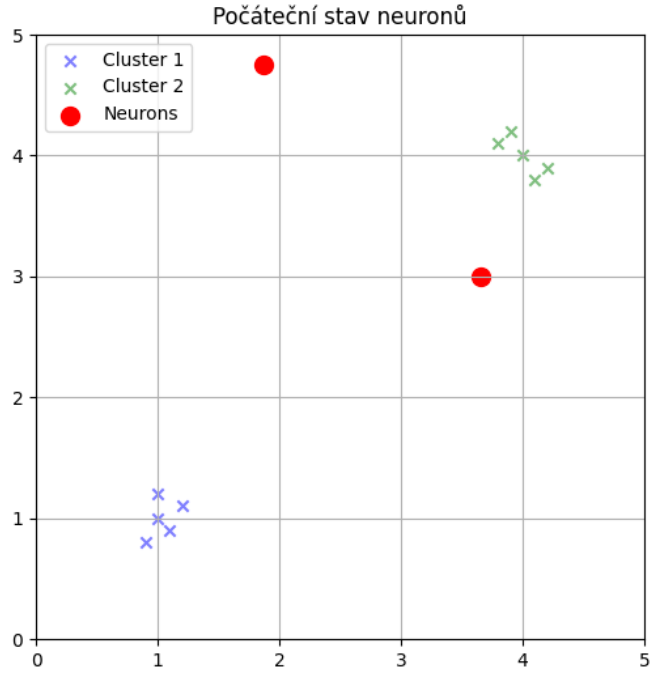
Ze zobrazení je zřejmé, že počet shluků m = 3.

Dále volme matici počátečních vah w. Rozměr 3 × 3 je dán jednak počtem shluků a jednak rozměrem vstupních dat.

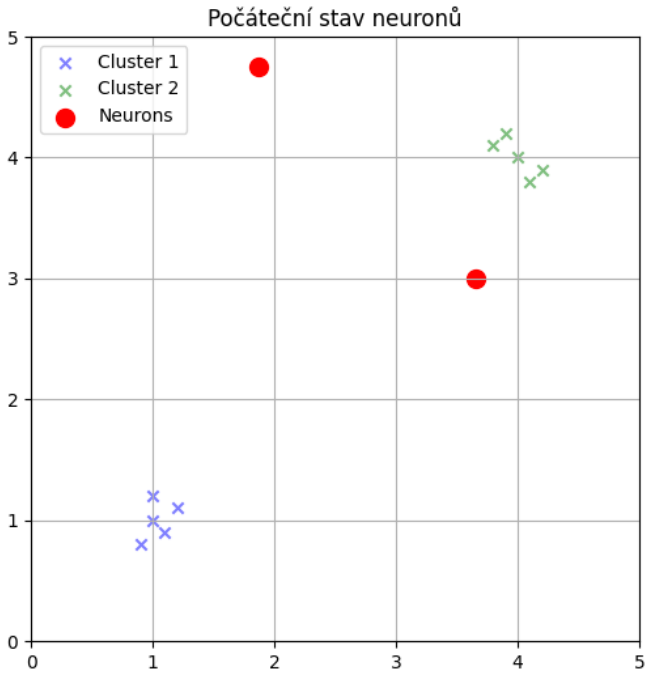
	1	2	3
1	90	135	135
2	62	93	88.5000
3	92	135	135

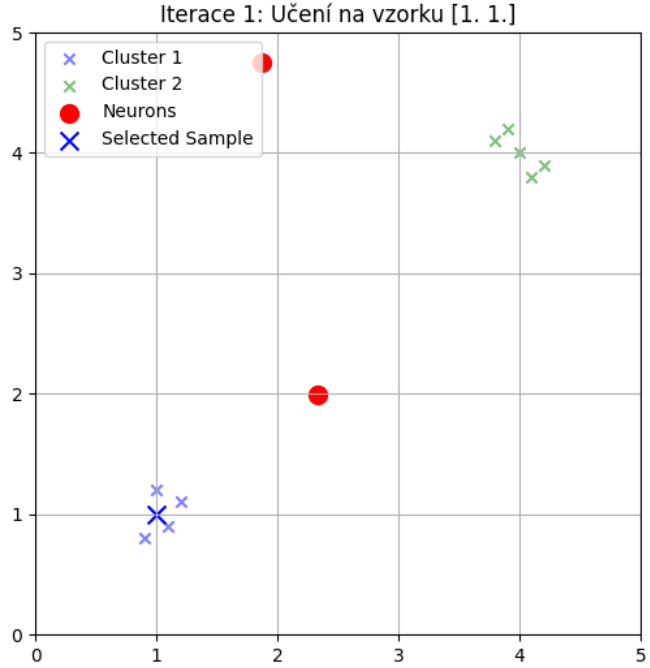


Příklad

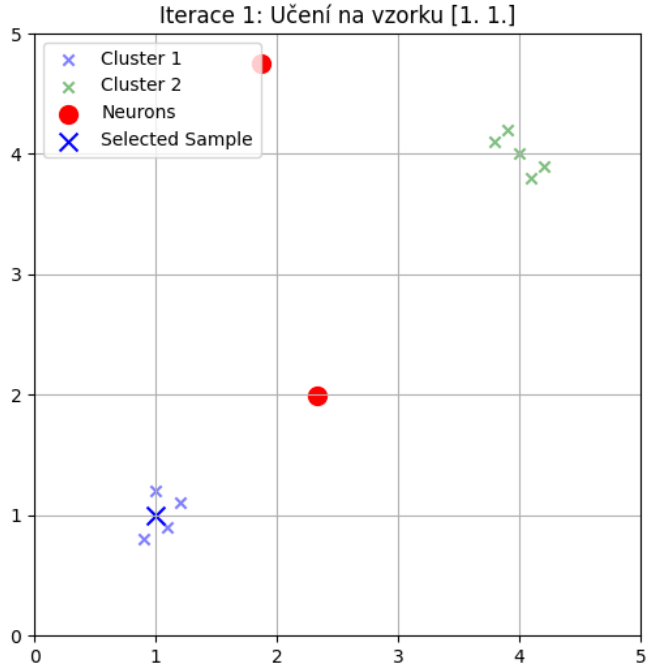


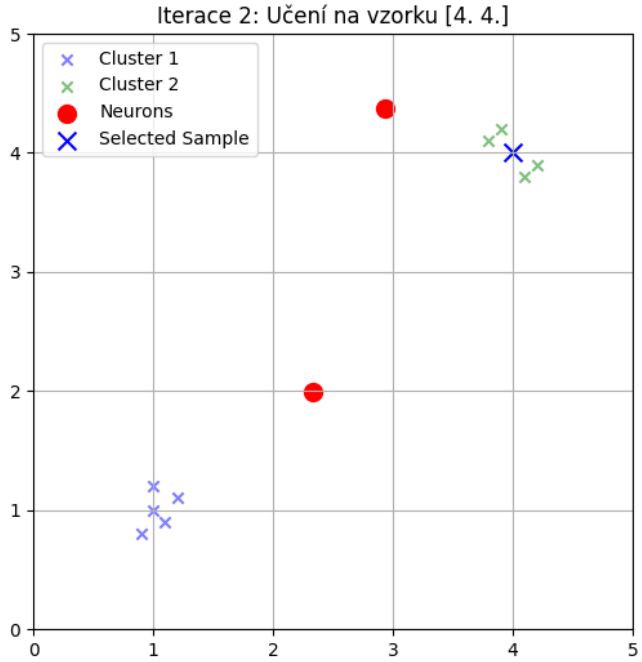
Příklad


Počáteční stav neuronů
Cluster 1
Cluster 2
Neurons
0
1
2
3
4
5

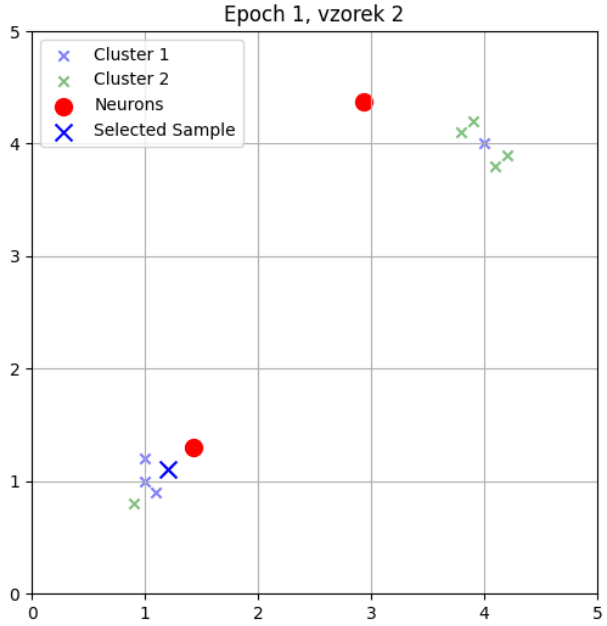

Iterace 1: Učení na vzorku [1. 1.]
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5

Příklad

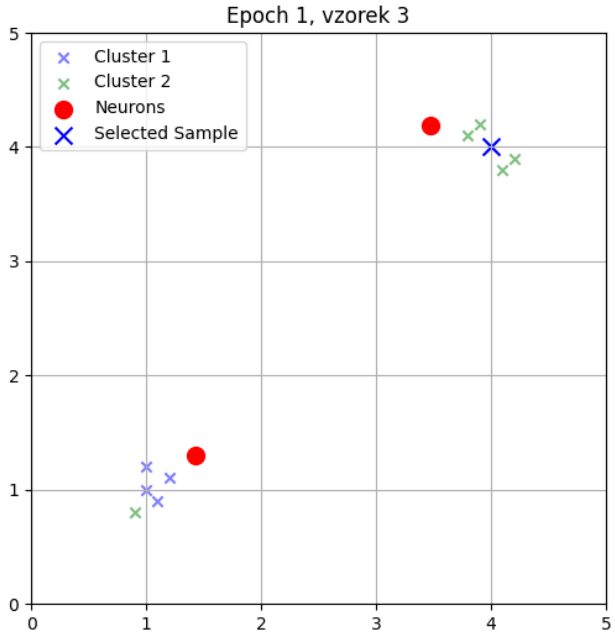

Iterace 1: Učení na vzorku [1. 1.]
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5


Iterace 2: Učení na vzorku [4. 4.]
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5

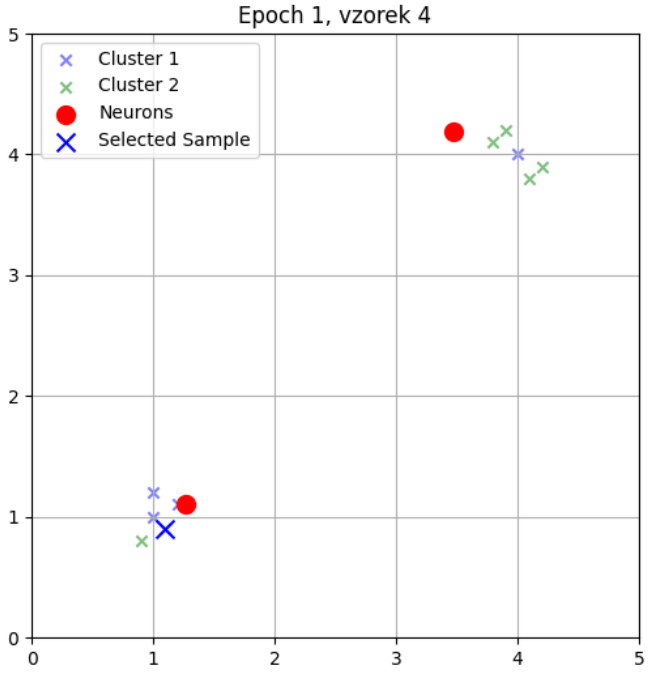
Příklad


Epoch 1, vzorek 2
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5

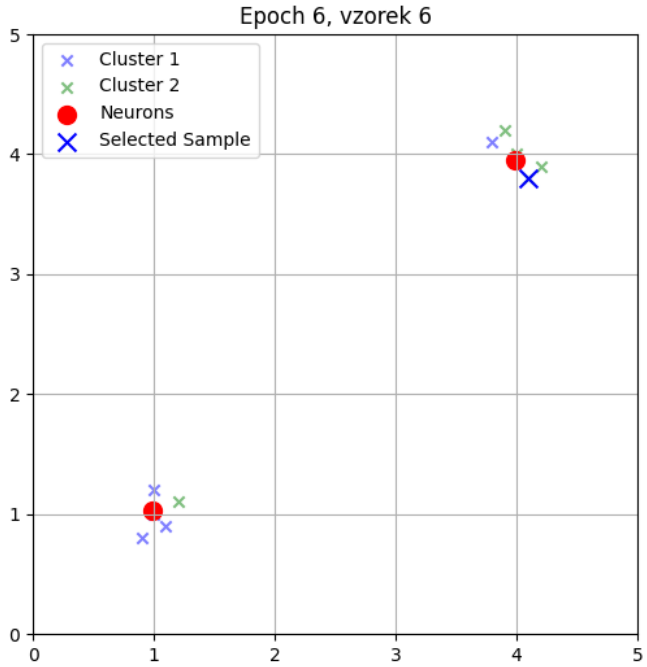
Příklad


Epoch 1, vzorek 3
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5

Příklad


Epoch 1, vzorek 4
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5

Príklad


Epoch 6, vzorek 6
Cluster 1
Cluster 2
Neurons
Selected Sample
0
1
2
3
4
5

Řešení příkladů

Diskrétní Hopfieldova síť

Kohonenova mapa

Cvičení

- Diskrétní Hopfieldova síť
  - Učení Hopfieldovy sítě
  - Vybavování Hopfieldovy sítě
- Kohonenova mapa
  - Učení Kohonenovy mapy
  - Vybavování Kohonenovy mapy

Učení Hopfieldovy sítě

- Aplikuje se Hebbův zákon učení
- Matice vah **w** je symetrická matice *n* x *n*, kde *n* je počet vstupů do sítě, pro každý prvek matice **w** platí:

$$w_{ij} = \begin{cases} \sum_{k=1}^{s} x_{ki} x_{kj} & pro \quad i \neq j \\ 0 & pro \quad i = j, \end{cases}$$

kde *S je* počet vzorů trénovací množiny

Vybavování Hopfieldovy sítě

1) $y_i(0) = x_i \qquad i = 1 \ldots n$

2) $y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$

3) Krok 2) se opakuje do ustálení

Příklad 1

- Dva vzory k zapamatování

$$\mathbf{x}_1 = [1 \quad 1 \quad 1 \quad -1]$$

$$\mathbf{x}_2 = [1 \quad 1 \quad -1 \quad -1]$$

$$w_{ij} = \begin{cases} \sum_{k=1}^{s} x_{ki} x_{kj} & pro \quad i \neq j \\ 0 & pro \quad i = j, \end{cases}$$

$$w_{12} = x_{11} \cdot x_{12} + x_{21} \cdot x_{22} = 2$$

$$w_{13} = x_{11} \cdot x_{13} + x_{21} \cdot x_{23} = 0$$

$$atd...$$

$$\mathbf{w} = \begin{bmatrix} 0 & x & x & x \\ x & 0 & x & x \\ x & x & 0 & x \\ x & x & x & 0 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

Příklad 1

- Vybavování – 1. iterace

$$\mathbf{test} = \begin{bmatrix} -1 & 1 & -1 & -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

$$y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$$

$$y_1 = f\left(0\cdot(-1) + 2\cdot 1 + 0\cdot(-1) + (-2)\cdot(-1)\right) = f(4) = 1$$

$$y_2 = f\left(2\cdot(-1) + 0\cdot 1 + 0\cdot(-1) + (-2)\cdot(-1)\right) = f(0) = 1$$

$$y_3 = f\left(0\cdot(-1) + 0\cdot 1 + 0\cdot(-1) + 0\cdot(-1)\right) = f(0) = 1$$

$$y_4 = f\left((-2)\cdot(-1) + (-2)\cdot 1 + 0\cdot(-1) + 0\cdot(-1)\right) = f(0) = 1$$

Příklad 1

- Vybavování – 2. iterace

$$y = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

$$y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$$

$$y_1 = f(0\cdot 1 + 2\cdot 1 + 0\cdot 1 + (-2)\cdot 1) = f(0) = 1$$

$$y_2 = f(2\cdot 1 + 0\cdot 1 + 0\cdot 1 + (-2)\cdot 1) = f(0) = 1$$

$$y_3 = f(0\cdot 1 + 0\cdot 1 + 0\cdot 1 + 0\cdot 1) = f(0) = 1$$

$$y_4 = f((-2)\cdot 1 + (-2)\cdot 1 + 0\cdot 1 + 0\cdot 1) = f(-4) = -1$$

Příklad 1

- Vybavování – 3. iterace

$$y = \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 \end{bmatrix}$$

$$y_j(k+1) = f\left(\sum_{i=1}^{n} w_{ij} y_i(k)\right) \qquad j = 1 \cdots n$$

$$y_1 = f\left(0\cdot 1 + 2\cdot 1 + 0\cdot 1 + (-2)\cdot(-1)\right) = f(4) = 1$$

$$y_2 = f\left(2\cdot 1 + 0\cdot 1 + 0\cdot 1 + (-2)\cdot(-1)\right) = f(4) = 1$$

$$y_3 = f\left(0\cdot 1 + 0\cdot 1 + 0\cdot 1 + 0\cdot(-1)\right) = f(0) = 1$$

$$y_4 = f\left((-2)\cdot 1 + (-2)\cdot 1 + 0\cdot 1 + 0\cdot(-1)\right) = f(-4) = -1$$

Učení Kohonenovy mapy

1. Inicializuj váhy neuronu malými náhodnými čísly, definuj rychlostní konstantu α, poloměr topologického sousedství R.
2. Dokud není splněno kritérium pro zastavení:
   - 2.1 Pro každý vstup $\mathbf{x} = [x_1, x_2, \cdots x_n]^T$, z trénovacích dat
     - 2.1.1 pro každé $j = 1, \ldots, m$ spočti:

       $$D(j) = \sum_{i=1}^{n} \left(w_{ij} - x_i \right)^2$$

     - 2.1.2 Najdi index J takový, že $D(J)$ je minimální.
     - 2.1.3 Aktualizuj váhy všech neuronů, které jsou v topologickém sousedství neuronu J vztahem:

       $$w_{ij}(k+1) = w_{ij}(k) + \alpha \left(x_i - w_{ij}(k) \right)$$

   - 2.2 Aktualizuj parametr rychlosti učení a zmenši poloměr topologického sousedství R.
   - 2.3 Otestuj podmínku ukončení.

Vybavování Kohonenovy mapy

- Jakmile je Kohonenova mapa správně adaptována, je možné ji využít ke shlukové analýze:
  - Předložíme-li síti vstupní vektor, soutěží jednotlivé neurony o to, který z nich je vstupu nejblíže.
  - Tento neuron se pak aktivuje (výstup roven jedné), zatímco ostatní neurony zůstávají pasivní (výstup roven nule). Každý neuron tak reprezentuje určitou množinu vstupních dat, které jsou mu blízké.
- Algoritmus: Pro vstup $\mathbf{x} = [x_1, x_2, \cdots x_n]^{\mathrm{T}}$,

  pro každé $j = 1, \ldots, m$ spočti:

  $$D(j) = \sum_{i=1}^{n} \left(w_{ij} - x_i \right)^2$$

  Najdi index J takový, že $D(J)$ je minimální.

  Výstupem je číslo aktivovaného neuronu J

Dopředná vícevrstvá umělá neuronová síť (ANN/FFNN)

Dopředná vícevrstvá umělá neuronová síť

- Je vhodná k řešení následujících problémů:
  - Univerzální aproximace
  - Kódování
  - Predikce
  - Rozpoznávání vzorů (klasifikace)
  - Rozhodování

Dopředná vícevrstvá umělá neuronová síť

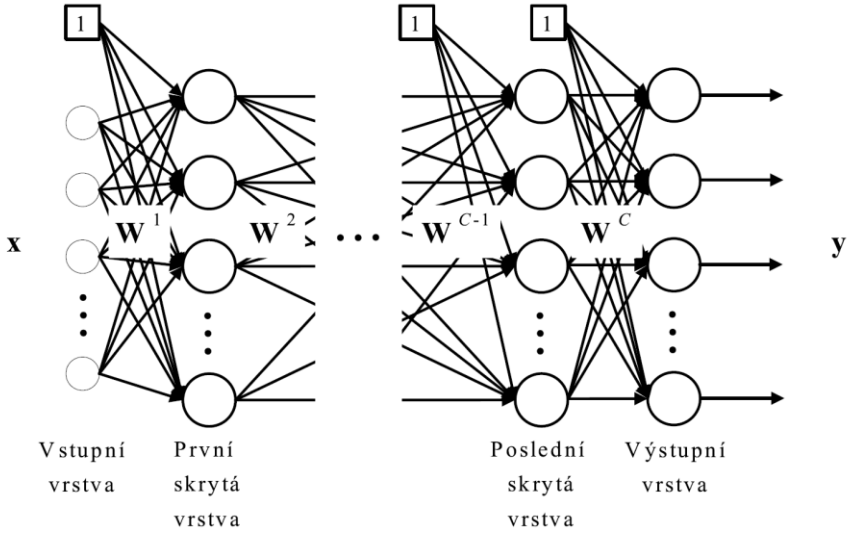
- Univerzální aproximace
  - Spojitá funkce *n* proměnných muže být vyjádřena pomocí konečného součtu funkcí jedné proměnné.
  - Libovolná spojitá funkce muže být s požadovanou přesností aproximovaná pomocí dopředné vícevrstvé umělé neuronové sítě s jednou skrytou vrstvou.
  - Postup tvorby FFNN jako univerzálního aproximátoru je experimentální a vychází ze systematického postupu, kde provádíme:
    - Hledání vhodné topologie sítě.
    - Optimalizace parametru rychlosti učení α.
    - Testování.

Dopředná vícevrstvá umělá neuronová síť

- Modelování a predikce
  - Každý systém (používá se i termín proces, případně soustava) je možno zobrazit jako blok, na který působí vstupní signály a případně poruchy a jehož chování je možno sledovat na základe výstupních veličin
  - Systém vykazuje 2 typy chování:
    - Statické – závislost mezi signály v ustáleném tvaru – soustava algebraických rovnic
    - Dynamické – popisuje chování systému v čase (v přechodovém stavu) – soustava diferenciálních/diferenčních rovnic
  - Pomocí NN lze takový systém modelovat následovně:
    - Statické chování – aproximace nelineální funkce
    - Dynamické chování – je závislá nejen ne aktuálních stavech, ale také na stavech předchozích → prevedení modelu do diskrétního tvaru a definice vstupních a výstupních proměnných → tvorba trénovací množiny dle předpokládaného modelu

Topologie FFNN

- Skládá se ze vstupní vrstvy, skrytých vrstev a výstupních vrstvev
- Vazby mezi neurony vedou pouze jedním směrem – dopředu.



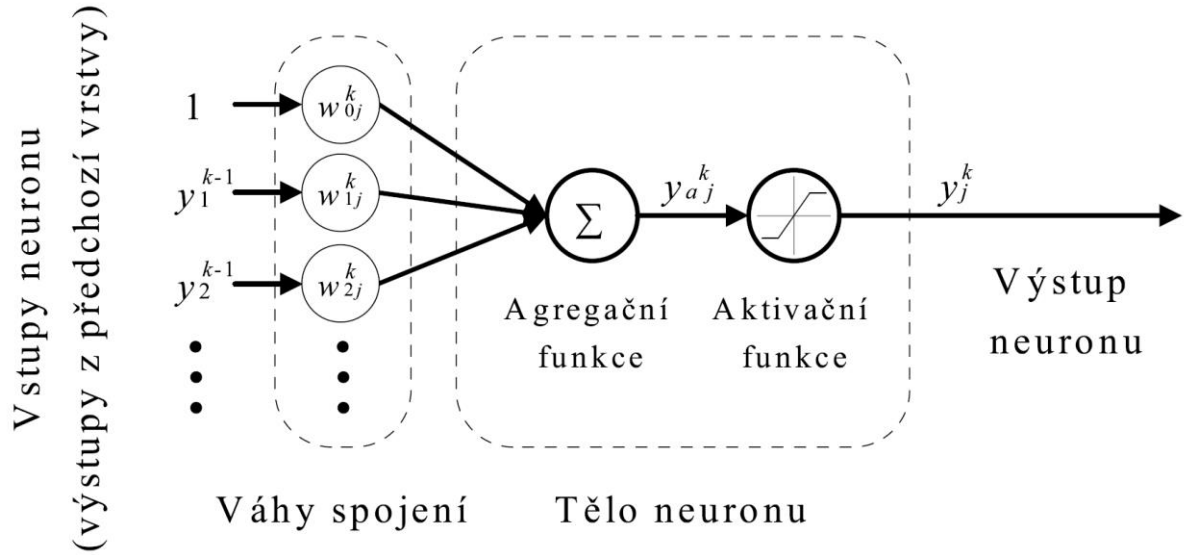
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_R \end{bmatrix} \qquad \mathbf{W}^k = \begin{bmatrix} w_{01}^k & w_{02}^k & \dots \\ w_{11}^k & w_{12}^k & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_Q \end{bmatrix}$$

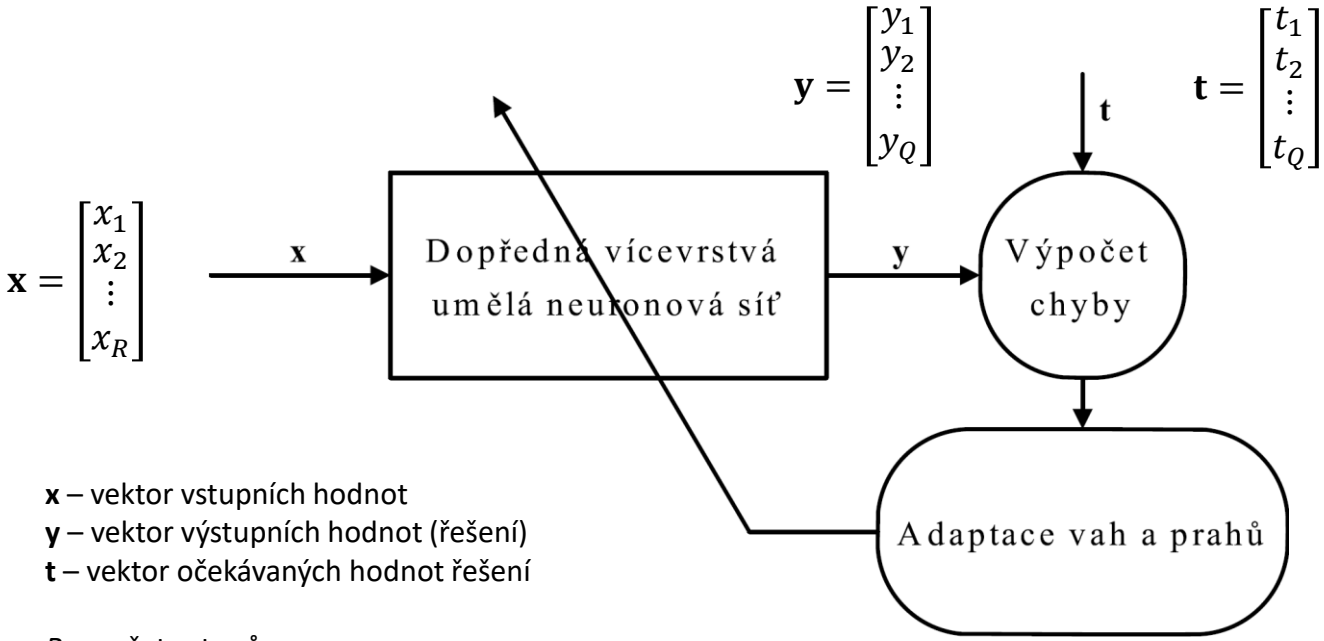
R – počet vstupů
Q – počet výstupů
C – počet skrytých vrstev
k – pořadové číslo vrstvy

Stavební blok FFNN

- FFNN je složena z formálních neuronů zapojených do vrstev.



Proces učení FFNN



x – vektor vstupních hodnot
y – vektor výstupních hodnot (řešení)
t – vektor očekávaných hodnot řešení

R – počet vstupů
Q – počet výstupů

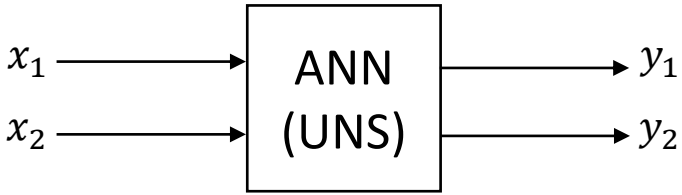
Chyby u FFNN

$$e = \sum_{k=1}^{Q} (t_k - y_k)^2$$

$$E = \frac{1}{N} \sum_{i=1}^{N} e_i$$

Proces učení FFNN

- ANN realizující součet a rozdíl dvou vstupů.



Chyby u FFNN

$$e = \sum_{k=1}^{Q} (t_k - y_k)^2$$

$$E = \frac{1}{N} \sum_{i=1}^{N} e_i$$

N – počet vzorů
Q – počet výstupů

Vzor	x_1	x_2	y_1	y_2	$t_1 = x_1 + x_2$	$t_2 = x_1 - x_2$	e
1.	2	5	6	-2	7	-3	2
2.	3	2	6	3	5	1	5
3.	7	5	11	2	12	3	2

$$E = \frac{1}{3}(2 + 5 + 2) = 3$$

Optimalizace FFNN

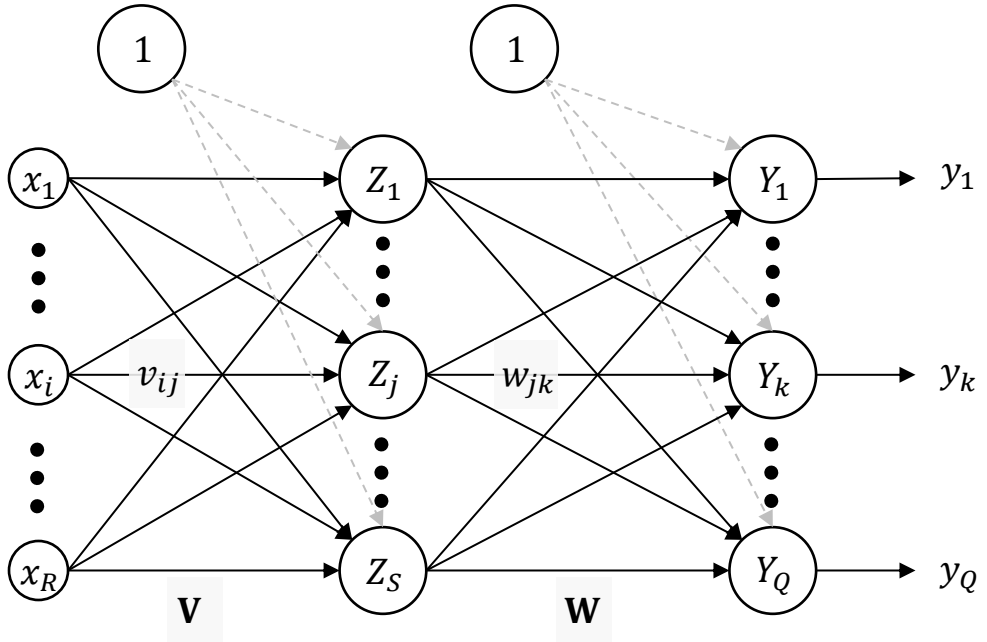
- Optimalizace topologie (počet vrstev, počet neuronů v každé vrstvě, volba aktivační funkce, …)
- Optimalizace vah a prahů (učení umělé neuronové sítě)

Učení umělé neuronové sítě

- Cílem je optimalizace vah a prahů pro zajištění požadované funkce UNS.
- Topologie UNS je definována.
- Je tedy třeba nastavit „správné“ hodnoty vah a prahů.
- Kvazi Newtonova metoda, Levenberův-Marquardtův algoritmus, …
- Algoritmus zpětného šíření chyby (Backpropagation Gradient Descent)

Algoritmus zpětného šíření chyby (BGD)

- Topologie FFNN je dána – 2 vrstvá FFNN (1 skrytá + 1 výstupní)



R – počet vstupů
Q – počet výstupů
S – počet neuronů ve skryté vrstvě

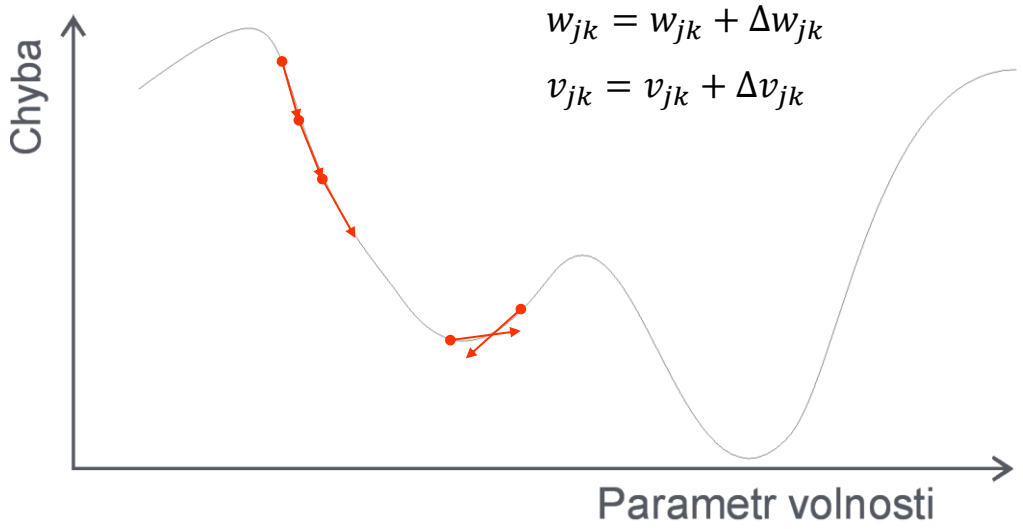
Hod. neuronů ve skryté Z_j
Hod. agregační funkce označeny indexem in
Aktivační funkce označena ϕ

Platí: $y_{in_k} = \sum_j w_{jk} z_j$

$$y_k = \phi(y_{in_k})$$

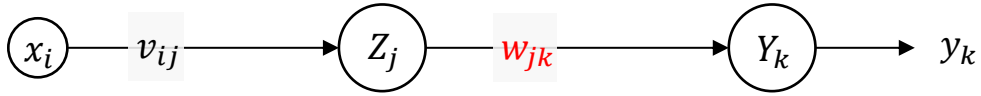
Algoritmus zpětného šíření chyby (BGD)

- Cílem učení je minimalizovat chybu $e = \sum_{k=1}^{Q}(t_k - y_k)^2$ změnou vah a prahů jednotlivých neuronů, tzn. že hledáme minimum $e = f(\boldsymbol{v}, \boldsymbol{w})$
- Cílem je najít vhodné přírůstky vah tak, aby bylo nalezeno minimum -> nalezení spádu funkce (gradientu) -> derivace chybové funkce



Algoritmus zpětného šíření chyby

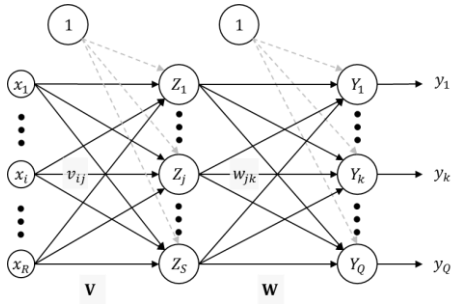
- Odvození Δw_{jk}



$$e = \sum_{k=1}^{Q} (t_k - y_k)^2$$

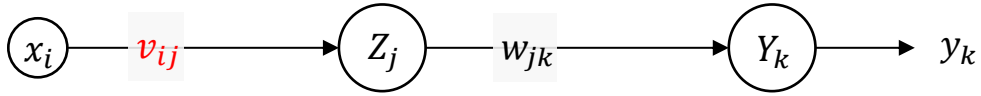
$$\frac{\partial e}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_{k=1}^{Q} (t_k - y_k)^2 = \frac{\partial}{\partial w_{jk}} \sum_{k=1}^{Q} \left(t_k - \phi\left(y_{in_k}\right)\right)^2 = -2(t_k - y_k) \frac{\partial}{\partial w_{jk}} \phi\left(y_{in_k}\right) =$$

$$= -2(t_k - y_k)\phi'\left(y_{in_k}\right) z_j = -\delta_k z_j$$



Algoritmus zpětného šíření chyby

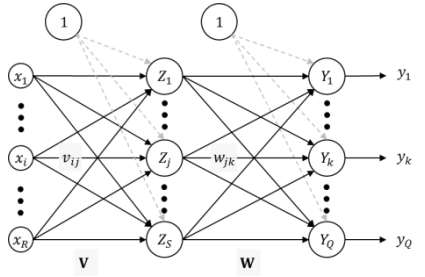
- Odvození Δv_{ij}



$$\frac{\partial e}{\partial v_{ij}} = \frac{\partial}{\partial v_{ij}} \sum_{k=1}^{Q} (t_k - y_k)^2 = -2 \sum_{k=1}^{Q} (t_k - y_k) \frac{\partial}{\partial v_{ij}} y_k = -2 \sum_{k=1}^{Q} (t_k - y_k) \phi'(y_{in_k}) \frac{\partial}{\partial v_{ij}} y_{in_k} =$$

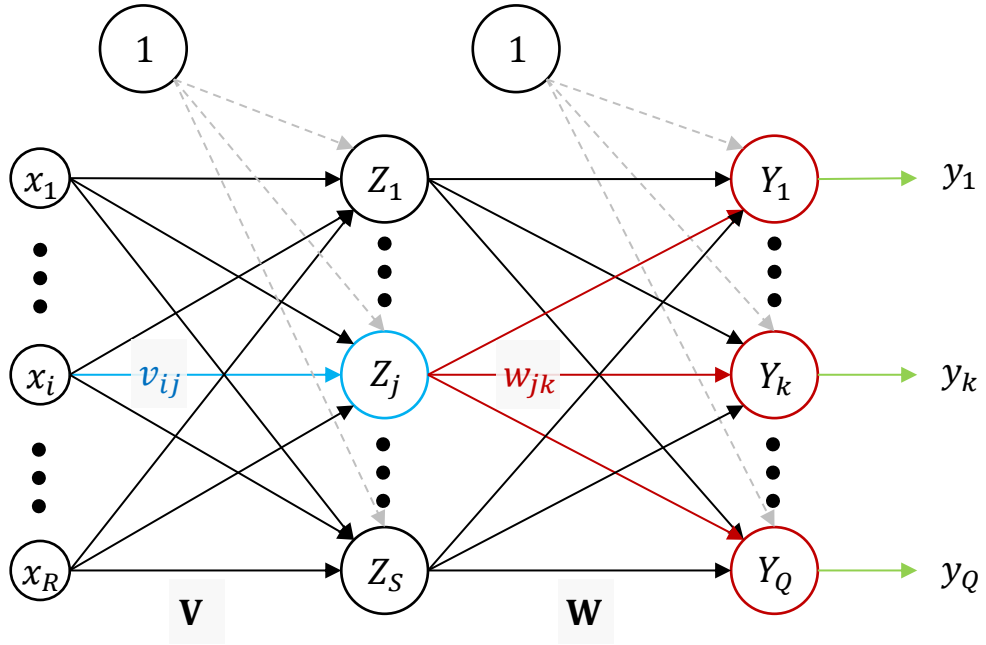
$$= -\sum_{k=1}^{Q} (t_k - y_k) \phi'(y_{in_k}) \frac{\partial}{\partial v_{ij}} y_{in_k} = -\sum_{k=1}^{Q} \delta_k \frac{\partial}{\partial v_{ij}} y_{in_k} = -\sum_{k=1}^{Q} \delta_k w_{jk} \frac{\partial}{\partial v_{ij}} z_j =$$

$$= -\sum_{k=1}^{Q} \delta_k w_{jk} \phi'\left(z_{in_j}\right) x_i = -\delta_j \, x_i$$



Algoritmus zpětného šíření chyby

- Graficky se chyba propaguje následujícím způsobem



Algoritmus zpětného šíření chyby

- Potom

$$\Delta w_{jk} = -\alpha \frac{\partial e}{\partial w_{jk}} = \boldsymbol{\alpha \delta_k z_j}$$

$$\Delta v_{ij} = -\alpha \frac{\partial e}{\partial v_{ij}} = \boldsymbol{\alpha \delta_j x_i}$$

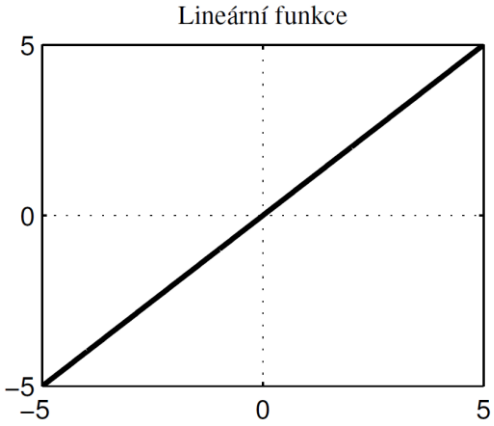
- Z principu algoritmu je třeba, aby všechny použité aktivační funkce byly derivovatelné (hladké, často lineární a sigmoidální)
- α se volí na intervalu (0; 1) metodou pokus-omyl

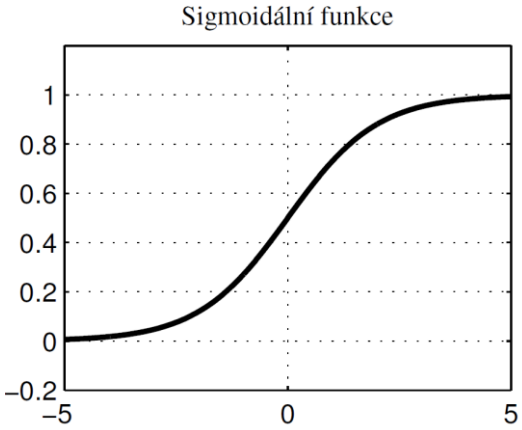
$$\varphi(x) = \frac{1}{1+e^{-\sigma x}}$$

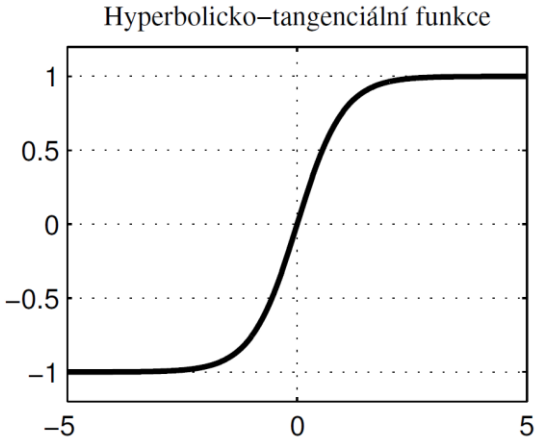
$$\varphi'(x) = \sigma\varphi(x)\left(1-\varphi(x)\right)$$

Vhodné aktivační funkce

- Vzhledem k aplikacím NN v nespojitém prostředí (digitální svět) jsou běžně užívány numerické derivace (diferencovatelnost) a další metody přibližně definující potřebné hodnoty



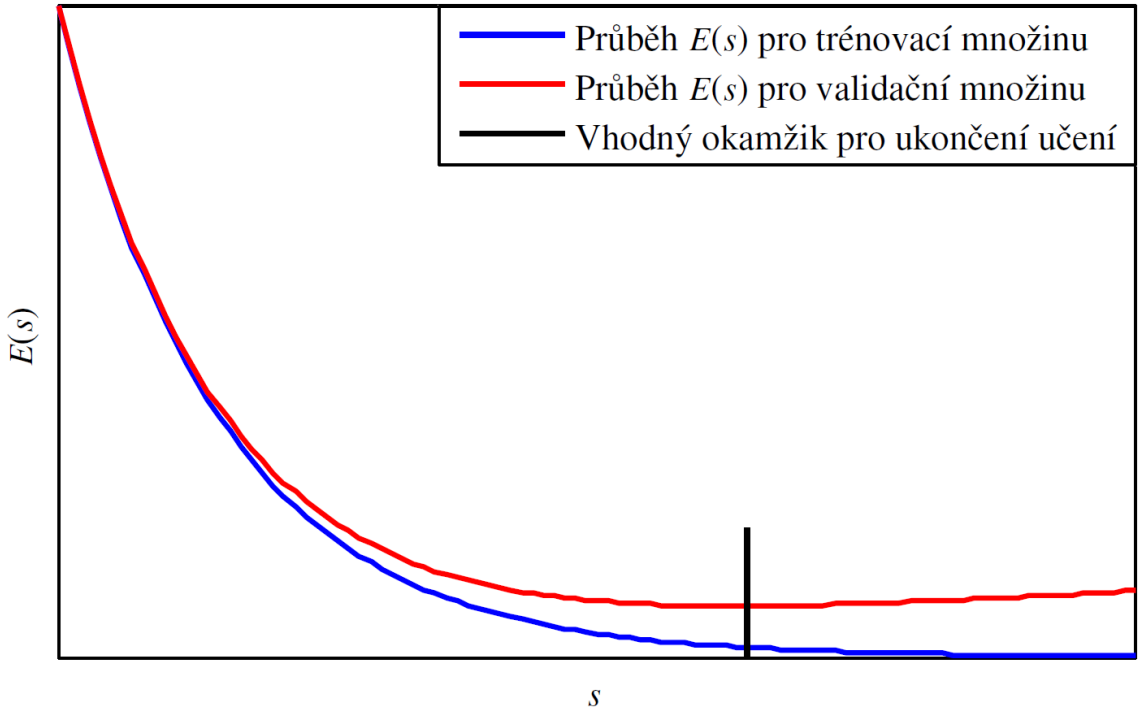




Celkový postup algoritmu učení

- Rozdělení dat na testovací, trénovací a validační množinu
- Nastavení vah a prahů sítě, nastavení koeficientu učení α
- Pro každou dvojici: vzor - očekávaný výstup,
  - Spočítání odezvy sítě
  - Spočítání δ_k pro všechny výstupní neurony
  - Spočítání δ_j pro neurony skryté vrstvy
  - Aktualizace vah a prahů
  - Test ukončení algoritmu
    - Maximální počet epoch
    - Změna hodnoty účelové funkce dále neklesá
    - Dosaženo kapacity sítě – výkon na validační množině se zhoršuje → přetrénování

Podmínka ukončení

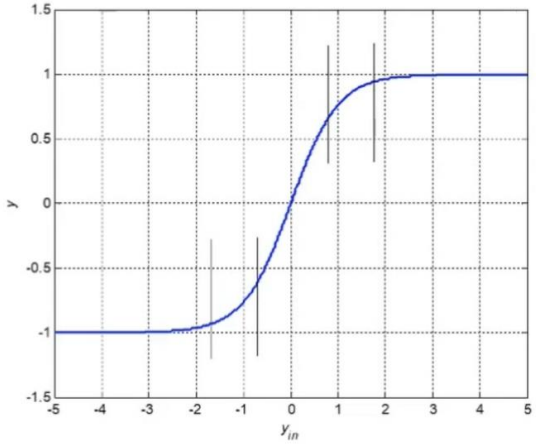


Varianty učení ANN

- Online
  - Po každém vzoru dojde k aktualizaci hodnot vah a prahů.
- Offline
  - Změny vah a prahů se sčítají v dočasné proměnné a jejich aktualizace se provede až na konci epochy trénování (po celé epoše).
- Dávkové (batch)
  - Kombinuje předchozí varianty, kde se aktualizace provádí po dávkách vzorků (většinou po 64, 32, 16, 8, 4 vzorech).
  - Kombinuje výhody a nevýhody obou variant (kompromis rychlosti a paměťové náročnosti)

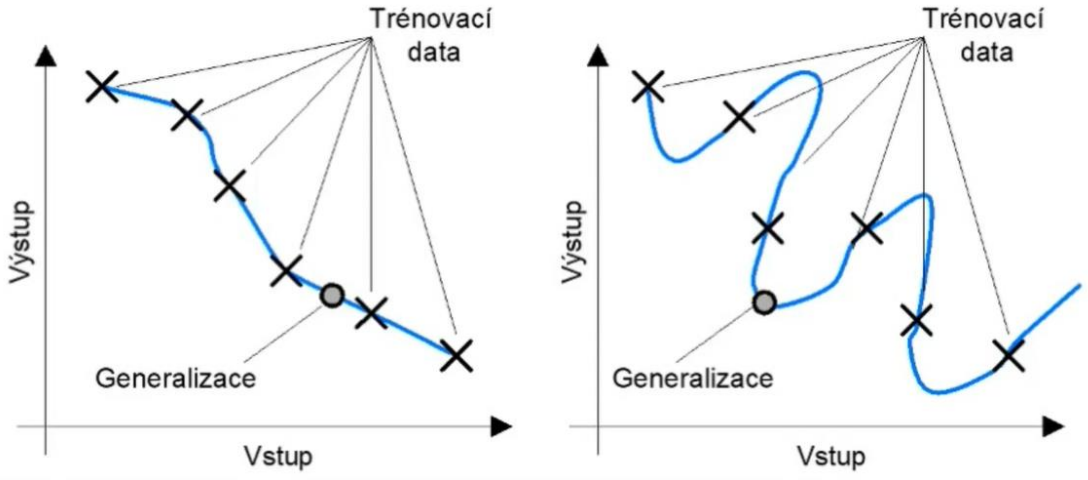
Heuristiky vylepšující BGD

- Výběr varianty trénování (online, offline, batch)
- Vhodný výběr dat trénovací množiny
  - Data způsobující velkou chybu
  - Data rozdílná od předchozích
  - Data normálně rozložená přes celou množinu
- Vhodná volba aktivační funkce
- Vhodná inicializace vah a prahů neuronů
  - Velké hodnoty → malé gradienty → pomalá progrese
  - Malé hodnoty → možnost dosažení sedlového bodu chybové funkce
  - Snaha volit inicializační hodnoty způsobující největší změny (nelineární oblasti)
- Normalizace dat



Heuristiky vylepšující BGD

- Vhodný výběr dat trénovací množiny pokrývající daný problém kvůli generalizaci



Optimalizace FFNN

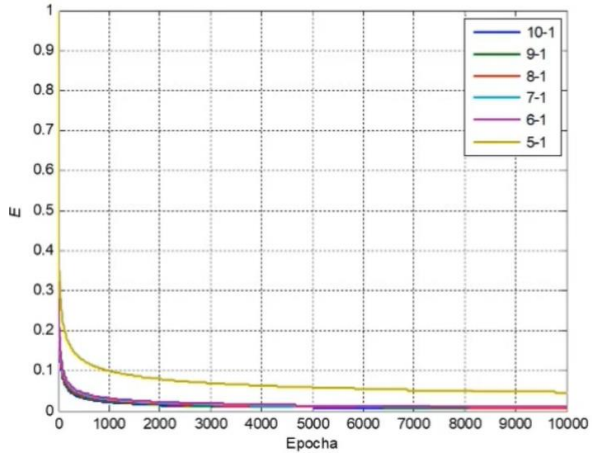
- Optimalizace topologie (počet vrstev, počet neuronů v každé vrstvě, volba aktivační funkce, …)
- Optimalizace vah a prahů (učení umělé neuronové sítě)

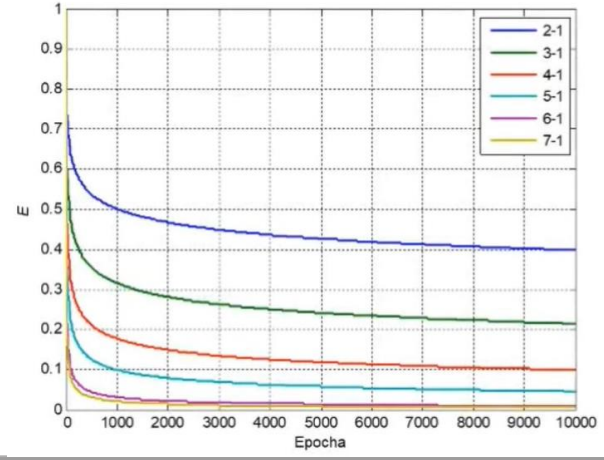
Optimalizace topologie FFNN

- Neexistuje analytický postup
- Počet vstupů a výstupů je dán problémem
- Počet skrytých vrstev

Optimalizace topologie FFNN

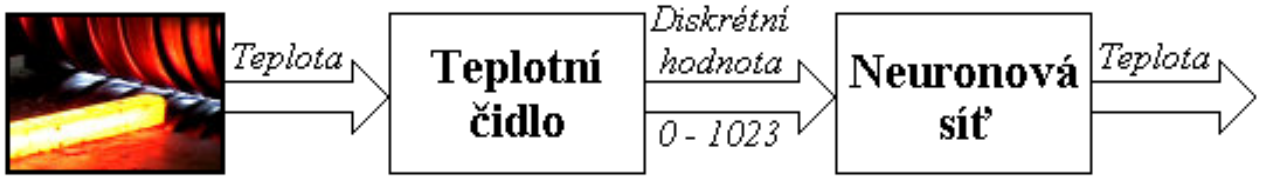
- Počet neuronů ve skrytých vrstvách
  - Různá doporučení v literatuře, ze zkušeností u 1 vrstvé ANN: $p = \sqrt{RQ}$
  - Metoda shora dolů
  - Metoda zdola nahoru





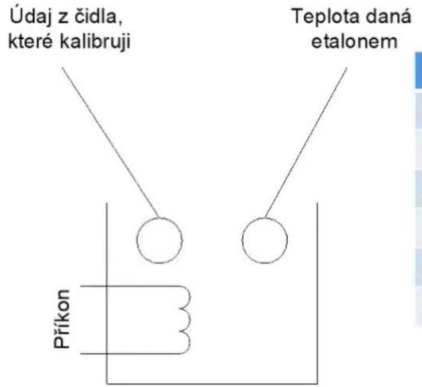
Příklad

- Teplotní čidlo, které měří teplotu v rozsahu 0 – 500 °C
- Výstupem čidla je diskrétní číslo v rozsahu 0 – 1023
- Pomocí kalibrace chceme navrhnou ANN, která výstup čidla bude transformovat (kódovat) na teplotu měřeného objektu

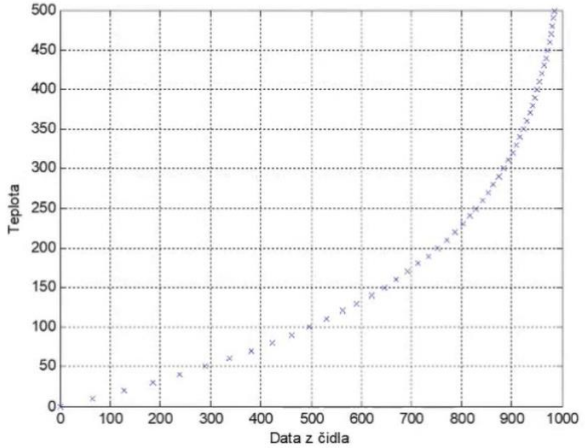


Příklad

- Zisk trenovací, testovací a validační množiny
  - Experiment (měření) s etalonem

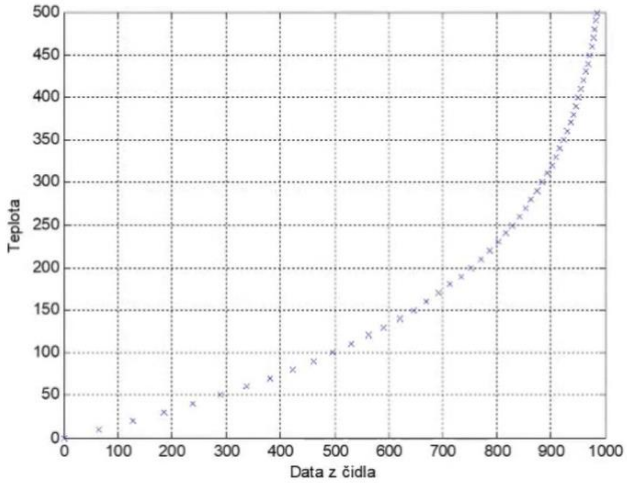


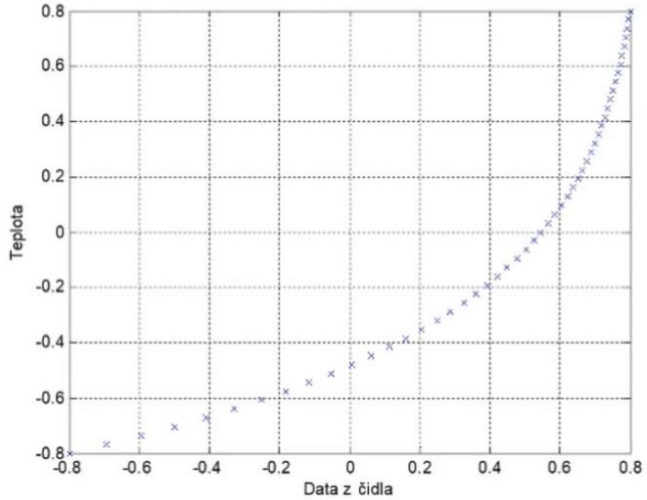
Údaj z čidla	Teplota z etalonu
0	0°C
65	10°C
127	20°C
185	30°C
239	40°C
...	...



Příklad

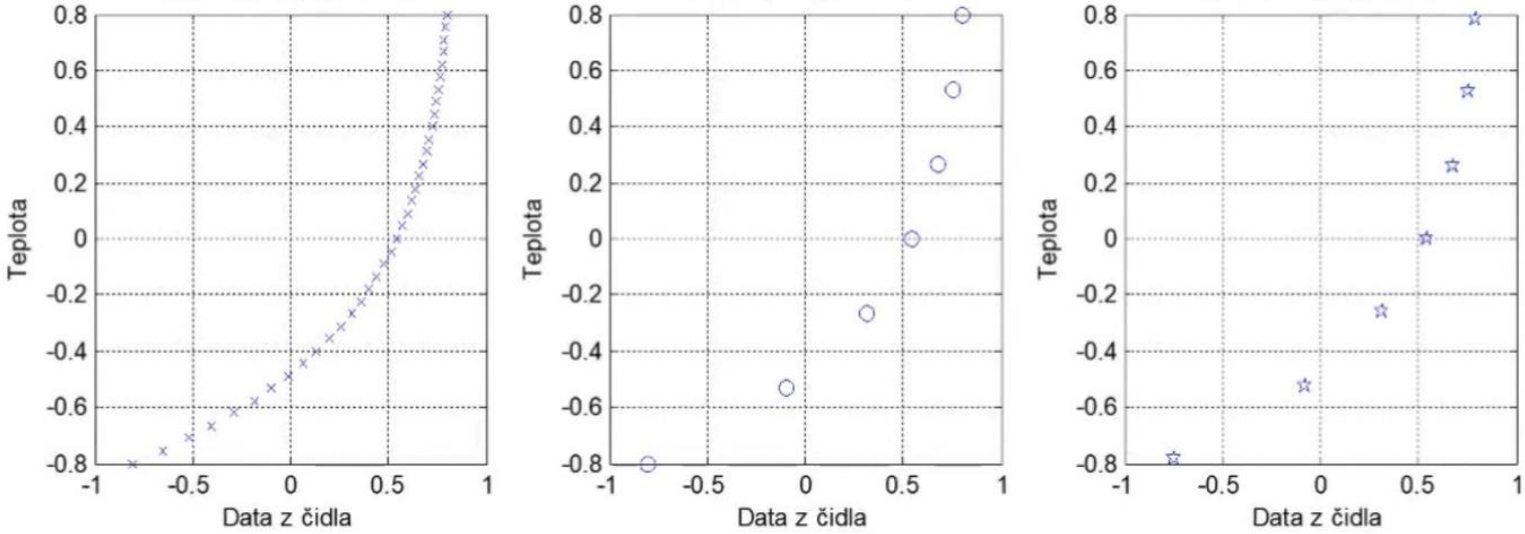
- Transformace (normalizace) dat





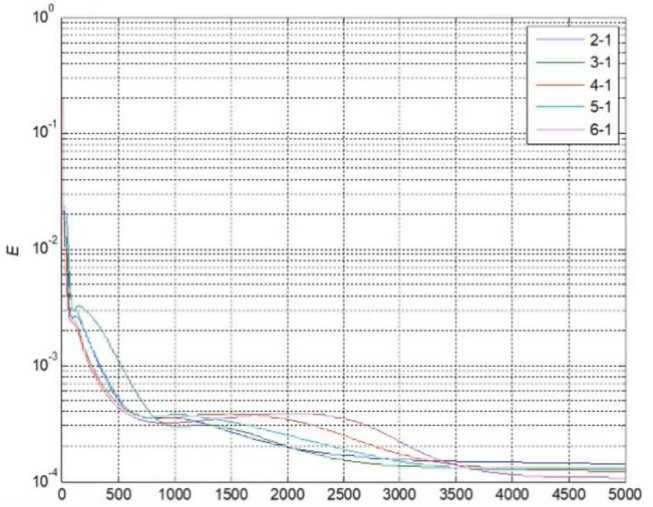
Příklad

- Rozdělení dat - zisk trénovací, testovací a validační množiny



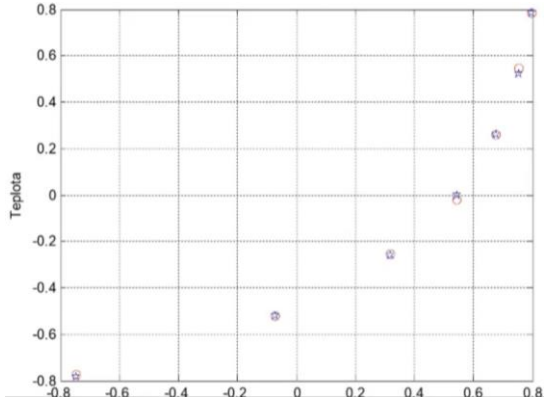
Příklad

- Jedná se spojitý průběh → uvažujme 1 skrytou vrstvu
- Trénování pro různé topologie



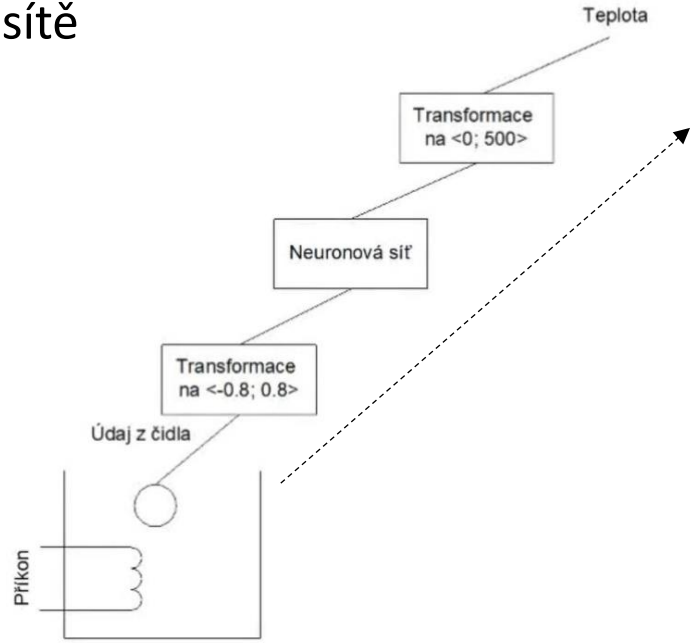
→ Více než 2 neurony ve skryté vrstvě nevedou k razantnímu zlepšení

Testování:



Příklad

- Použití sítě



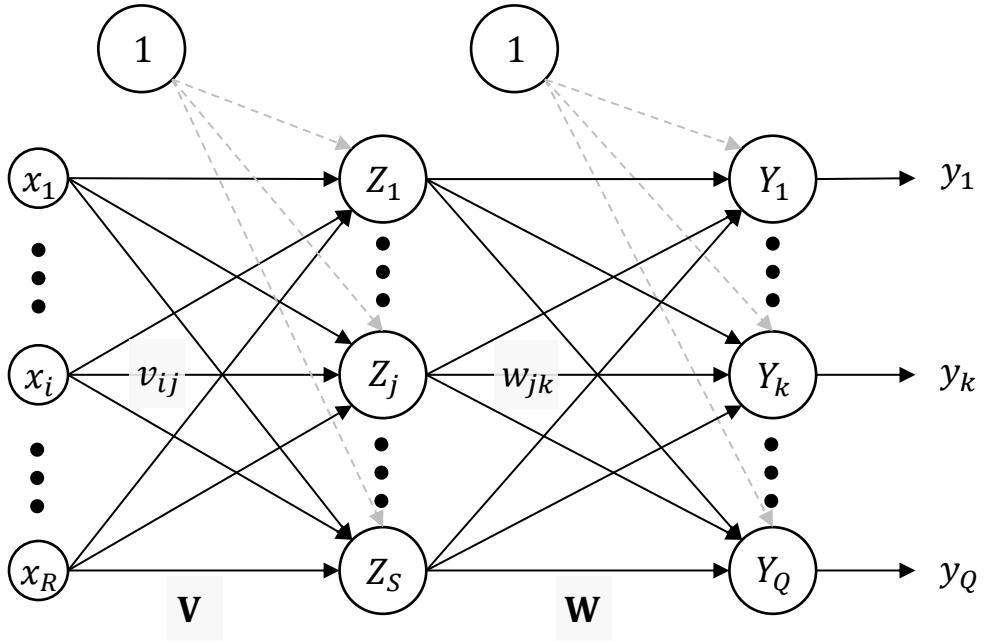
Metody učení neuronových sítí

Učení umělé neuronové sítě

- Cílem je optimalizace vah a prahů pro zajištění požadované funkce UNS.
- Topologie UNS je definována.
- Je tedy třeba nastavit „správné“ hodnoty vah a prahů.
- Kvazi Newtonova metoda, Levenberův-Marquardtův algoritmus, …
- Algoritmus zpětného šíření chyby (Backpropagation Gradient Descent)

Algoritmus zpětného šíření chyby (BGD)

- Topologie FFNN je dána – 2 vrstvá FFNN (1 skrytá + 1 výstupní)



R – počet vstupů
Q – počet výstupů
S – počet neuronů ve skryté vrstvě

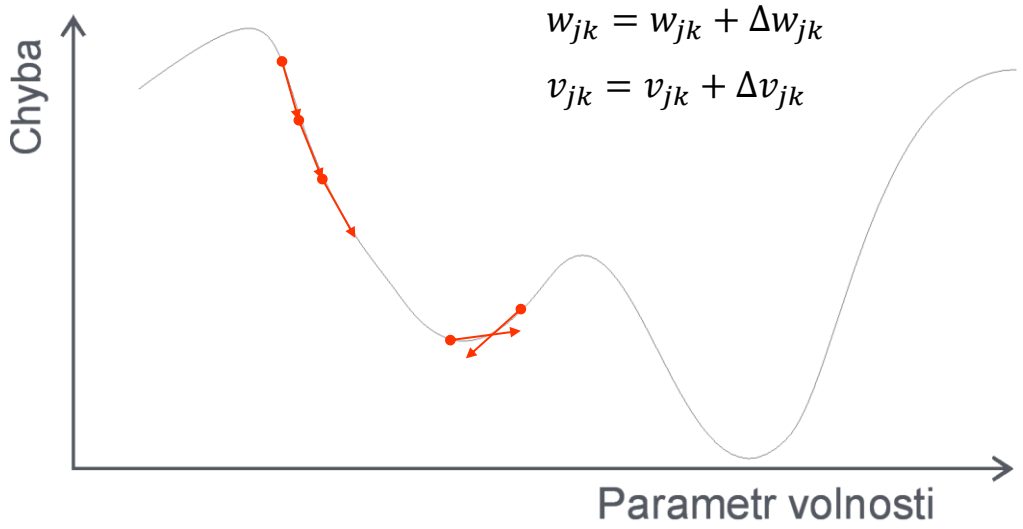
Hod. neuronů ve skryté Z_j
Hod. agregační funkce označeny indexem in
Aktivační funkce označena ϕ

Platí: $y_{in_k} = \sum_j w_{jk} z_j$

$$y_k = \phi(y_{in_k})$$

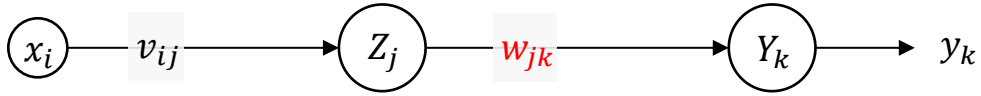
Algoritmus zpětného šíření chyby (BGD)

- Cílem učení je minimalizovat chybu $e = \sum_{k=1}^{Q}(t_k - y_k)^2$ změnou vah a prahů jednotlivých neuronů, tzn. že hledáme minimum $e = f(\boldsymbol{v}, \boldsymbol{w})$
- Cílem je najít vhodné přírůstky vah tak, aby bylo nalezeno minimum -> nalezení spádu funkce (gradientu) -> derivace chybové funkce



Algoritmus zpětného šíření chyby

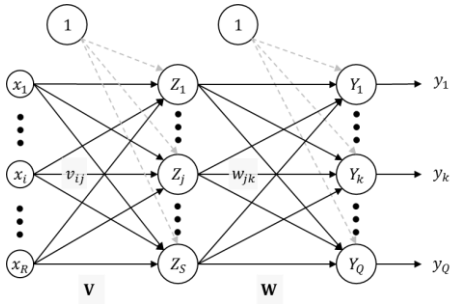
- Odvození Δw_{jk}



$$e = \sum_{k=1}^{Q} (t_k - y_k)^2$$

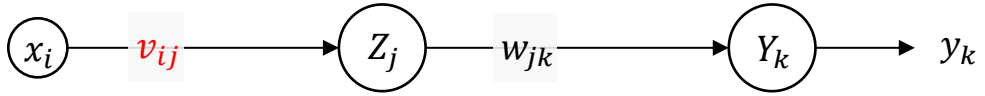
$$\frac{\partial e}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_{k=1}^{Q} (t_k - y_k)^2 = \frac{\partial}{\partial w_{jk}} \sum_{k=1}^{Q} \left(t_k - \phi\left(y_{in_k}\right)\right)^2 = -2(t_k - y_k) \frac{\partial}{\partial w_{jk}} \phi\left(y_{in_k}\right) =$$

$$= -2(t_k - y_k)\phi'\left(y_{in_k}\right) z_j = -\delta_k z_j$$



Algoritmus zpětného šíření chyby

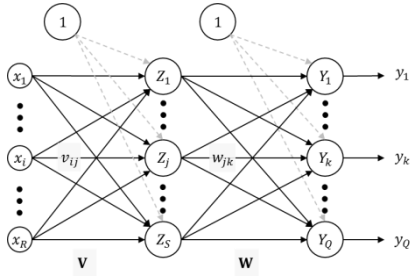
- Odvození Δv_{ij}



$$\frac{\partial e}{\partial v_{ij}} = \frac{\partial}{\partial v_{ij}} \sum_{k=1}^{Q} (t_k - y_k)^2 = -2 \sum_{k=1}^{Q} (t_k - y_k) \frac{\partial}{\partial v_{ij}} y_k = -2 \sum_{k=1}^{Q} (t_k - y_k) \phi'(y_{in_k}) \frac{\partial}{\partial v_{ij}} y_{in_k} =$$

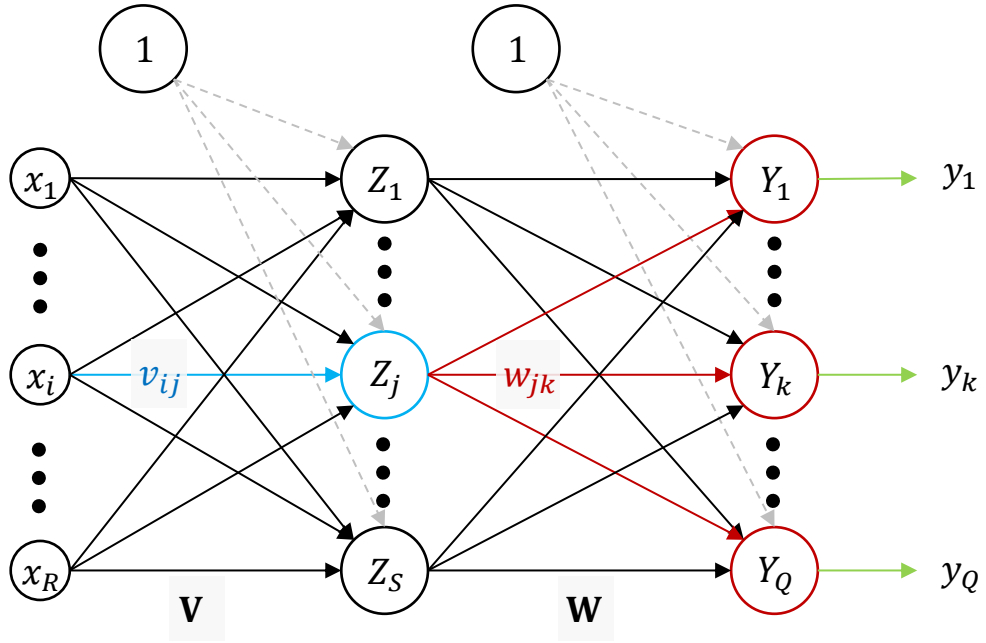
$$= - \sum_{k=1}^{Q} (t_k - y_k) \phi'(y_{in_k}) \frac{\partial}{\partial v_{ij}} y_{in_k} = - \sum_{k=1}^{Q} \delta_k \frac{\partial}{\partial v_{ij}} y_{in_k} = - \sum_{k=1}^{Q} \delta_k w_{jk} \frac{\partial}{\partial v_{ij}} z_j =$$

$$= - \sum_{k=1}^{Q} \delta_k w_{jk} \phi' \left(z_{in_j} \right) x_i = -\delta_j \, x_i$$



Algoritmus zpětného šíření chyby

- Graficky se chyba propaguje následujícím způsobem



Algoritmus zpětného šíření chyby

- Potom

$$\Delta w_{jk} = -\alpha \frac{\partial e}{\partial w_{jk}} = \boldsymbol{\alpha \delta_k z_j}$$

$$\Delta v_{ij} = -\alpha \frac{\partial e}{\partial v_{ij}} = \boldsymbol{\alpha \delta_j x_i}$$

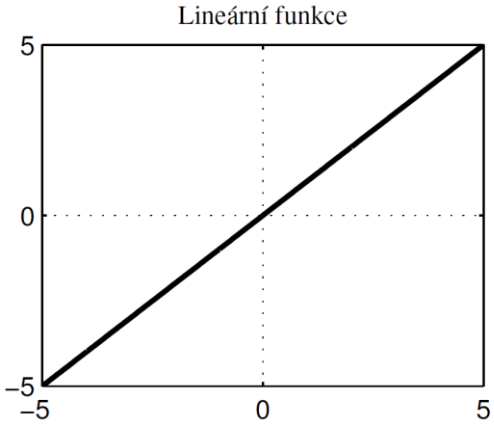
- Z principu algoritmu je třeba, aby všechny použité aktivační funkce byly derivovatelné (hladké, často lineární a sigmoidální)
- α se volí na intervalu (0; 1) metodou pokus-omyl

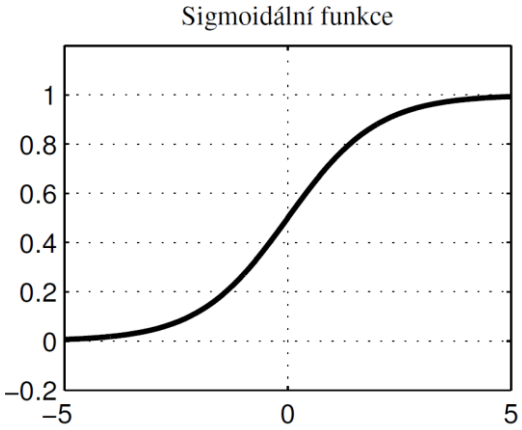
$$\varphi(x) = \frac{1}{1+e^{-\sigma x}}$$

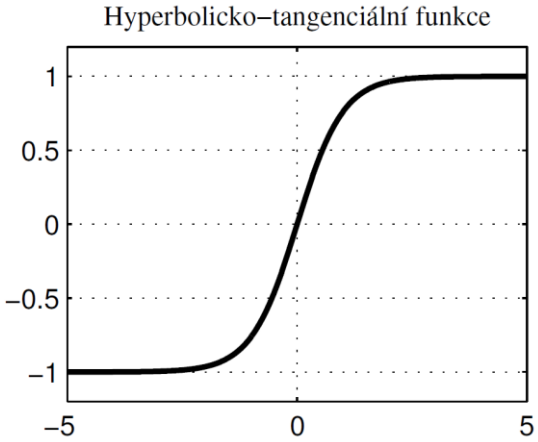
$$\varphi'(x) = \sigma\varphi(x)\left(1-\varphi(x)\right)$$

Vhodné aktivační funkce

- Vzhledem k aplikacím NN v nespojitém prostředí (digitální svět) jsou běžně užívány numerické derivace (diferencovatelnost) a další metody přibližně definující potřebné hodnoty







Celkový postup algoritmu učení

- Rozdělení dat na testovací, trénovací a validační množinu
- Nastavení vah a prahů sítě, nastavení koeficientu učení α
- Pro každou dvojici: vzor - očekávaný výstup,
  - Spočítání odezvy sítě
  - Spočítání δ_k pro všechny výstupní neurony
  - Spočítání δ_j pro neurony skryté vrstvy
  - Aktualizace vah a prahů
  - Test ukončení algoritmu
    - Maximální počet epoch
    - Změna hodnoty účelové funkce dále neklesá
    - Dosaženo kapacity sítě – výkon na validační množině se zhoršuje → přetrénování

Učení umělé neuronové sítě

- Kvazi Newtonova metoda
  - Založena na aproximaci Hessianovy matice bez nutnosti jejího výpočtu nebo inverze.
  - Metoda aktualizuje aproximaci Hessianovy matice (nebo její inverze) na základě gradientu cílové funkce a rozdílů mezi po sobě jdoucími iteracemi.
  - Pro aktualizaci se často používá Broyden-Fletcher-Goldfarb-Shanno (BFGS)

$$H_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}$$

kde platí:

- $s_k = w_{k+1} - w_k$ je rozdíl parametrů mezi iteracemi.
- $y_k = \nabla E(w_{k+1}) - \nabla E(w_k)$ je rozdíl gradientů chybové funkce.
- I je jednotková matice.

Učení umělé neuronové sítě

- Levenberův-Marquardtův algoritmus (LMA)
  - Algoritmus kombinuje metody nejmenších čtverců a gradientního sestupu.
  - Je zvláště užitečný pro problémy, kde je počet měření větší než počet neznámých.
  - (typické při trénování neuronových sítí s omezeným počtem parametrů a velkým množstvím dat)
  - LMA upravuje Jacobiho matici přidáním diagonální matice, čímž zvyšuje její stabilitu a usnadňuje inverzi.
  - Tento přístup umožňuje algoritmu efektivně přecházet mezi směrem největšího poklesu a kvadratickou konvergencí.

Učení umělé neuronové sítě

- Levenberův-Marquardtův algoritmus (LMA)

  Pro iterativní aktualizaci vah se používá vztah:

$$w_{k+1} = w_k - [J^T J + \lambda I]^{-1} J^T e$$

  kde:

  - J je Jacobiho matice (matice prvních derivací chybové funkce vzhledem k vahám).
  - e je vektor chyb: $e_i = y_i - f(x_i, w_k)$.
  - λ je adaptivní parametr (regulátor kroku), který určuje míru „přiblížení" buď ke gradientnímu sestupu, nebo ke Gauss-Newtonově metodě:
    - **Pokud je λ velké**, metoda se podobá gradientnímu sestupu (menší, ale stabilní kroky).
    - **Pokud je λ malé**, algoritmus se blíží Gauss-Newtonově metodě (rychlá konvergence, ale menší stabilita).

Učení umělé neuronové sítě

- ADAM algoritmus (Adaptive Moment Estimation)
  - Adaptivní optimalizační metoda gradientního sestupu, která se stala velmi populární pro trénování hlubokých neuronových sítí.
  - ADAM kombinuje dvě klíčové techniky
    - Momentum (setrvačnost) – Zahrnuje informace o předchozích gradientech k urychlení sestupu a zabránění oscilací.
    - Adaptivní učení – Dynamicky upravuje learning rate (rychlost učení) pro každý parametr zvlášť na základě historie gradientů.

Učení umělé neuronové sítě

- ADAM algoritmus (Adaptive Moment Estimation)

1. **Výpočet gradientu:**
   Spočítáme gradient chybové funkce vzhledem k váhám:

$$g_k = \nabla_w E(w_k)$$

2. **Aktualizace momentů:**
   - Aktualizujeme první moment (průměr gradientů):

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$$

   - Aktualizujeme druhý moment (průměr kvadrátu gradientů):

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$$

3. **Korekce vychýlení momentů (bias-correction):** Jelikož na začátku jsou momenty vychýlené (směrem k nule), použijeme korekci:

$$\hat{m}_k = \frac{m_k}{1 - \beta_1^k}, \quad \hat{v}_k = \frac{v_k}{1 - \beta_2^k}$$

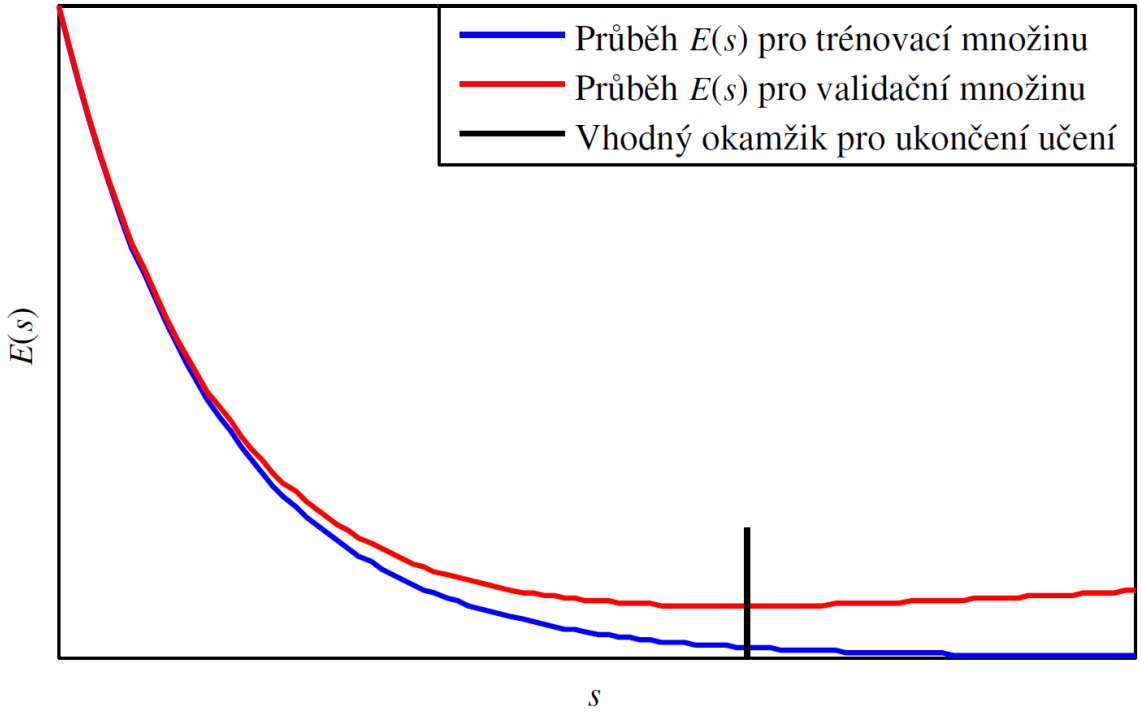
4. **Aktualizace parametrů:** Nové váhy jsou vypočteny jako:

$$w_{k+1} = w_k - \eta \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \varepsilon}$$

Učení umělé neuronové sítě

- Každá z metod má své specifické využití a optimální použití závisí na konkrétním problému a struktuře dat.
- Kvazi-Newtonovy metody jsou vhodné pro problémy s malým až středním množstvím parametrů, kde je výpočet nebo aproximace Hessianovy matice proveditelný.
- Levenberg-Marquardtův algoritmus exceluje v problémech transformovatelných na optimalizaci pomocí nejmenších čtverců.
- Zpětné šíření chyby je jednoduchým algoritmem, implementovatelným pro většinu neuronových sítí.

Podmínka ukončení trénování - přetrénování



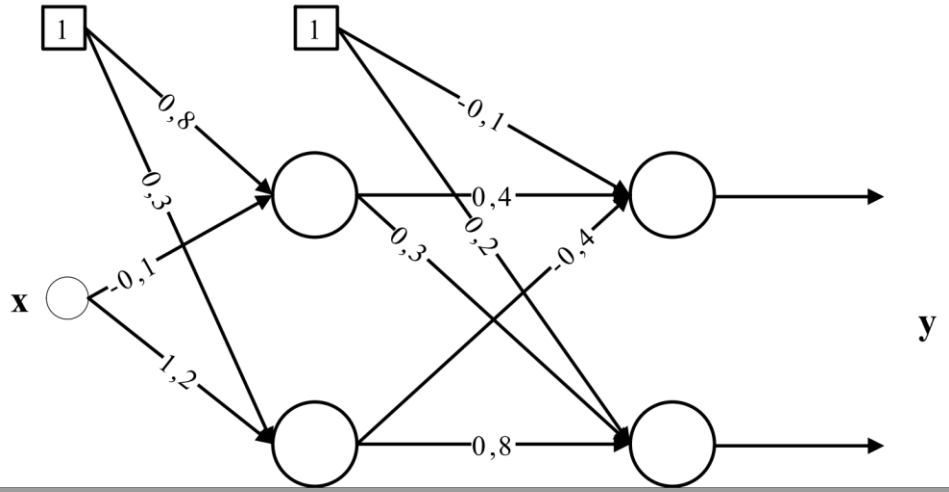
Varianty učení ANN

- Online
  - Po každém vzoru dojde k aktualizaci hodnot vah a prahů.
- Offline
  - Změny vah a prahů se sčítají v dočasné proměnné a jejich aktualizace se provede až na konci epochy trénování (po celé epoše).
- Dávkové (batch)
  - Kombinuje předchozí varianty, kde se aktualizace provádí po dávkách vzorků (většinou po 64, 32, 16, 8, 4 vzorech).
  - Kombinuje výhody a nevýhody obou variant (kompromis rychlosti a paměťové náročnosti)

Výpočty BGD s FFNN

Výpočet odezvy FFNN

- Odezvu FFNN lze počítat obdobně jako u jednoduchého perceptronu s tím rozdílem, že se signál propaguje dál sítí, takže se výpočty postupně provádí dokud není dosaženo konkrétní výstupní hodnoty.

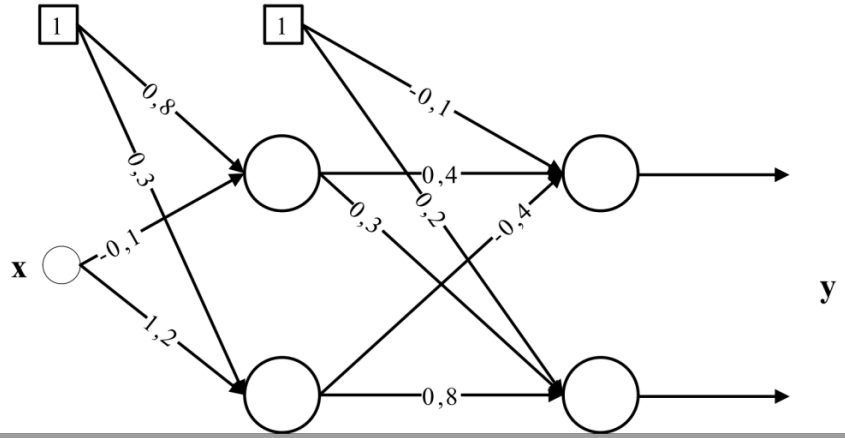


$y = \tanh y_a$ $y = y_a$

Výpočet odezvy FFNN

- Je patrné, že:

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}8 & 0{,}3 \\ -0{,}1 & 1{,}2 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} -0{,}1 & 0{,}2 \\ 0{,}4 & 0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix}$$



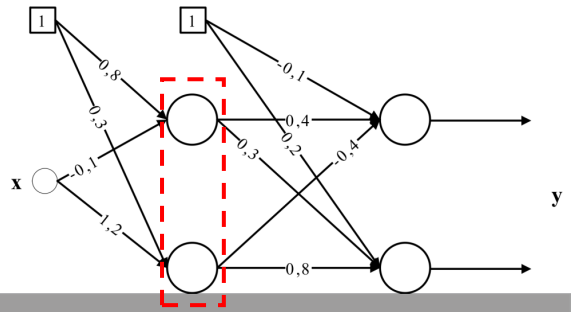
Výpočet odezvy FFNN

- Pro vzor x = 0,25, počteme odezvu následovně:

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}8 & 0{,}3 \\ -0{,}1 & 1{,}2 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} -0{,}1 & 0{,}2 \\ 0{,}4 & 0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix}$$

$$y_a^1 = (\mathbf{W}^1)^T \cdot y^0 = (\mathbf{W}^1)^T \cdot \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} 0{,}8 & -0{,}1 \\ 0{,}3 & 1{,}2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0{,}25 \end{bmatrix} = \begin{bmatrix} 0{,}8 - 0{,}1 \cdot 0{,}25 \\ 0{,}3 + 1{,}2 \cdot 0{,}25 \end{bmatrix} = \begin{bmatrix} 0{,}775 \\ 0{,}600 \end{bmatrix}$$

$$y^1 = \tanh y_a^1 = \begin{bmatrix} 0{,}649 \\ 0{,}537 \end{bmatrix}$$

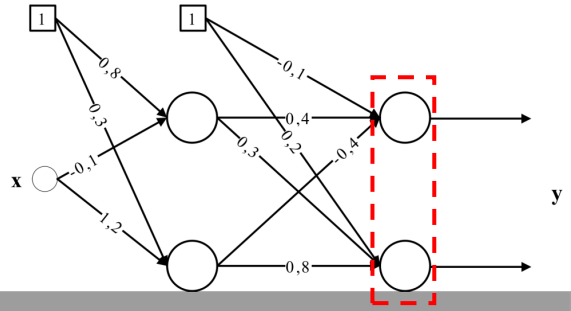


Výpočet odezvy FFNN

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}8 & 0{,}3 \\ -0{,}1 & 1{,}2 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} -0{,}1 & 0{,}2 \\ 0{,}4 & 0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix} \qquad y^1 = \tanh y_a^1 = \begin{bmatrix} 0{,}649 \\ 0{,}537 \end{bmatrix}$$

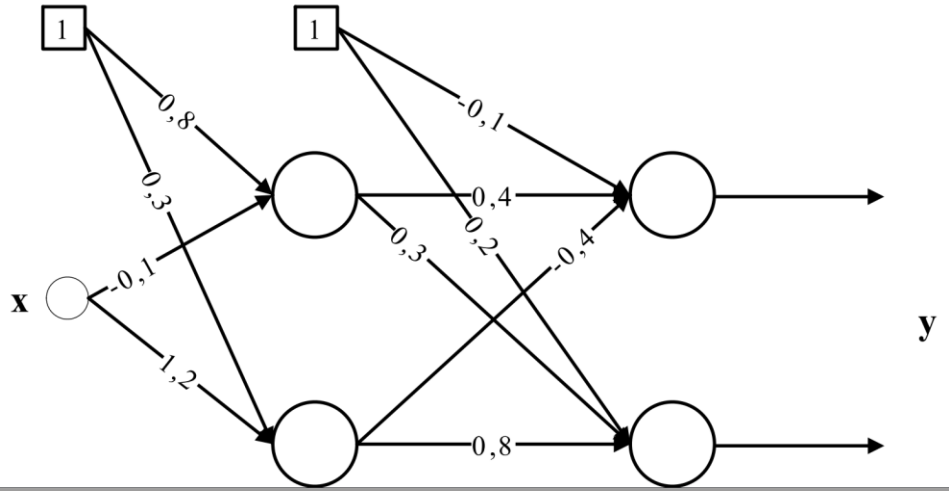
$$y_a^2 = (\mathbf{W^2})^T \cdot y^1 = \begin{bmatrix} -0{,}1 & 0{,}4 & -0{,}4 \\ 0{,}2 & 0{,}3 & 0{,}8 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0{,}649 \\ 0{,}537 \end{bmatrix} = \begin{bmatrix} -0{,}0549 \\ 0{,}8246 \end{bmatrix}$$

$$y^2 = y_a^2 = \begin{bmatrix} -0{,}0549 \\ 0{,}8246 \end{bmatrix}$$



Výpočet BGD pro FFNN

- Pro učení je nejprve vypočítat odezvu FFNN, poté je třeba určit chybu a provézt výpočty a aktualizace vah. Pro vzor x = 0,25, t = $\begin{bmatrix} 0{,}45 \\ -0{,}2 \end{bmatrix}$



$y = \tanh y_a$ $y = y_a$

Výpočet BGD pro FFNN

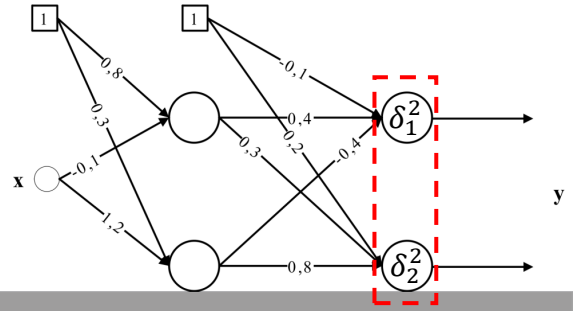
$x = 0{,}25,\ t = \begin{bmatrix} 0{,}45 \\ -0{,}2 \end{bmatrix}\ \mathbf{W}^1 = \begin{bmatrix} 0{,}8 & 0{,}3 \\ 0{,}1 & 1{,}2 \end{bmatrix}\ \mathbf{W}^2 = \begin{bmatrix} -0{,}1 & 0{,}2 \\ 0{,}4 & 0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix}\ y = \begin{bmatrix} -0{,}0549 \\ 0{,}8246 \end{bmatrix}$

$$e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} t_1 - y_1 \\ t_2 - y_2 \end{bmatrix} = \begin{bmatrix} 0{,}45 - (-0{,}0549) \\ -0{,}2 - 0{,}8246 \end{bmatrix} = \begin{bmatrix} 0{,}5049 \\ -1{,}0246 \end{bmatrix}$$

- Derivace lineární identické funkce je 1, takže

$$\delta_1^2 = e_1 \cdot \phi'(y_{a1}^2) = e_1 \cdot 1 = e_1 = 0{,}5049$$

$$\delta_2^2 = e_2 \cdot \phi'(y_{a2}^2) = e_2 \cdot 1 = e_2 = -1{,}0246$$

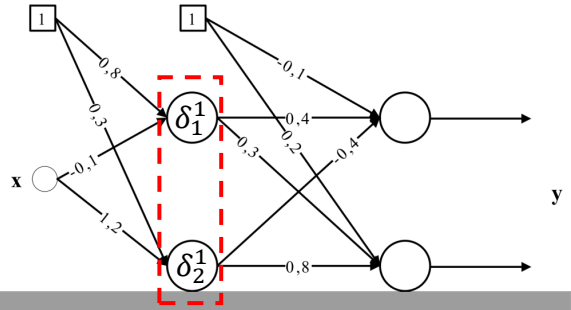


Výpočet BGD pro FFNN

- Derivace hyperbolicko-tangenciální funkce je: $\boldsymbol{\phi^{k'}(y_{aj}^{k}) = 1 - \phi^{k}(y_{j}^{k})^{2}}$, takže

$$\phi^{1'}(y_{a1}^{1}) = \phi^{1'}(0{,}775) = 1 - 0{,}649^{2} = 0{,}5777$$
$$\phi^{1'}(y_{a2}^{1}) = \phi^{1'}(0{,}600) = 1 - 0{,}537^{2} = 0{,}7116$$

$$\tanh y_{a}^{1} = \begin{bmatrix} 0{,}649 \\ 0{,}537 \end{bmatrix}$$

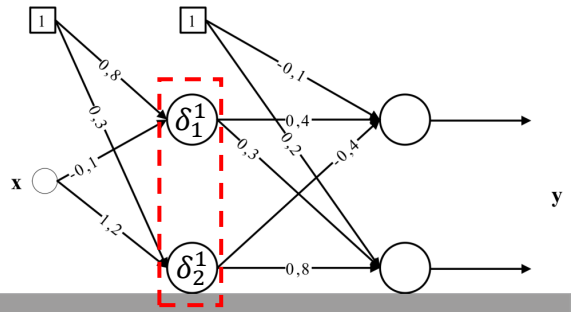


Výpočet BGD pro FFNN

- Derivace hyperbolicko-tangenciální funkce je: $\boldsymbol{\phi^{k'}(y_{aj}^{k}) = 1 - (y_{j}^{k})^{2}}$, takže

$$\delta_1^1 = \phi^{1'}(y_{a1}^1) \sum_{l=1}^{2} \delta_l^2 \cdot w_{1l}^2 = 0{,}5777(0{,}5049 \cdot 0{,}4 + (-1{,}0246) \cdot 0{,}3) = -0{,}0609$$

$$\delta_2^1 = \phi^{1'}(y_{a2}^1) \sum_{l=1}^{2} \delta_l^2 \cdot w_{2l}^2 = -0{,}7270$$



Výpočet BGD pro FFNN

- Lokální gradienty jsou vypočítané, nyní lze přistoupit k zpětnému šíření chyby

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}8 & 0{,}3 \\ 0{,}1 & 1{,}2 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} -0{,}1 & 0{,}2 \\ 0{,}4 & 0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix}$$

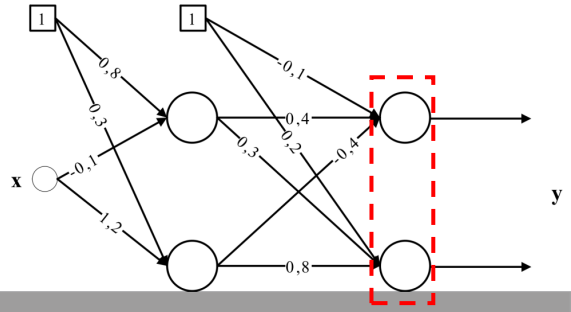
$\delta_1^2 = 0{,}5049; \delta_2^2 = -1{,}0246$

$\Delta w_{01}^2 = \alpha \delta_1^2 y_0^1 = 0{,}1 \cdot (0{,}5049) \cdot 1 = 0{,}0505$

$w_{01}^2 = w_{01}^2 + \Delta w_{01}^2 = -0{,}1 + 0{,}0505 = -0{,}0495$

$\Delta w_{11}^2 = \alpha \delta_1^2 y_1^1 = 0{,}1 \cdot (0{,}5049) \cdot 0{,}649 = 0{,}0328$

$w_{11}^2 = w_{11}^2 + \Delta w_{11}^2 = 0{,}4 + 0{,}0328 = 0{,}4328$



Výpočet BGD pro FFNN

- Lokální gradienty jsou vypočítané, nyní lze přistoupit k zpětnému šíření chyby

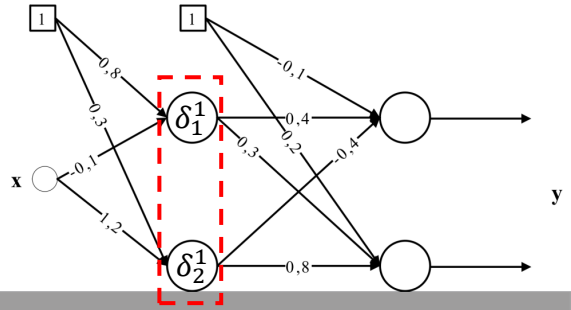
$$\mathbf{W}^1 = \begin{bmatrix} 0{,}8 & 0{,}3 \\ 0{,}1 & 1{,}2 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} -0{,}1 & 0{,}2 \\ 0{,}4 & 0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix}$$

$\delta_1^1 = -0{,}0609$
$\delta_2^1 = -0{,}7270$

$$\Delta w_{01}^1 = \alpha \delta_1^1 y_0^0 = 0{,}1 \cdot (-0{,}0609) \cdot 1 = -0{,}0061$$

$$w_{01}^1 = w_{01}^1 + \Delta w_{01}^1 = 0{,}8 - 0{,}0061 = 0{,}7939$$

$$\Delta w_{11}^1 = \alpha \delta_1^1 y_1^0 = 0{,}1 \cdot (-0{,}0609) \cdot 0{,}25 = -0{,}0015$$



Metody stanovení topologie FFNN

Optimalizace FFNN

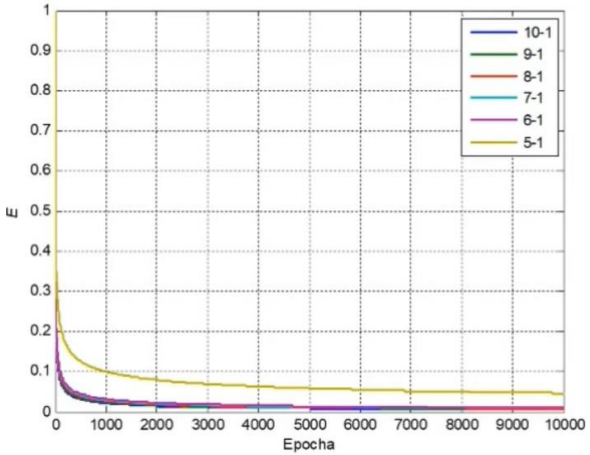
- Optimalizace topologie (počet vrstev, počet neuronů v každé vrstvě, volba aktivační funkce, …)
- Optimalizace vah a prahů (učení umělé neuronové sítě)

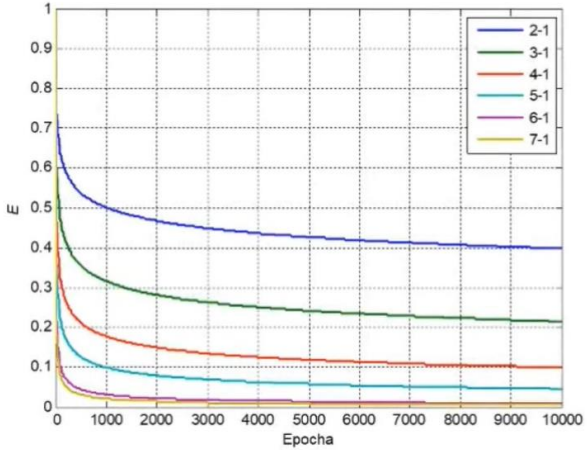
Optimalizace topologie FFNN

- Neexistuje analytický postup
- Počet vstupů a výstupů je dán problémem
- Počet skrytých vrstev

Optimalizace topologie FFNN

- Počet neuronů ve skrytých vrstvách
  - Různá doporučení v literatuře, ze zkušeností u 1 vrstvé ANN: $p = \sqrt{RQ}$
  - Metoda shora dolů
  - Metoda zdola nahoru





Řešení problémů pomocí FFNN

Dopředná vícevrstvá umělá neuronová síť

- Je vhodná k řešení určitých typů problémů:
  - Aproximace / regrese
    - Nahrazení lokálního funkčního předpisu funkce jeho přibližným vyjádřením pomocí funkce. Účelem je snížení výpočetní náročnosti. Zjednodušení probíhá na úkor přesnosti.
    - Libovolná spojitá funkce muže být s požadovanou přesností aproximovaná pomocí dopředné vícevrstvé umělé neuronové sítě.
  - Modelování a predikce
    - Každý systém je možno zobrazit jako blok (model), na který působí vstupní signály a případně poruchy a jehož chování je možno sledovat na základe výstupních veličin
  - Rozpoznávání vzorů a klasifikace
    - Rozpoznávání vzorů a jejich třídění do příslušných tříd.
  - Rozhodování

Dopředná vícevrstvá umělá neuronová síť

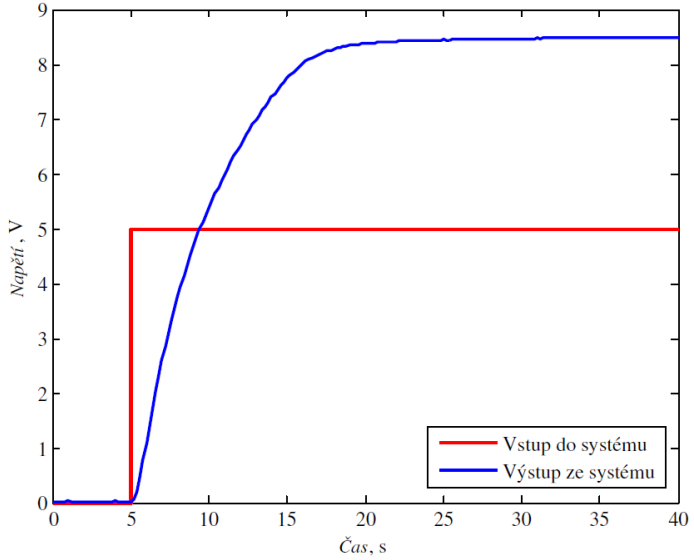
Úloha	Typ výstupu	Výstupní neurony	Aktivační funkce	Význam výstupu
Regrese	Reálné číslo (např. 23.5)	1	Žádná (lineární výstup)	Predikovaná hodnota
Binární klasifikace	Pravděpodobnost (0–1)	1	Sigmoid	Příslušnost ke kladné třídě
Klasifikace více tříd	Vektor pravděpodobností (např. [0.1, 0.7, 0.2])	n tříd	Softmax	Rozdělení pravděpodobností, součet = 1
Multilabel klasifikace	Vektor nezávislých pravděpodobností	n labelů	Sigmoid (nezávisle)	P(ke každému labelu zvlášť), bez nutnosti součtu
Klasifikace s prahováním	Binární hodnota (0 nebo 1)	1 nebo n	Sigmoid + threshold (např. 0.5)	Rozhodnutí na základě pravděpodobnosti (> práh)

Experimenty s neuronovými sítěmi

- Stanovení metodiky (stanovení cíle, předzpracování dat, volba topologie, kolik experimentů, jak budou testovány výsledky)
- Naměření/získání datasetu (dat k trénování, dat popisující chování, které chceme modelovat, dat popisujících skupiny, které chceme shlukovat/rozdělovat) a jeho úprava
- Jednotlivé experimenty s NN – vždy provádět učení minimálně z 10 různých počátečních podmínek pro výběr nejvhodnějšího modelu
- Při hledání vhodné topologie daného typu NN se běžně používají porovnávání výsledků na testovací množině

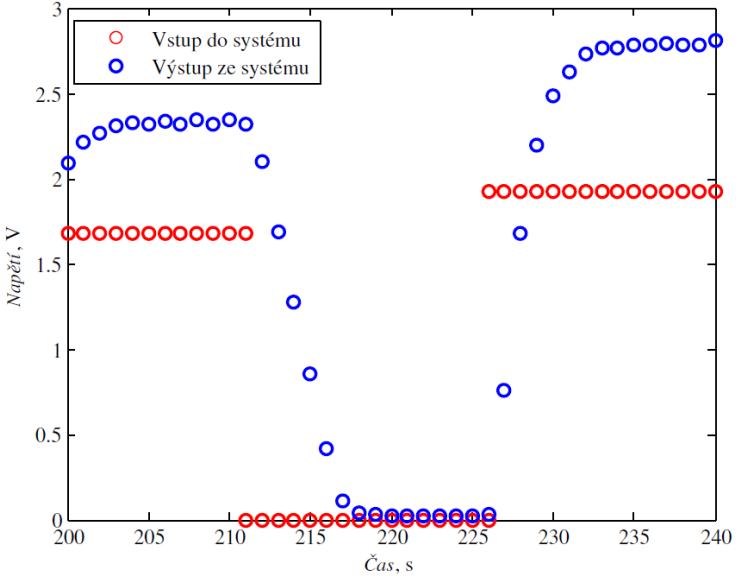
Dopředná vícevrstvá umělá neuronová síť

- Modelování a predikce – Příklad **Soustava pro řízení otáček**
  - Dynamiku je vhodné testovat na celém rozsahu (přes celou statickou charakteristiku)



Dopředná vícevrstvá umělá neuronová síť

- Modelování a predikce – Příklad **Soustava pro řízení otáček**



Tabulka 2: Část naměřených dat

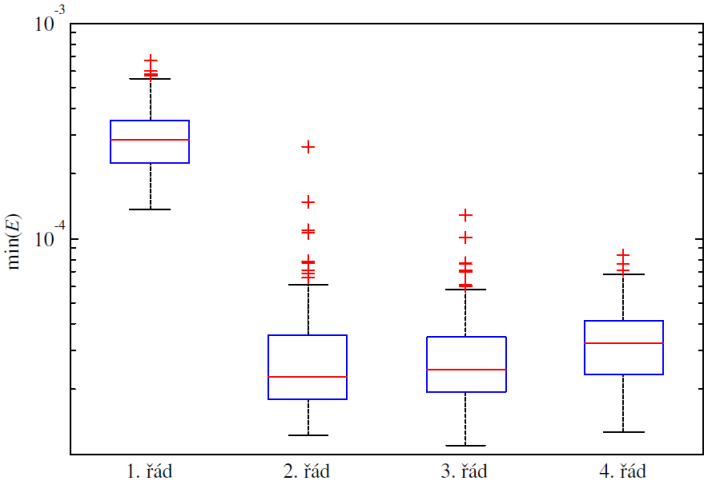
k	$u(k)$, V	$y_S(k)$, V	$u(k)$, norm	$y_S(k)$, norm
1	3,7521	0,0195	0,5008	-0,9961
2	3,7521	0,9375	0,5008	-0,8125
3	3,7521	2,3389	0,5008	-0,5322
4	3,7521	3,5205	0,5008	-0,2959
5	3,7521	4,4434	0,5008	-0,1113
6	3,7521	5,1855	0,5008	0,0371
7	3,7521	5,6934	0,5008	0,1387
8	3,7521	5,9717	0,5008	0,1943
9	3,7521	6,1328	0,5008	0,2266
10	3,7521	6,2451	0,5008	0,249
11	3,7521	6,2891	0,5008	0,2578
12	3,7521	6,3184	0,5008	0,2637
13	3,7521	6,3428	0,5008	0,2686
14	3,7521	6,3574	0,5008	0,2715
15	3,7521	6,3672	0,5008	0,2734
16	5,0000	6,3867	1,0000	0,2739
17	5,0000	6,5967	1,0000	0,3193
⋮	⋮	⋮	⋮	⋮
1000	1,689	2,3438	-0,3244	-0,5313

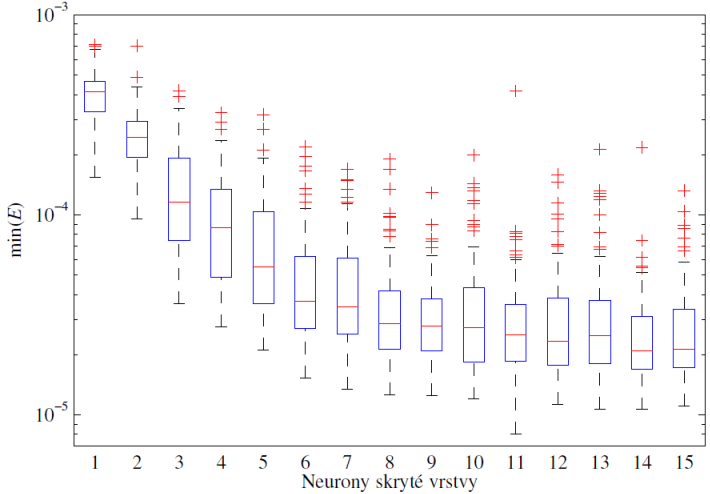
Tabulka 3: Část trénovací množiny

k	Vstupy				Cíle
	$y_S(k-1)$	$y_S(k-2)$	$u(k-1)$	$u(k-2)$	$y_S(k)$
3	-0,8125	-0,9961	0,5008	0,5008	-0,5322
4	-0,5322	-0,8125	0,5008	0,5008	-0,2959
5	-0,2959	-0,5322	0,5008	0,5008	-0,1113
6	-0,1113	-0,2959	0,5008	0,5008	0,0371
7	0,0371	-0,1113	0,5008	0,5008	0,1387
8	0,1387	0,0371	0,5008	0,5008	0,1943
9	0,1943	0,1387	0,5008	0,5008	0,2266
10	0,2266	0,1943	0,5008	0,5008	0,249
11	0,249	0,2266	0,5008	0,5008	0,2578
12	0,2578	0,249	0,5008	0,5008	0,2637
13	0,2637	0,2578	0,5008	0,5008	0,2686
14	0,2686	0,2637	0,5008	0,5008	0,2715
15	0,2715	0,2686	0,5008	0,5008	0,2734
16	0,2734	0,2715	0,5008	0,5008	0,2739
17	0,2739	0,2734	1	0,5008	0,3193
⋮	⋮	⋮	⋮	⋮	⋮

Dopředná vícevrstvá umělá neuronová síť

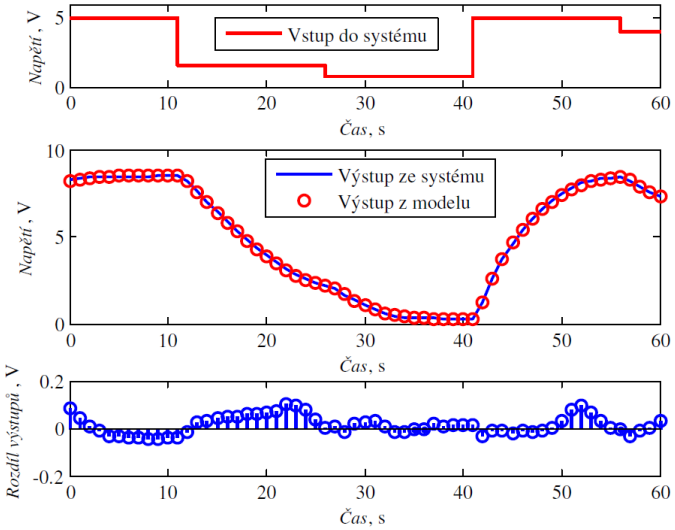
- Modelování a predikce – Příklad **Soustava pro řízení otáček**
  - Pro potřeby identifikace často určujeme řád modelu, poté provádíme optimalizace samotného počtu neuronů ve skryté vrstvě

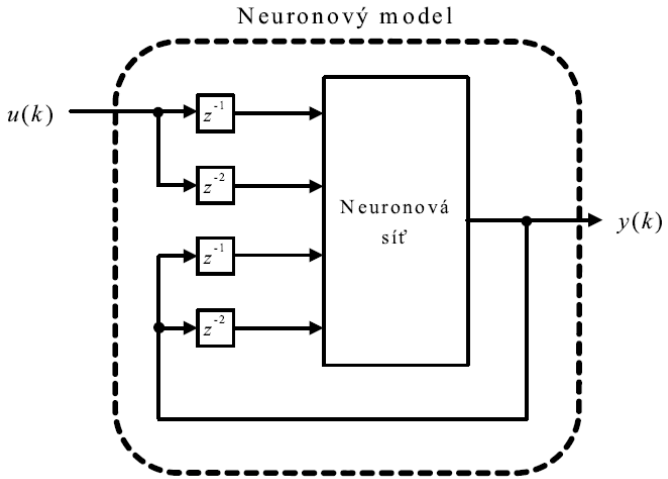




Dopředná vícevrstvá umělá neuronová síť

- Modelování a predikce – Příklad **Soustava pro řízení otáček**
  - Provedeme testování a při splnění požadovaného chování určíme NN model



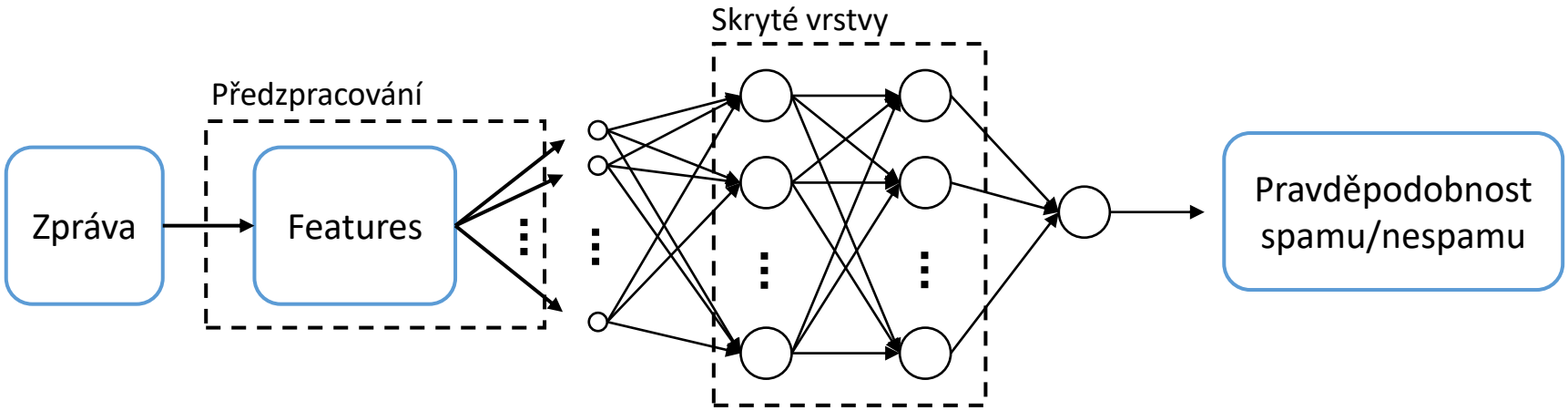


Binární klasifikace

- Příklad na detekci spamu
  - Vstupní proměnné
    - Většinou numerické reprezentace slov (např. TF-IDF, word embedding, bag-of-words).
    - Počet vstupů může být od desítek po tisíce podle velikosti slovníku.
  - Výstup:
    - nespam vs. spam
    - v praxi reprezentováno jedním neuronem se sigmoid aktivací (1 = spam) nebo dvěma neurony se softmax.

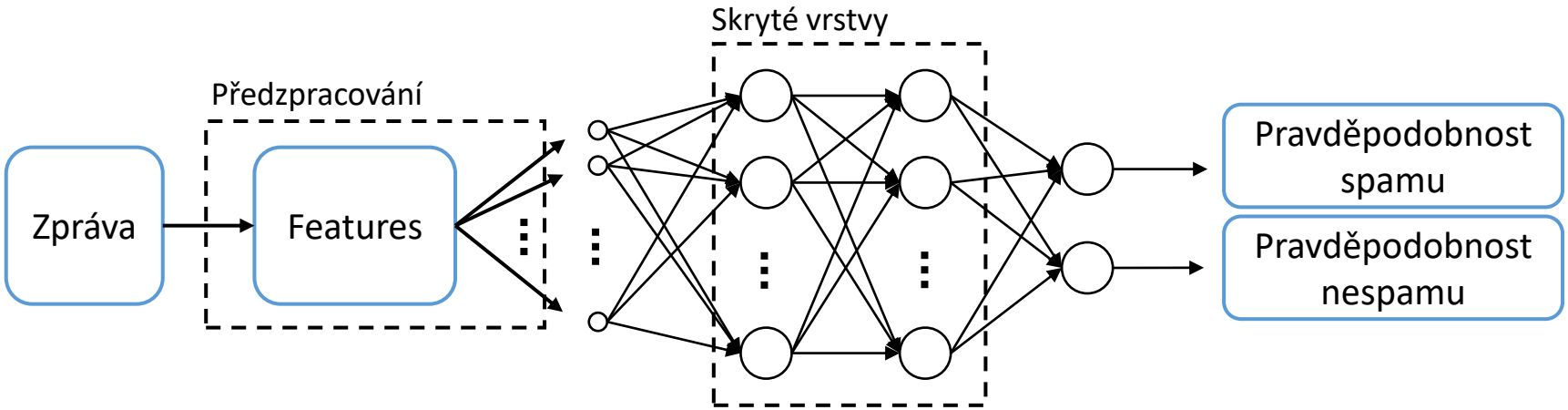
Binární klasifikace

- Příklad na detekci spamu



Binární klasifikace

- Příklad na detekci spamu

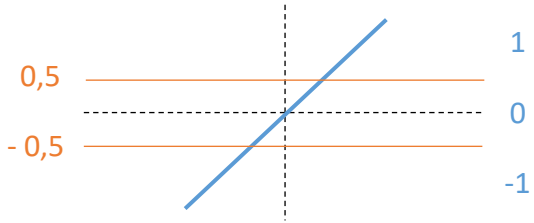


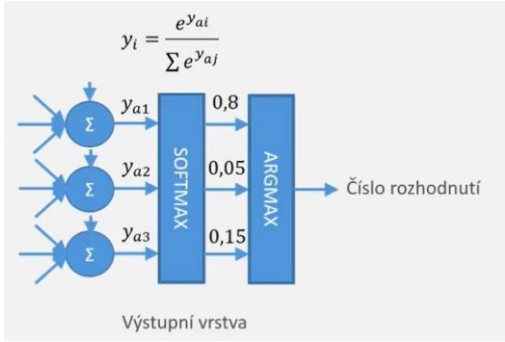
Klasifikace a rozpoznávání vzorů

- Příklad na klasifikaci kosatců - výška/šířka kališního/okvětního lístku

Klasifikace a rozpoznávání vzorů

- Příklad na klasifikaci kosatců - výška/šířka kališního/okvětního lístku





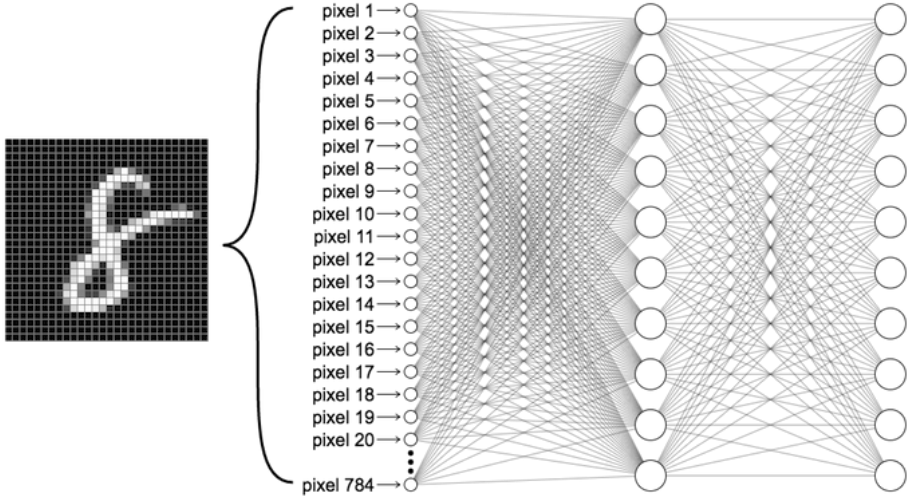
- Nešlo by druh kosatce určit lépe?
- Pravděpodobnost příslušnosti k dané třídě – aktivační funkce softmax

Klasifikace a rozpoznávání vzorů

- Klasifikace obrazových dat
  - Přímé zpracování pomocí FFNN (pixel po pixelu)
  - Užití předzpracování obrazu pro extrakci klíčových vlastností
    - Detekce hran
    - Extrakce vlastností pomocí histogramů orientovaných gradientů
  - Využití konvolučních neuronových sítí

Klasifikace a rozpoznávání vzorů

- Klasifikace obrazových dat
  - Přímé zpracování pomocí FFNN (pixel po pixelu)

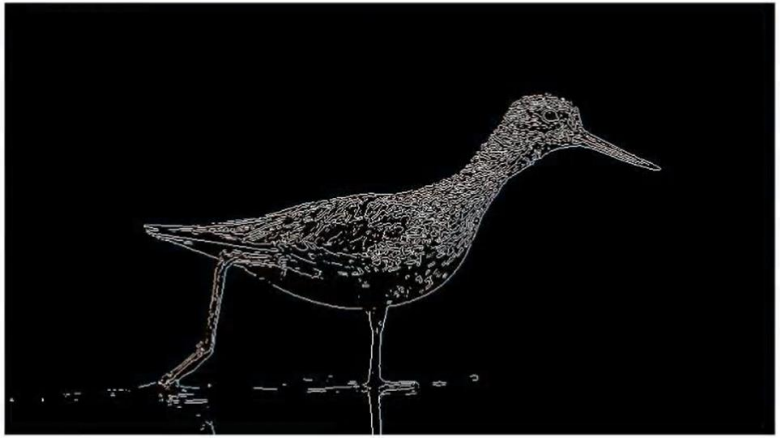


Klasifikace a rozpoznávání vzorů

- Klasifikace obrazových dat
  - Přímé zpracování pomocí FFNN (pixel po pixelu)
    - Topologie je využívá tzv. serializace vstupu, to znamená, že ze vstupní obrazové matice pospojuje jednotlivé řádky nebo sloupce do jednoho vektoru.
    - Vnitřní topologie odpovídá standardní vícevrstvé neuronové síti a výstupní vrstva obsahuje počet neuronů odpovídající počtu klasifikačních tříd

Klasifikace a rozpoznávání vzorů

- Užití předzpracování obrazu pro extrakci klíčových vlastností
  - Motivace – člověk je instinktivně schopen určit důležitou část obrazu (oddělit pozadí od sledovaného objektu), stroj sám o sobě nedisponuje takovou schopností – je potřeba mu klíčové informace „zvýraznit“

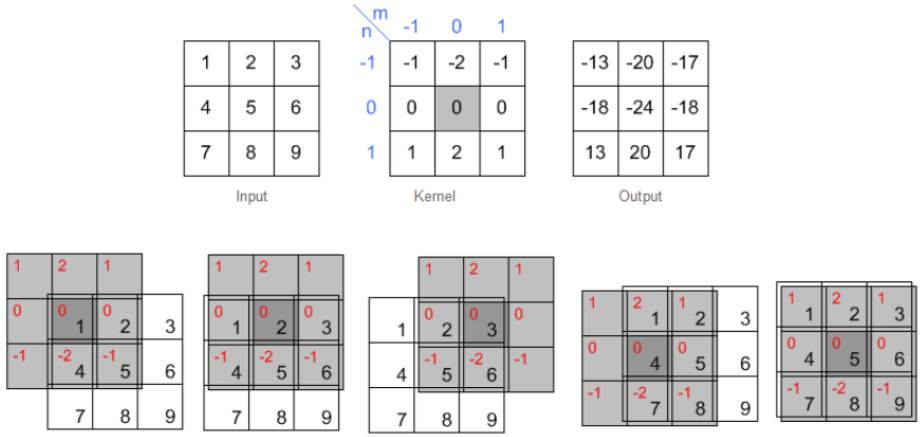


Klasifikace a rozpoznávání vzorů

- Detekce hran – gradient
  - Prakticky je definiční obor jasové funkce diskrétní, gradient vypočítáme pomocí diferencí jasové funkce
  - Častou jsou implementovány tzv. masky zvýrazňující přechody (hrany)
    - Roberts, Prewitt, Sobel, …

Klasifikace a rozpoznávání vzorů

- Po vynásobení masky skrze obrázek dochází obecně k filtraci, v případě vhodných masek pak k zvýraznění určitých vlastností (typicky hran)



Klasifikace a rozpoznávání vzorů

- Př. Originální a zašuměný obrázek

Klasifikace a rozpoznávání vzorů

- Př. Box filtr

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Klasifikace a rozpoznávání vzorů

- Př. Gausův filtr – běžně je použitá mat. funkce násobící každý pixel, nicméně se dá aproximovat do podoby masky

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Klasifikace a rozpoznávání vzorů

- Př. Matice pro Prewittův a Sobelův operátor a Laplacián

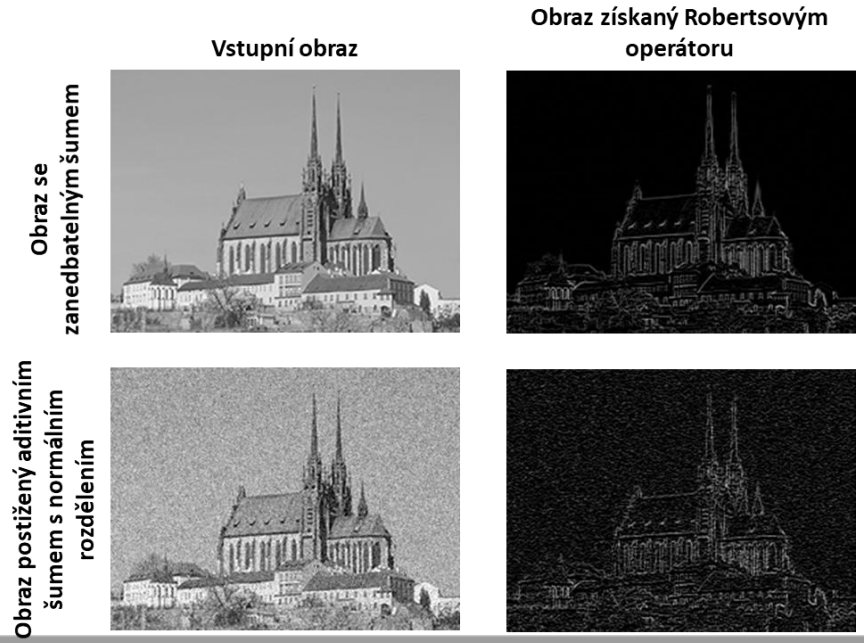
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Klasifikace a rozpoznávání vzorů

- Příklad aplikace Robetsova operátoru

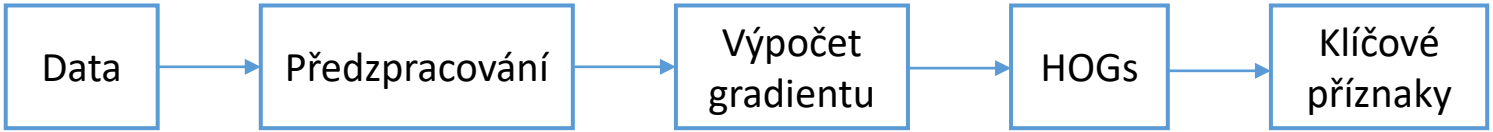


Klasifikace a rozpoznávání vzorů

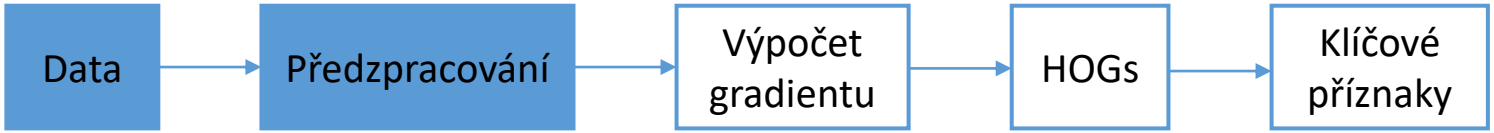
- Pomocí metod detekce hran, lze použít přímo detekované hrany nebo vstupní obrazová data upravit a použít takto vzniklé obrázky obdobně jako při klasifikaci s FFNN

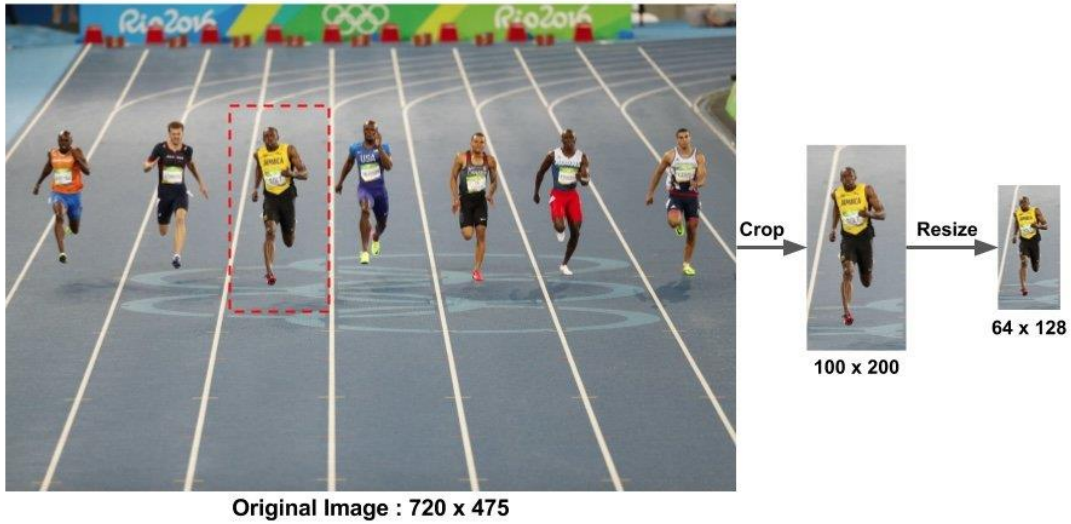
Klasifikace a rozpoznávání vzorů

- Extrakce vlastností pomocí histogramů orientovaných gradientů
  - Zobecnění detekce hran
  - Tvar a vzhled objektu je možné charakterizovat pomocí gradientu jasové funkce
  - Výsledkem je informace o dominantních tvarech v obrazu

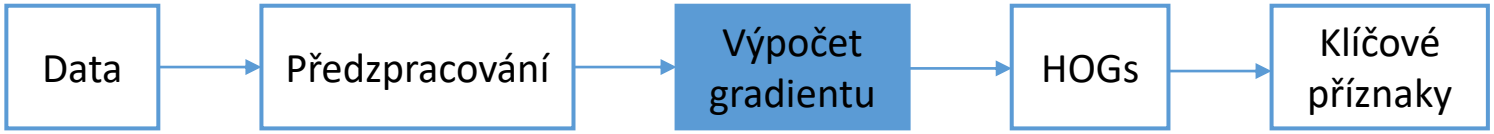


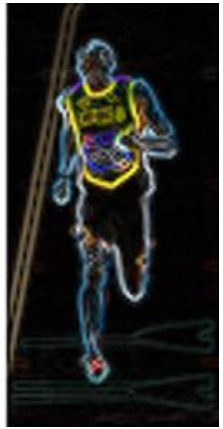
Extrakce vlastností pomocí HOGs



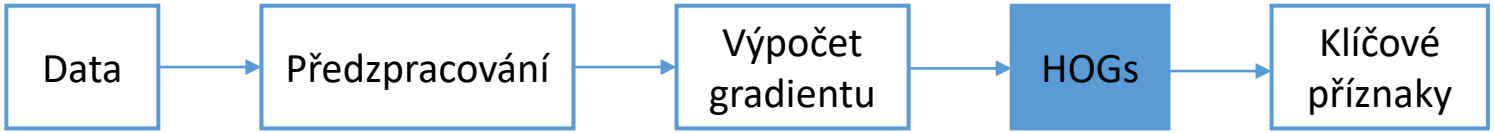


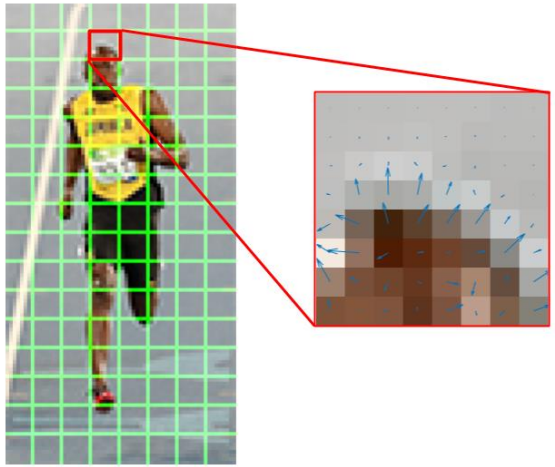
Extrakce vlastností pomocí HOGs





Extrakce vlastností pomocí HOGs





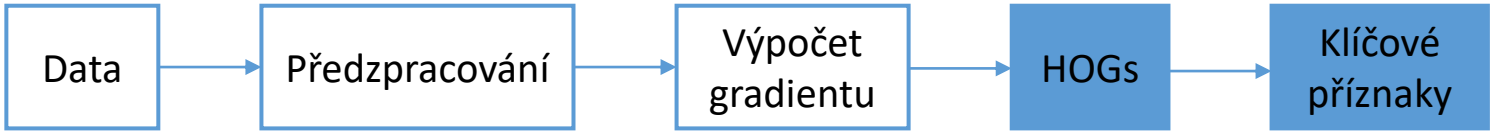
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

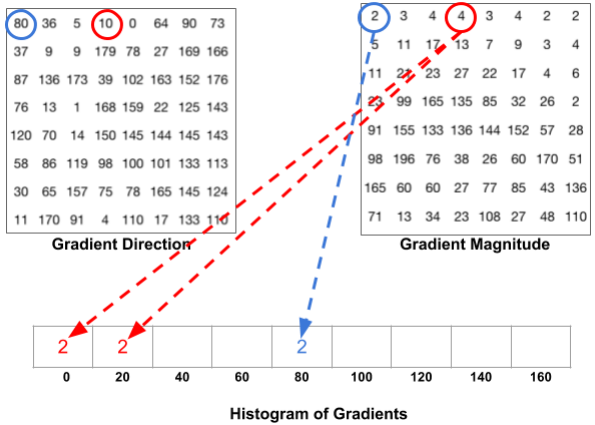
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

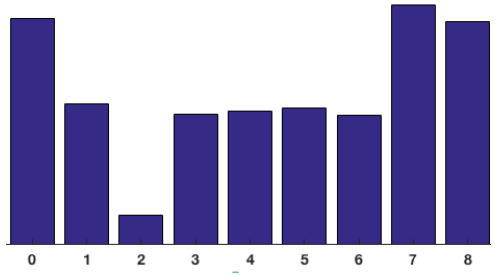
Extrakce vlastností pomocí HOGs



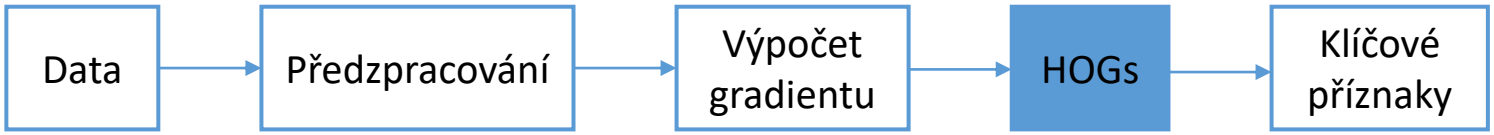
Magnitudy jednotlivých pixelů přiřazuji s danou váhou do příslušných skupin úhlů gradientu

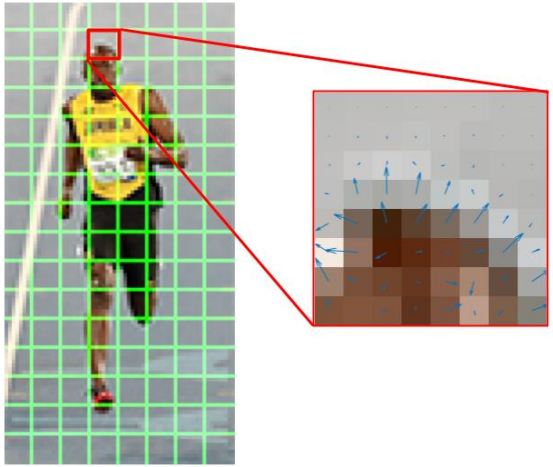


Vektor klíčových vlastností



Extrakce vlastností pomocí HOGs



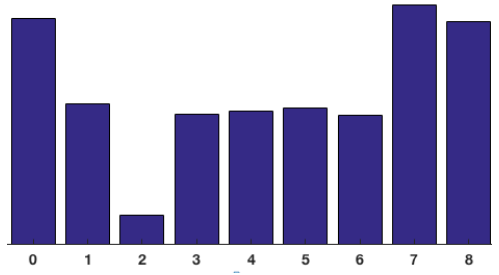


2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction



Metriky pro hodnocení neuronových sítí

Metriky pro hodnocení neuronových sítí

- Klíčové je rozdělení datasetu – metriky se hodnotí nad testovacími daty
  - Jednotlivé metody je potřeba vždy porovnávat na stejné testovací množině
  - Rozdělení je běžně 70:15:15 (trénování:validace:testování)
  - Vyhodnocení modelu se vždy provádí pouze nad testovacím datasetem
  - !Pozor neplést s nejnižší možnou trénovací chybou → je pravděpodobné, že pokud není model přeučen, tak s nižší chybou bude lépe performovat i na testovací sadě

Metriky pro hodnocení neuronových sítí

- Podle řešeného problému se aplikují metriky pro hodnocení neuronových sítí.
- Metriky související s regresí a modelováním:
  - Jejich použití je jednoduché. Na testovací množině se aplikuje konkrétní metrika, která určuje přesnost modelu.
  - Střední kvadratická chyba
    - Nejoblíbenější metrikou používanou pro regresní problémy. V podstatě zjišťuje průměrnou kvadratickou chybu mezi předpovídanou a skutečnou hodnotou.
  - Střední absolutní chyba
    - Zjišťuje průměrnou absolutní vzdálenost mezi předpovídanou a cílovou hodnotou.

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2 \qquad MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

Metriky pro hodnocení neuronových sítí

- Podle řešeného problému se aplikují metriky pro hodnocení neuronových sítí.
- Metriky související s klasifikací:
  - Klasifikace je jedním z nejpoužívanějších problémů strojového učení s různými průmyslovými aplikacemi, od rozřazování do skupin podle vlastností, rozpoznávání obličejů, kategorizace videí (např. na YouTube), moderování obsahu, lékařské diagnostiky až po klasifikaci textu (detekci nenávistných projevů na Twitteru).
  - Klasifikační metriky používají základní hodnocení, zda predikce odpovídá příslušnosti do dané třídy nebo nikoli.
  - Jsou rozlišována 4 hodnocení: TP, FP, FN, TN

Metriky pro hodnocení neuronových sítí

- Jsou rozlišována 4 hodnocení: TP, FP, FN, TN
  - TP – True Positive
    - Predikovaná třída je shodná s třídou daného vzorku testovací množiny.
  - FP – False Positive
    - Byla přiřazena třída, která ovšem neměla být přiřazena.
  - FN – False Negative
    - Nebyla přiřazena třída, která měla být přiřazena.
  - TN – True Negative
    - Značí výskyty ostatních tříd, které správně nebyly detekovány.

Metriky pro hodnocení neuronových sítí

- Jsou rozlišována 4 hodnocení: TP, FP, FN, TN – Graficky pomocí tzv. confusion matrix (matice záměn).

		Predikovaná třída	
		Ano	Ne
Reálná třída	Ano	**TP**	**FN**
	Ne	**FP**	**TN**

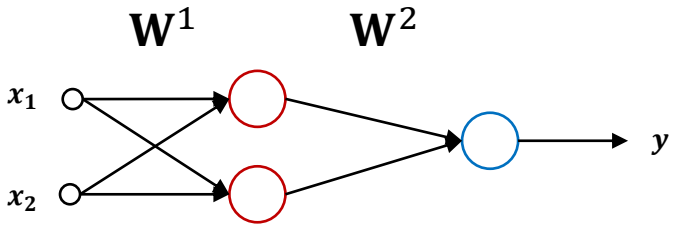
Metriky pro hodnocení neuronových sítí

		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	**Negative Predictive Value** $\frac{TN}{(TN + FN)}$	**Accuracy** $\frac{TP + TN}{(TP + TN + FP + FN)}$

Výpočty BGD s FFNN

Výpočty BGD s FFNN

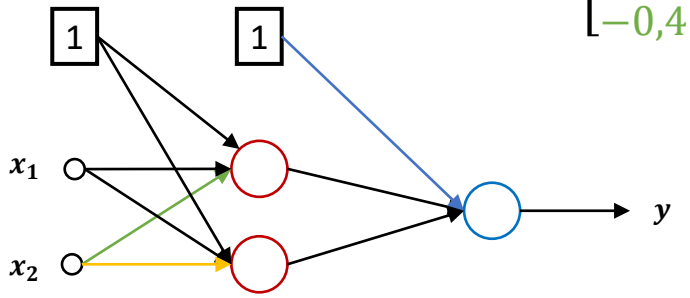
- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů



Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}3 \\ -0{,}4 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$



$x_1 = 1$
$x_2 = 0{,}5$

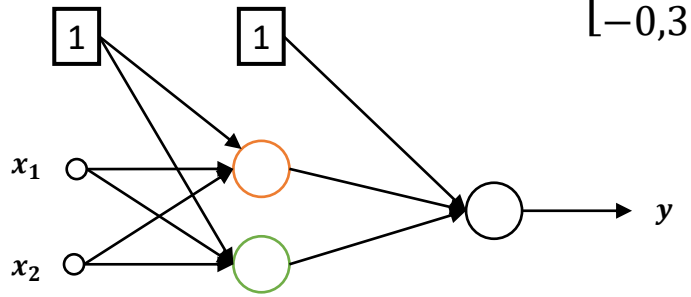
$t = 0{,}5$

$y = \tanh y_a$ $\qquad$ $y = y_a$

Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$



$x_1 = 1$
$x_2 = 0{,}5$

$t = 0{,}5$

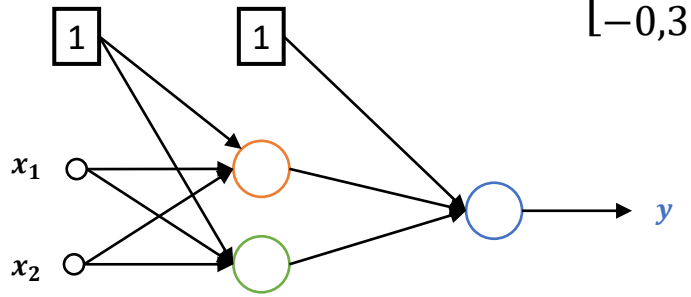
$y = \tanh y_a$ $\qquad$ $y = y_a$

$z_1 = 0{,}1 + 0{,}5 \cdot x_1 - 0{,}3 \cdot x_2 = 0{,}45; a_1 = \tanh(0{,}45) = 0{,}421$
$z_2 = 0{,}2 - 0{,}4 \cdot x_1 + 0{,}8 \cdot x_2 = 0{,}2; a_2 = \tanh(0{,}2) = 0{,}197$

Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$



$x_1 = 1$
$x_2 = 0{,}5$

$t = 0{,}5$

$y = \tanh y_a$ $\qquad$ $y = y_a$

$z_1 = 0{,}1 + 0{,}5 \cdot x_1 - 0{,}3 \cdot x_2 = 0{,}45; a_1 = \tanh(0{,}45) = 0{,}421$
$z_2 = 0{,}2 - 0{,}4 \cdot x_1 + 0{,}8 \cdot x_2 = 0{,}2; a_2 = \tanh(0{,}2) = 0{,}197$

$y = 0{,}05 + 0{,}7 \cdot a_1 - 0{,}6 \cdot a_2 = 0{,}227$

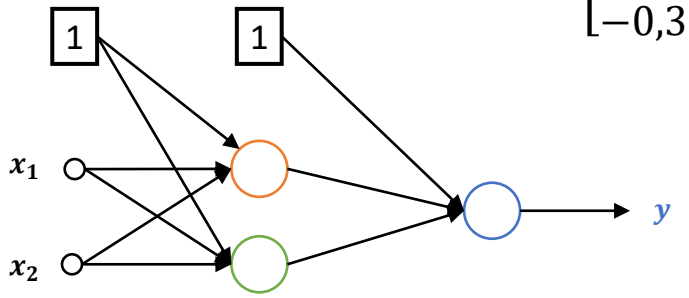
Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů
- Uvažujme kvadratickou chybu

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$

$$E = \frac{1}{2}(t-y)^2,$$

$$\frac{\partial E}{\partial y} = \frac{1}{2} \cdot 2(t-y) \cdot (-1) = y - t$$



$x_1 = 1$
$x_2 = 0{,}5$

$t = 0{,}5$

$y = \tanh y_a$ $\qquad$ $y = y_a$

$$\frac{\partial E}{\partial y} = 0{,}227 - 0{,}5 = -0{,}273$$

$z_1 = 0{,}1 + 0{,}5 \cdot x_1 - 0{,}3 \cdot x_2 = 0{,}45; a_1 = \tanh(0{,}45) = 0{,}421$
$z_2 = 0{,}2 - 0{,}4 \cdot x_1 + 0{,}8 \cdot x_2 = 0{,}2; a_2 = \tanh(0{,}2) = 0{,}197$

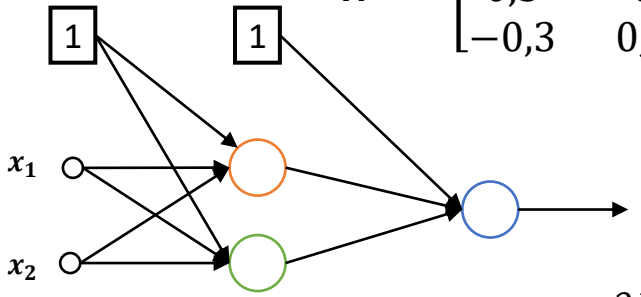
$y = 0{,}05 + 0{,}7 \cdot a_1 - 0{,}6 \cdot a_2 = 0{,}227$

Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů
- Gradient pro výstupní váhy

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$

$$\frac{\partial E}{\partial w_i^2} = \frac{\partial E}{\partial y} \cdot a_i$$



$x_1 = 1$
$x_2 = 0{,}5$

$$\frac{\partial E}{\partial w_0^2} = -0{,}273 \cdot 1$$

$$\frac{\partial E}{\partial w_1^2} = -0{,}273 \cdot 0{,}421 = -0{,}115$$

$$\frac{\partial E}{\partial w_2^2} = -0{,}273 \cdot 0{,}197 = -0{,}054$$

$$\frac{\partial E}{\partial y} = 0{,}227 - 0{,}5 = -0{,}273$$

$z_1 = 0{,}45; a_1 = \tanh(0{,}45) = 0{,}421$
$z_2 = 0{,}2; a_2 = \tanh(0{,}2) = 0{,}197$

$y = 0{,}227$

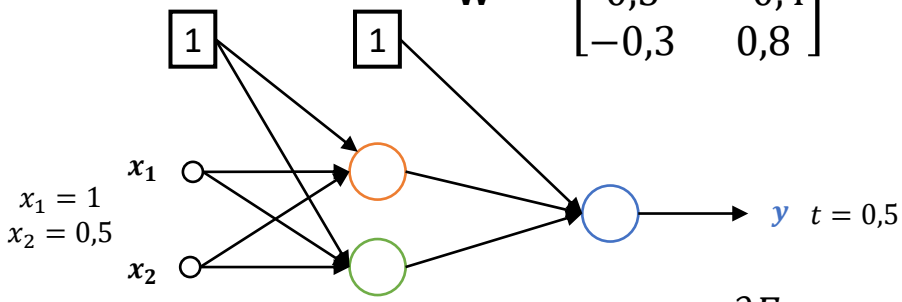
Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů
- Gradient pro váhy skryté vrstvy

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$

$$\frac{\partial E}{\partial z_i} = \frac{\partial E}{\partial y} \cdot w_i^2 \cdot \frac{d}{dz}\tanh(z)$$

$$\frac{d}{dz}\tanh(z) = 1 - \tanh^2(z)$$



$$\frac{\partial E}{\partial z_1^1} = -0{,}273 \cdot 0{,}7 \cdot (1 - \overset{0{,}823}{0{,}421^2}) = -0{,}157$$

$$\frac{\partial E}{\partial z_2^1} = -0{,}273 \cdot -0{,}6 \cdot (1 - \underset{0{,}961}{0{,}197^2}) = 0{,}157$$

$y = \tanh y_a$ $\quad y = y_a$

$z_1 = 0{,}45; a_1 = \tanh(0{,}45) = 0{,}421$
$z_2 = 0{,}2; a_2 = \tanh(0{,}2) = 0{,}197$

$y = 0{,}227$

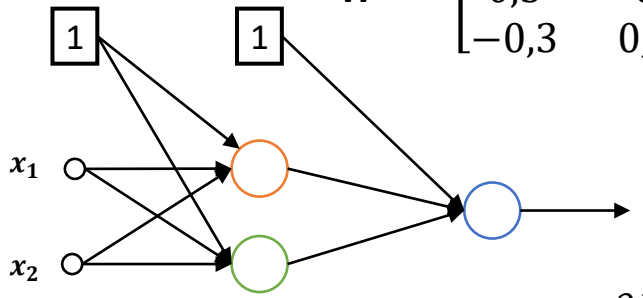
$$\frac{\partial E}{\partial y} = 0{,}227 - 0{,}5 = -0{,}273$$

Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů
- Gradient pro váhy skryté vrstvy

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$

$$\frac{\partial E}{\partial w_{ij}^1} = \frac{\partial E}{\partial z_i} \cdot x_i$$



$$\frac{\partial E}{\partial w_{01}^1} = -0{,}157$$

$$\frac{\partial E}{\partial w_{11}^1} = -0{,}157 \cdot x_1 = -0{,}157$$

$$\frac{\partial E}{\partial w_{21}^1} = -0{,}157 \cdot x_2 = -0{,}0785$$

$y = \tanh y_a$ $\quad y = y_a$

$$\frac{\partial E}{\partial y} = 0{,}227 - 0{,}5 = -0{,}273$$

$z_1 = 0{,}45; a_1 = \tanh(0{,}45) = 0{,}421$
$z_2 = 0{,}2; a_2 = \tanh(0{,}2) = 0{,}197$

$y = 0{,}227$

+ výpočet pro váhy z 2 neuronu skryté vrsty

Výpočty BGD s FFNN

- Mějme takto definovanou síť, kterou chceme naučit odčítání vstupů
- Vypočtené gradienty vah se pak přičtou vynásobené koef. Rychlosti učení k původním vahám

$$\mathbf{W}^1 = \begin{bmatrix} 0{,}1 & 0{,}2 \\ 0{,}5 & -0{,}4 \\ -0{,}3 & 0{,}8 \end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix} 0{,}05 \\ 0{,}7 \\ -0{,}6 \end{bmatrix}$$

Implementace
FFNN v pyTorch

Úvod do PyTorch

- Instalace přes pip (pip install torch)
- https://pytorch.org/docs/stable/nn.html
- PyTorch – tensor
  - torch.Tensor = základní datová struktura
  - Podobné jako NumPy pole, ale může běžet na GPU

```
import torch

a = torch.tensor([[1., 2.], [3., 4.]])
b = torch.rand(2, 2)

print(a + b)
```

Tvorba modelu

- nn.Module
  - Základní třída všech modelů v PyTorch
- Obsahuje
  - konstruktor __init__() – kde se definují vrstvy
  - metodu forward() – definuje výpočetní tok (tzv. forward pass)

```
import torch
import torch.nn as nn

class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(4, 10)   # Vstupní vrstva: 4 -> 10
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(10, 3)   # Výstupní vrstva: 10 -> 3

    def forward(self, x):
        x = self.relu(self.fc1(x))
        return self.fc2(x)
```

Tvorba modelu

- Více vrstev → schopnost modelovat složitější nelinearity
- Aktivace umožňují nelineární transformace → bez nich by síť byla jen lineární mapa
  - ReLU – rychlá, standardní volba (většina moderních sítí)
  - Tanh – symetrická, používá se méně často
  - Sigmoid – u binární klasifikace

```
class DeepMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(4, 32),
            nn.Tanh(),
            nn.Linear(32, 16),
            nn.ReLU(),
            nn.Linear(16, 3)
        )

    def forward(self, x):
        return self.net(x)
```

Tvorba modelu

- Regularizace
  - Dropout: náhodně vypíná některé neurony během tréninku → zabraňuje přeučení
  - BatchNorm: stabilizuje výstupy z vrstev → urychluje trénink, snižuje kolísání

```
class RegularizedMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(4, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 3)
        )

    def forward(self, x):
        return self.model(x)
```

Datasety

- TensorDataset: jednoduchý obal nad (x, y)
- Používáme, pokud máme data v NumPy nebo pandas

```
from sklearn.datasets import load_iris
from torch.utils.data import TensorDataset

iris = load_iris()
X = torch.tensor(iris.data, dtype=torch.float32)
y = torch.tensor(iris.target, dtype=torch.long)

dataset = TensorDataset(X, y)
```

Datasety

- Dělení dat:
  - random_split(dataset, [train, val]) – náhodně rozdělí

```
from torch.utils.data import random_split

train_ds, val_ds = random_split(dataset, [120, 30])
```

  - Subset + indexy – umožní přesnější dělení (např. stratifikaci)

```
from sklearn.model_selection import train_test_split
from torch.utils.data import Subset

idx_train, idx_val = train_test_split(range(len(dataset)), test_size=0.2, stratify=y)
train_ds = Subset(dataset, idx_train)
val_ds = Subset(dataset, idx_val)
```

Datasety

- DataLoader(dataset, batch_size, shuffle, num_workers)
- Vytváří dávky a iteruje přes ně

```
from torch.utils.data import DataLoader

train_loader = DataLoader(train_ds, batch_size=16, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=32)
```

  - batch_size – kolik vzorků v jednom kroku
  - shuffle=True – promíchá data mezi epocha
  - minum_workers – paralelní načítání (funguje mimo Windows)

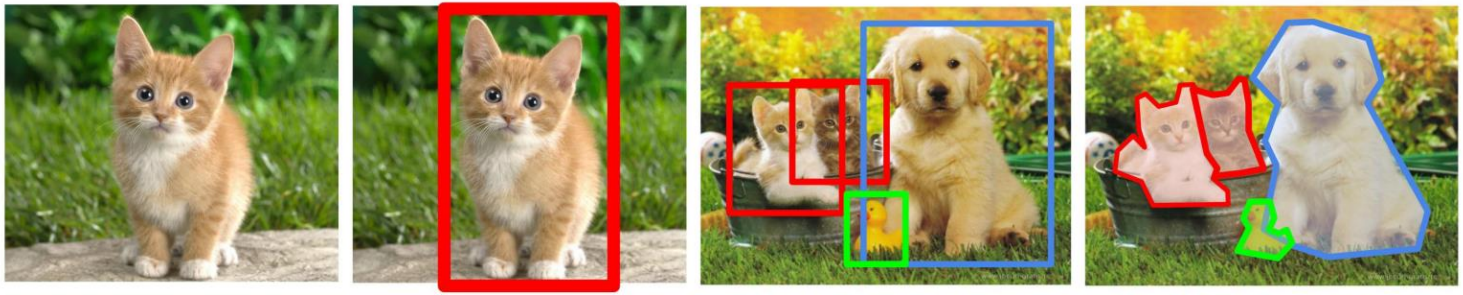
Konvoluční neuronová síť (CNN)

Problémy zpracování obrazu

- Klasifikace – zařazení vstupního obrázku do příslušné třídy
- Lokalizace – nalezení objektu ve vstupním obrázku
- Detekce – souběžná lokalizace a klasifikace
- Segmentace – rozdělení vstupního obrázku na segmenty
  - Sémantická – každý pixel je přiřazen do určité třídy
  - Instanční – označuje pixely odpovídající jednotlivým instancím objektu

Problémy zpracování obrazu

- Klasifikace – zařazení vstupního obrázku do příslušné třídy
- Lokalizace – nalezení objektu ve vstupním obrázku
- Detekce – souběžná lokalizace a klasifikace
- Segmentace – rozdělení vstupního obrázku na segmenty



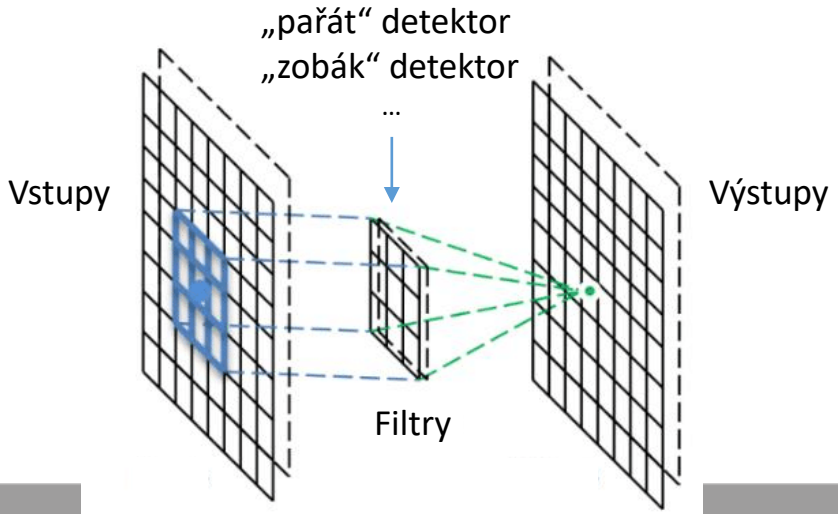
Konvoluční neuronová síť – myšlenka

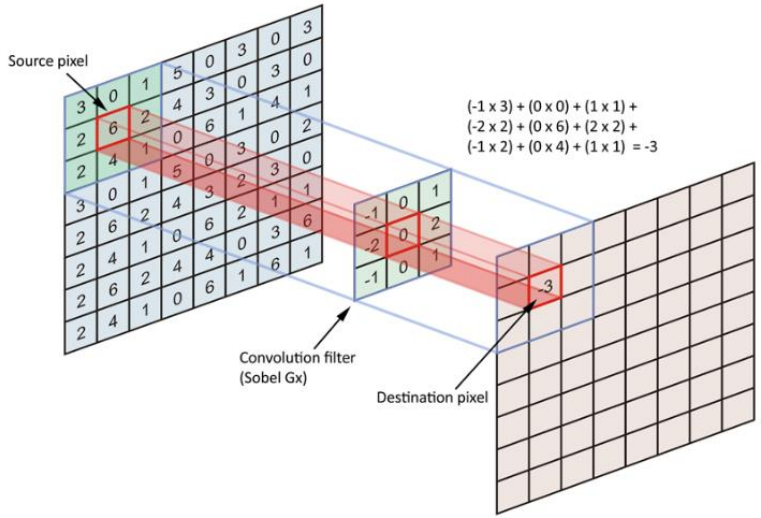
- Typicky člověk vybírá v datech specifické příznaky – hrany, tvary, barvy
- V datech se obecně nacházejí vzory (vlastnosti) definující dané objekty
- Tyto vzory mohou být různých velikostí a různě umístěné v datech
- → návrh skupiny detektorů, které procházejí zpracovávaná data
- Jednotlivé části dat mohou být kódovány stejným detektorem (stejným způsobem)

Konvoluční neuronová síť – myšlenka

Konvoluční neuronová síť - úvod

- Jednotlivé detektory jsou realizované pomocí konvoluční vrstvy
- Konvoluční vrstva je představovaná skupinou filtrů provádějících operaci konvoluce





Konvoluční neuronová síť – konvoluce

- 1 konvoluční vrstva je tvořena definovaným počtem filtrů, které detekují vzory definované velikosti (velikost jádra), váhy filtrů jsou získány trénováním.

Obrázek 6x6

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Filtr 1 (3x3)

1	-1	-1
-1	1	-1
-1	-1	1

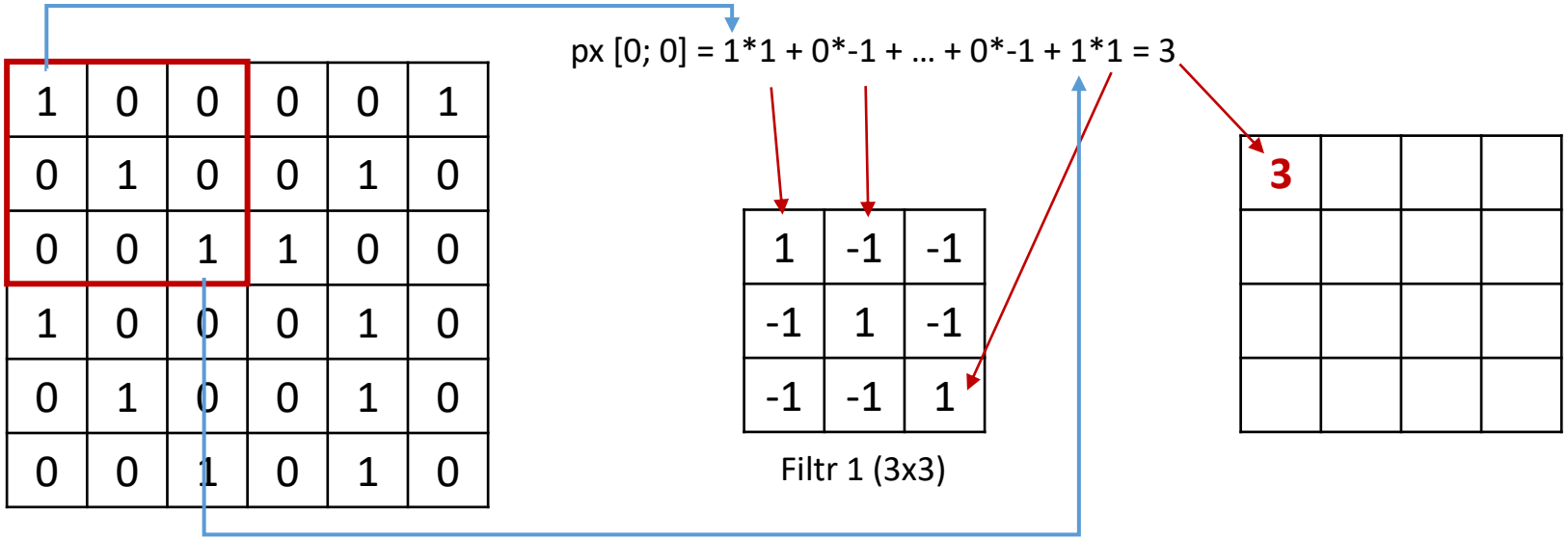
⋮ ⋮

Filtr n (3x3)

-1	1	-1
-1	1	-1
-1	1	-1

Konvoluční neuronová síť – konvoluce

- Pro každý pixel aplikujeme skalární součin konvolučního filtru s vstupem a získáváme výstupní hodnotu tvořící matici příznaků



Konvoluční neuronová síť – konvoluce

- Pro každý pixel aplikujeme skalární součin konvolučního filtru s vstupem a získáváme výstupní hodnotu tvořící matici příznaků

Stride = 1 (po kolika pixelech brát další blok)

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Filtr 1 (3x3)

3	**-1**		

Konvoluční neuronová síť – konvoluce

- Pro každý pixel aplikujeme skalární součin konvolučního filtru s vstupem a získáváme výstupní hodnotu tvořící matici příznaků

Stride = 1 (po kolika pixelech brát další blok)

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Filtr 1 (3x3)

3	-1	**-3**	

Konvoluční neuronová síť – konvoluce

- Pro každý pixel aplikujeme skalární součin konvolučního filtru s vstupem a získáváme výstupní hodnotu tvořící matici příznaků

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Filtr 1 (3x3)

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	**-1**

Konvoluční neuronová síť – konvoluce

- Matice příznaků určuje, kde se v původním obrázku vyskytují dominantní příznaky shodné s těmi ve filtru
- Určuje, kde v obrázku jsou přítomné dané tvary

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Filtr 1 (3x3)

Matice příznaků – filtr 1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Konvoluční neuronová síť – konvoluce

- Matice příznaků určuje, kde se v původním obrázku vyskytují dominantní příznaky shodné s těmi ve filtru
- Určuje, kde v obrázku jsou přítomné dané tvary

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

-1	1	-1
-1	1	-1
-1	1	-1

Filtr n (3x3)

Matice příznaků – filtr n

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

Konvoluční neuronová síť – konvoluce

- Aplikací všech filtrů pak dochází k zisku mapy příznaků

Obrázek 6x6

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Filtr 1 (3x3)

⋮

Filtr n (3x3)

Mapa příznaků
jednotlivé matice příznaku za sebou

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

Konvoluční neuronová síť – konvoluce

- Padding (vyplňování) vs. Normálně – dochází ke snížení velikosti

Obrázek 6x6

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Filtr 1 (3x3)

1	-1	-1
-1	1	-1
-1	-1	1

Matice příznaků 4x4

3			

Konvoluční neuronová síť – konvoluce

- Padding = Same – Zachování velikosti obrazu

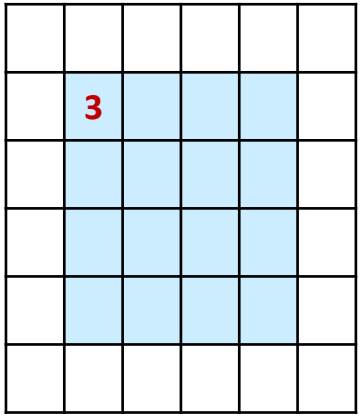
Obrázek 6x6 + výplň (celkem 8x8)

	1	0	0	0	0	1	
	0	1	0	0	1	0	
	0	0	1	1	0	0	
	1	0	0	0	1	0	
	0	1	0	0	1	0	
	0	0	1	0	1	0	

Filtr 1 (3x3)

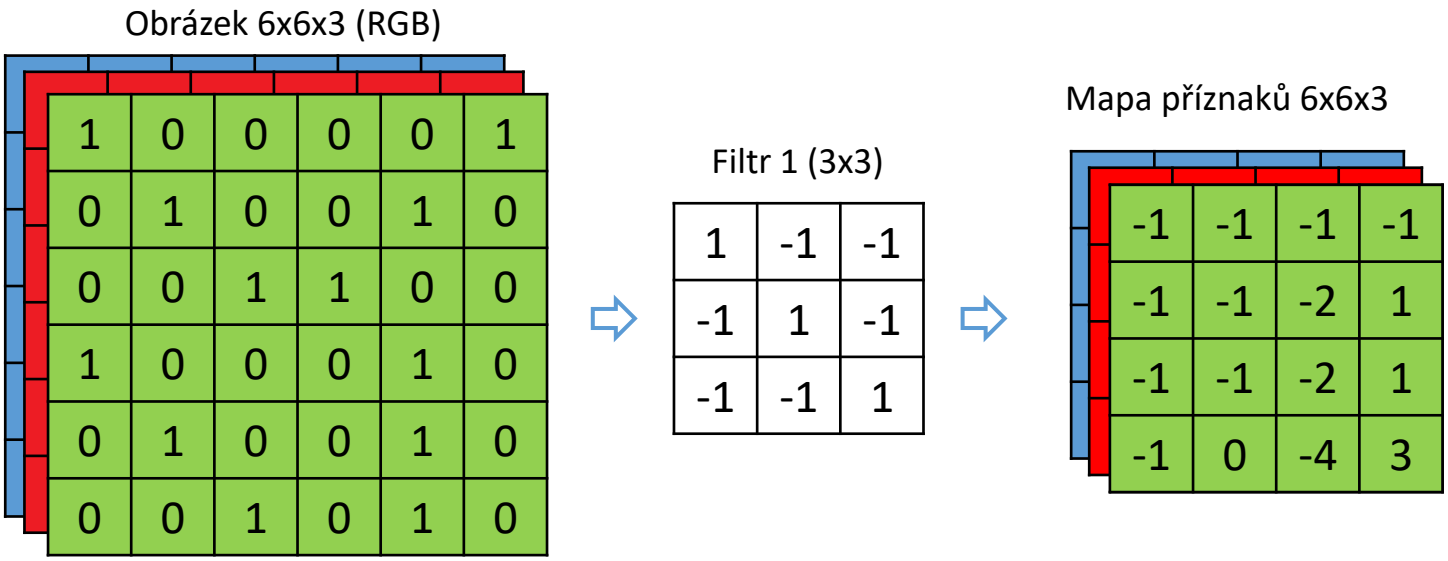
1	-1	-1
-1	1	-1
-1	-1	1

Matice příznaků 6x6



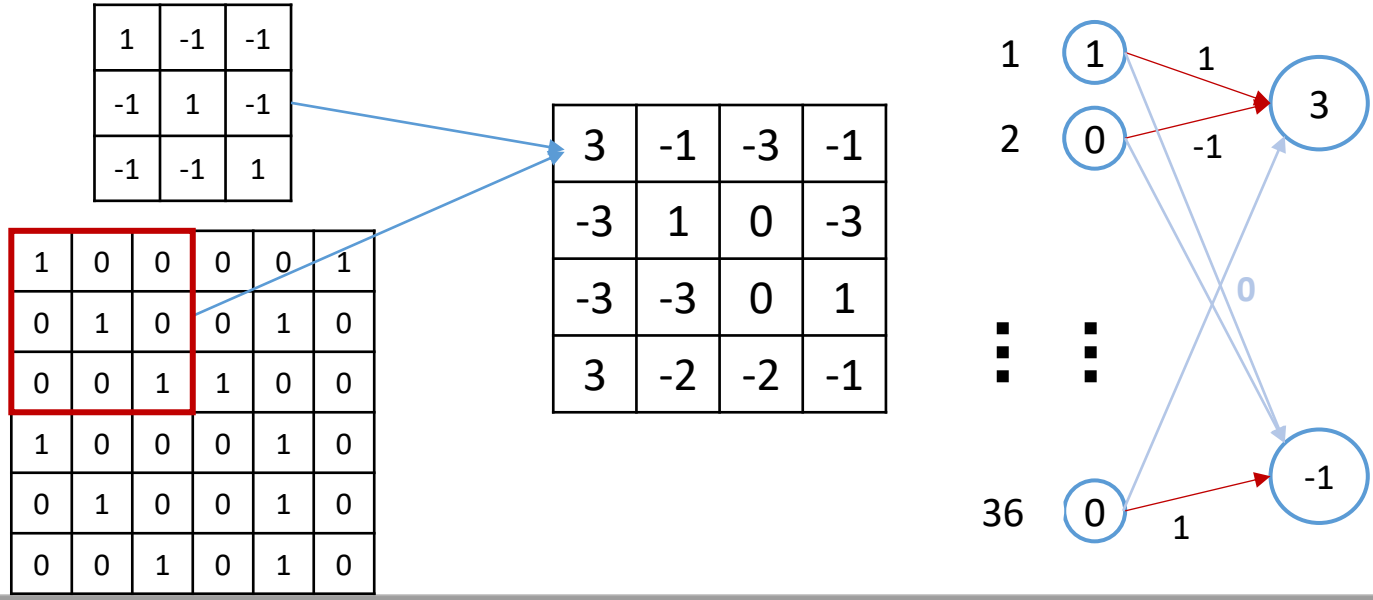
Konvoluční neuronová síť – konvoluce

- U barevného obrázku (RGB) se provádí pro každý kanál
- Ziskem jsou mapy příznaků (dominantních vlastnosti vstupních dat)



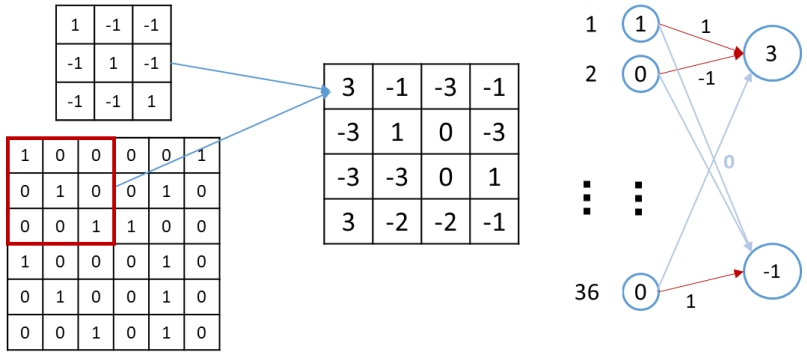
Konvoluční neuronová síť – motivace

- Konvoluční vrstva „sama“ extrahuje klíčové vlastnosti
- Konvoluční vs. plně propojená vrstva



Konvoluční neuronová síť – motivace

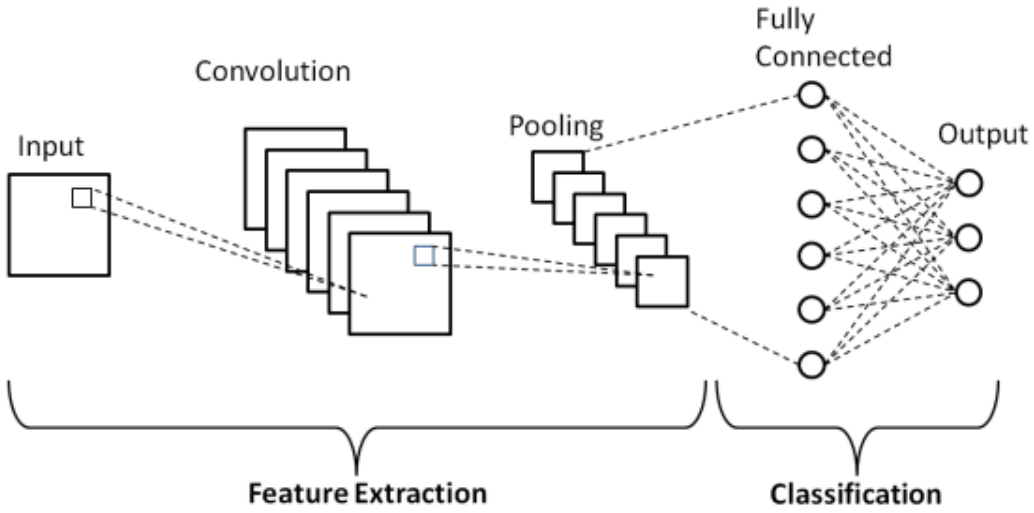
- Konvoluční vrstva není plně propojená (neuron ve skryté vrstvě – příznak je definován počtem vah odpovídající velikosti filtru)
- Dále dochází ke sdílení vah – ještě méně parametrů
- FFNN a CNN se stejným počtem neuronů má méně vah k učení



CNN – Klasifikace

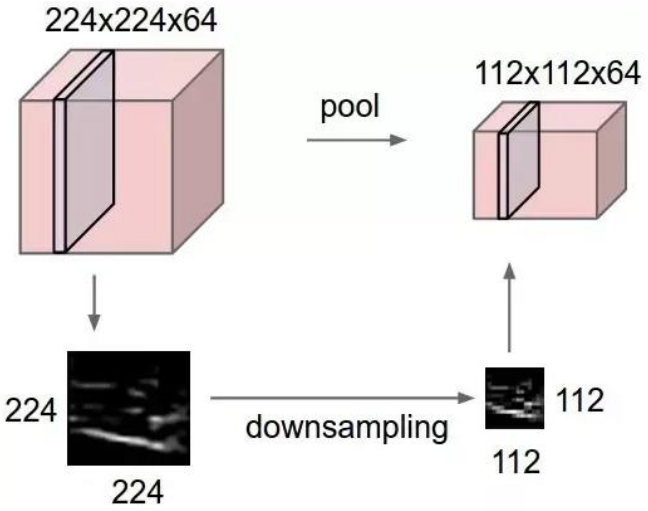
Konvoluční neuronová síť – struktura

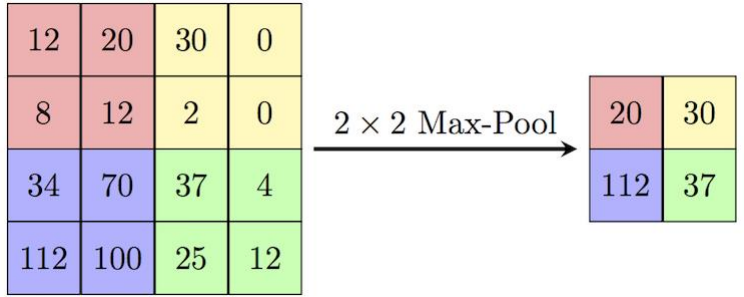
- CNN je obecně složená z konvolučních, pooling, flatten, a dense (plně propojených) vrstev



Konvoluční neuronová síť – struktura

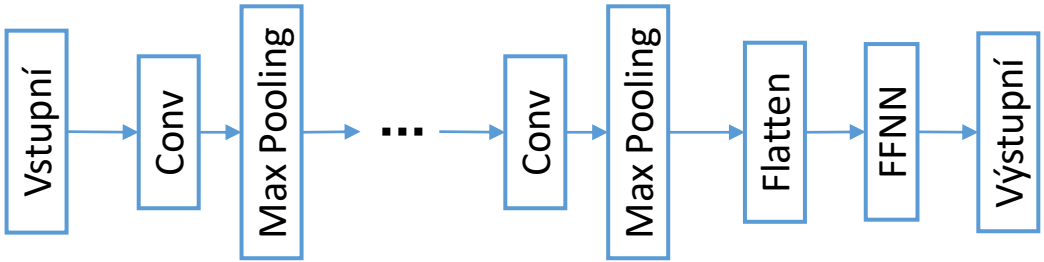
- Pooling vrstvy sjednocují skupinu vlastností a vybírají jen tu nejvíce dominantní – max pooling vybírá maximální hodnotu, snižují velikost





Konvoluční neuronová síť – topologie

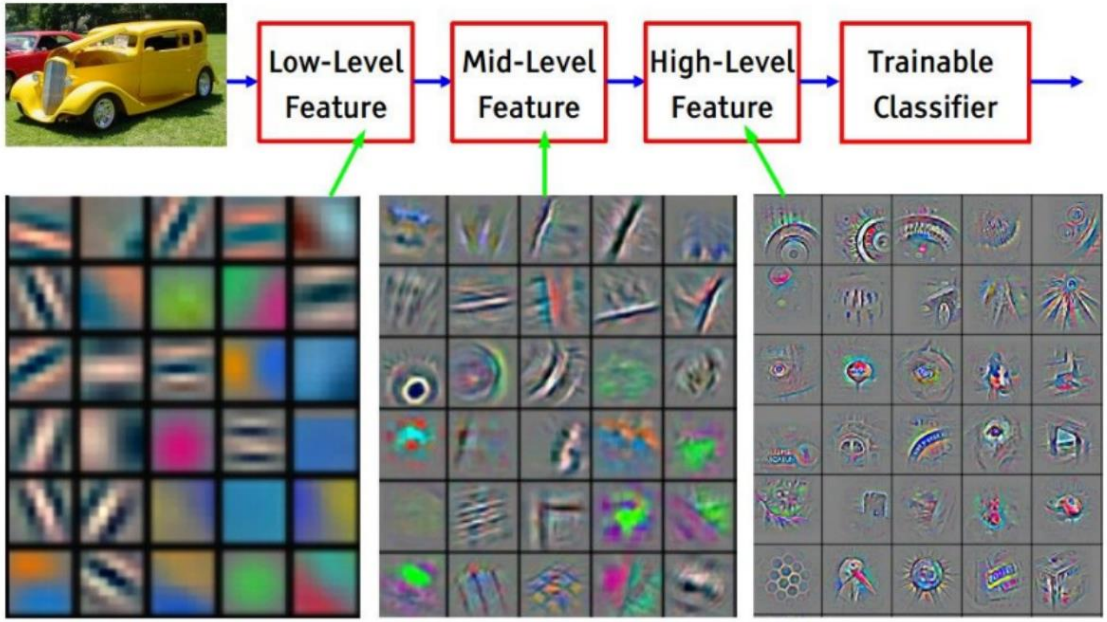
- Jednotlivé CNN se skládají z párů Conv + MaxPooling



- Samotných topologií CNN je velké množství a návrh samotné topologie je již značně komplexní
- Běžnou inženýrskou praktikou je výběr z již existujících topologií vhodných pro řešení daného problému

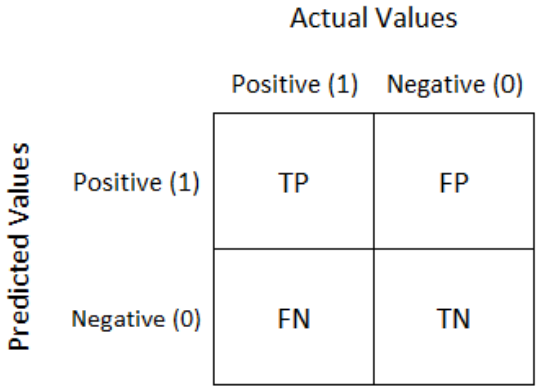
Konvoluční neuronová síť – topologie

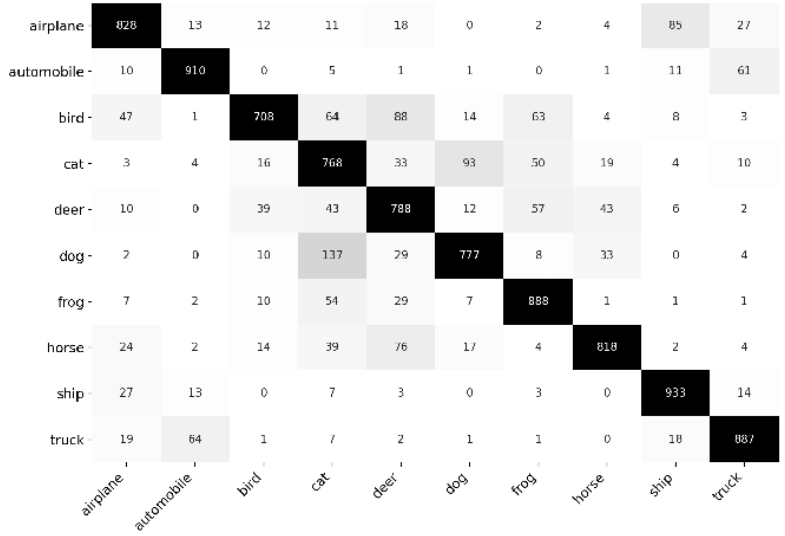
- Jednotlivé CNN se skládají z párů Conv + MaxPooling



Konvoluční neuronová síť – vyhodnocení

- K vyhodnocení klasifikačních úloh se používá Confusion Matrix (Matice záměn)



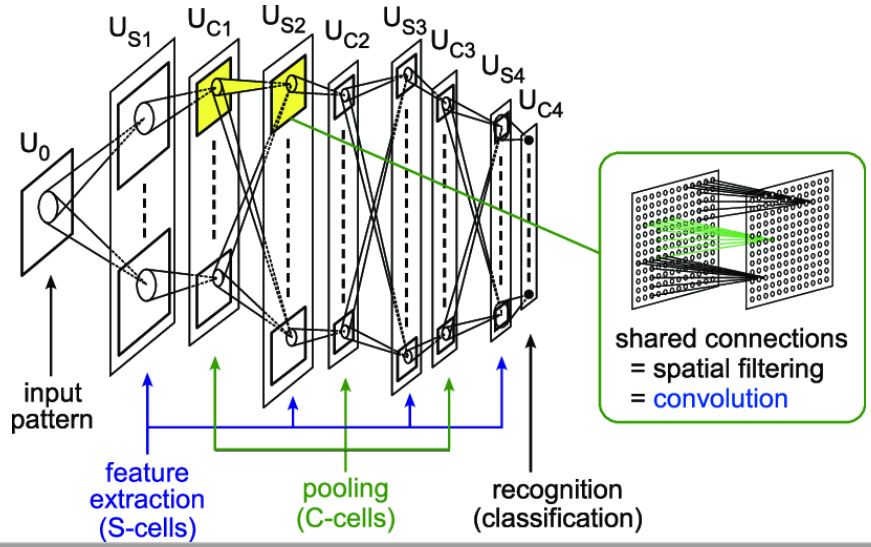


Konvoluční neuronová síť – historie

- 1959 - Simple and Complex Cells - David Hubel and Torsten Wiesel
- Zkoumali lidské zrakovém ústrojí a navrhli, že existují určité druhy buněk, které člověk využívá při rozpoznávání vzorů.
- Simple cell (S-Buňka) reaguje na hrany a pruhy určité orientace v daném perceptivním poli.
- Complex cell (C-Buňka) také reaguje na hrany a pruhy určité orientace, ale od jednoduché buňky se liší tím, že tyto hrany a pruhy mohou být na scéně posunuty a buňka bude stále reagovat.

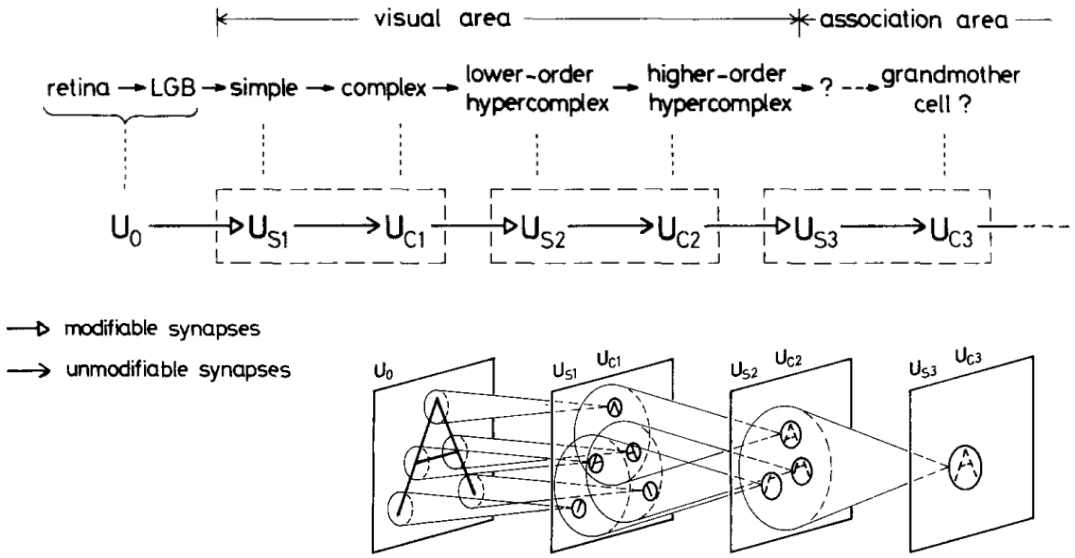
Konvoluční neuronová síť – historie

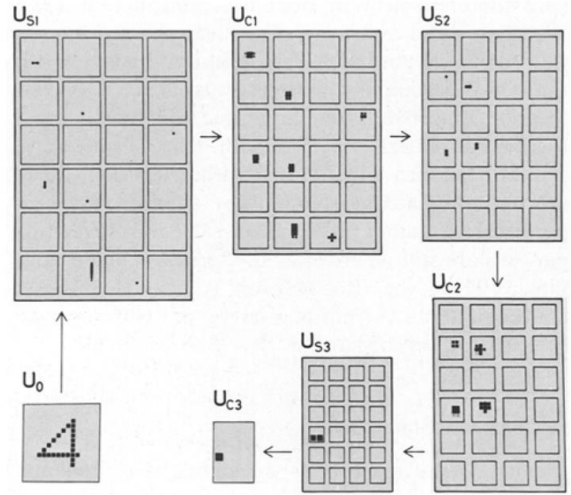
- 1980 - Neocognitron - Kunihiko Fukushima
  - Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.
  - Model zahrnuje komponenty označované jako S-buňky a C-buňky realizující matematické operace.
  - Celkovou myšlenkou je zachytit koncept "od jednoduchého ke složitému" a přeměnit jej na výpočetní model pro rozpoznávání vizuálních vzorů.



Konvoluční neuronová síť – historie

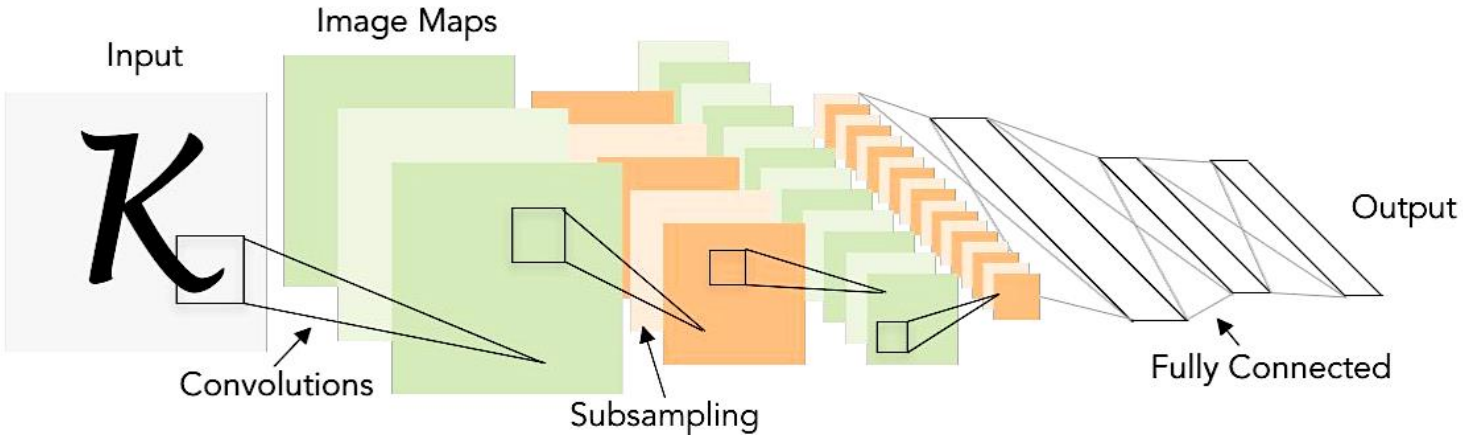
- 1980 - Neocognitron - Kunihiko Fukushima





Konvoluční neuronová síť – historie

- 1998 - LeNet - Yann LeCun
  - Gradient-Based Learning Applied to Document Recognition

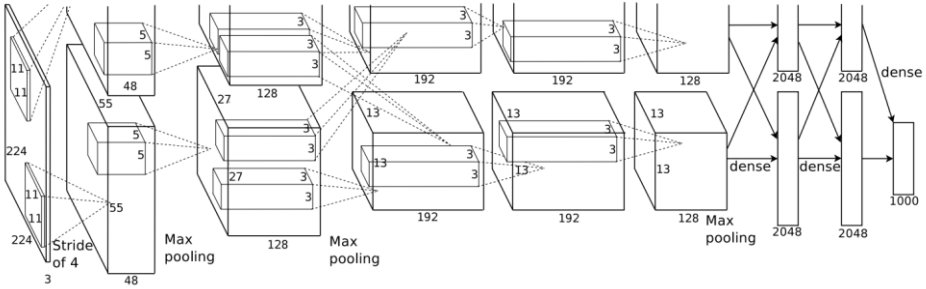


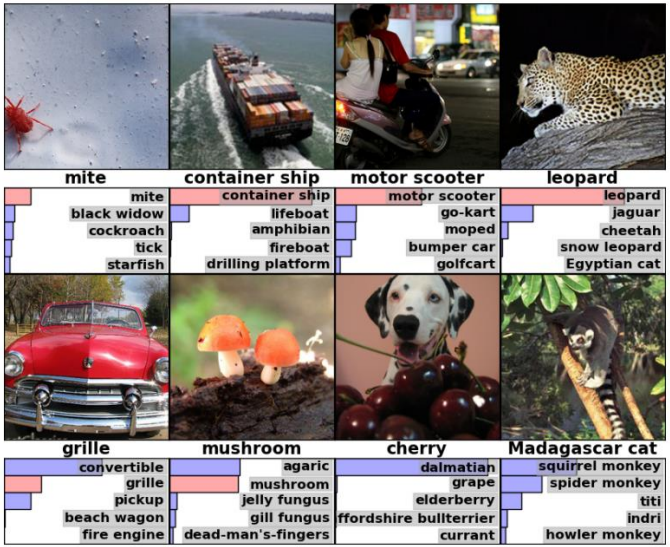
Konvoluční neuronová síť – historie

- 2009 – IMAGENET – Fei-Fei Li & Team
  - První rozsáhlý dataset (1000 tříd; 1,3 milionu obrázků) vyzívající k soutěžení v klasifikaci obrazových dat mezi výzkumníky
  - Díky soutěžení vznikají návrhy nových topologií klíčových pro rozvoj aplikace neuronových sítí pro zpracování obrazu
- 2012 – AlexNet – Alex Krizhevsky
  - ReLu aktivační funkce, velký počet filtrů v konvolučních vrstvách
  - Paralelní trénování architektury
  - Augmentace dat
  - Zavedení Dropout vrstvy – nastavení výstupu neuronů s určitou pravděpodobností na hodnotu 0. (Omezení vzájemných vztahů neuronů → neuron se nemůže spoléhat na přítomnost jiných neuronů)

Konvoluční neuronová síť – historie

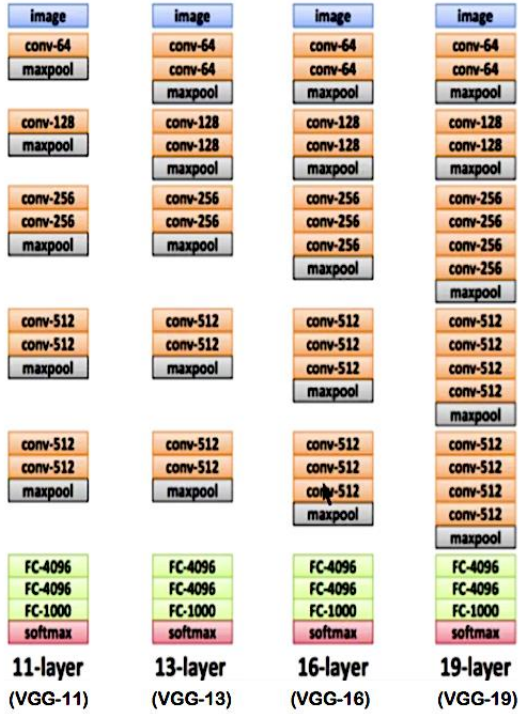
- 2012 – AlexNet – Alex Krizhevsky
  - Překrývající se části z Pooling vrstev





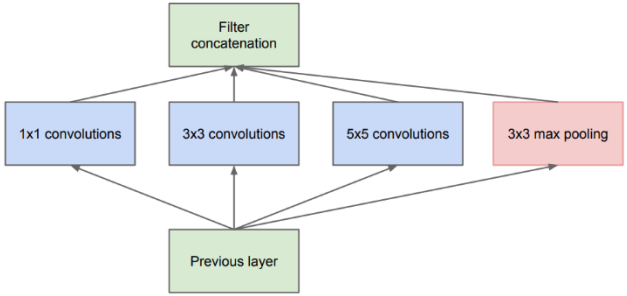
Konvoluční neuronová síť – historie

- 2013 – ZFNet – Zeiler and Fergus
  - Založený na AlexNet, ale měnící nastavení jednotlivých vrstev
- 2014 – VGGNet – Visual Geometry Group
  - Přineslo hlubší architekturu CNN, která dosahovala nižší chybovosti v soutěži na datasetu ImageNet.
  - Nová filozofie – zvětšením hloubky lze modelovat více nelinearit ve funkci → zohledňování hloubky jako kritické složky při návrhu topologie.

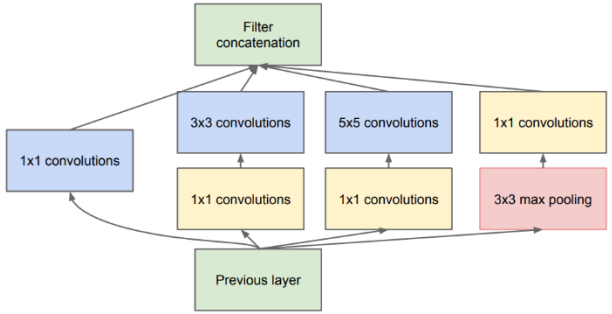


Konvoluční neuronová síť – historie

- 2014 – GoogLeNet
  - Trend zvětšování hloubky sítě, ale bez použití plně propojených vrstev (12x méně parametrů než u AlexNet, 28x méně parametrů než u VGG)
  - Zavedení „inception“ modulu - cílem je aproximovat optimální lokální struktur CNN. Umožňuje použít v jednom bloku více velikostí filtrů, místo abychom byli omezeni na jednu velikost filtru, které pak spojíme a předáme další vrstvě.

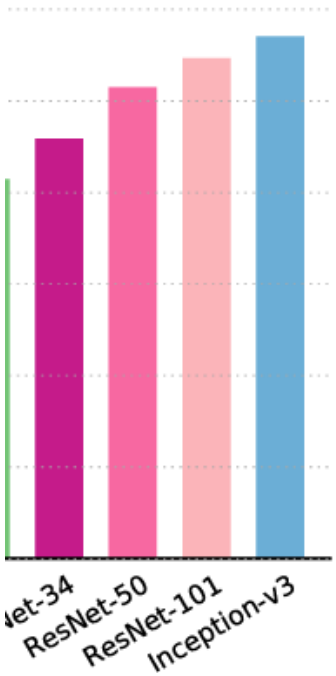


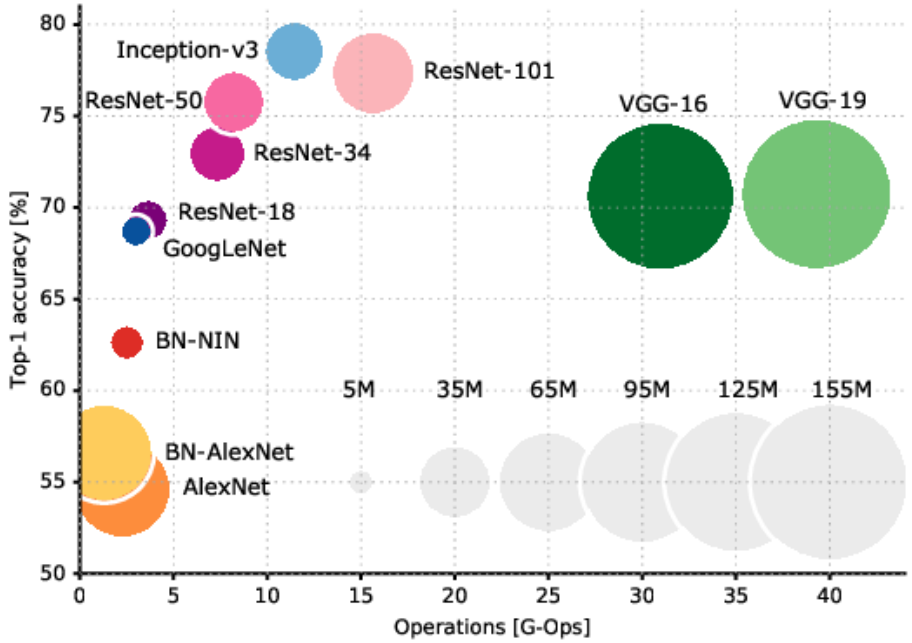
(a) Inception module, naïve version



(b) Inception module with dimension reductions

Konvoluční neuronová síť – porovnání

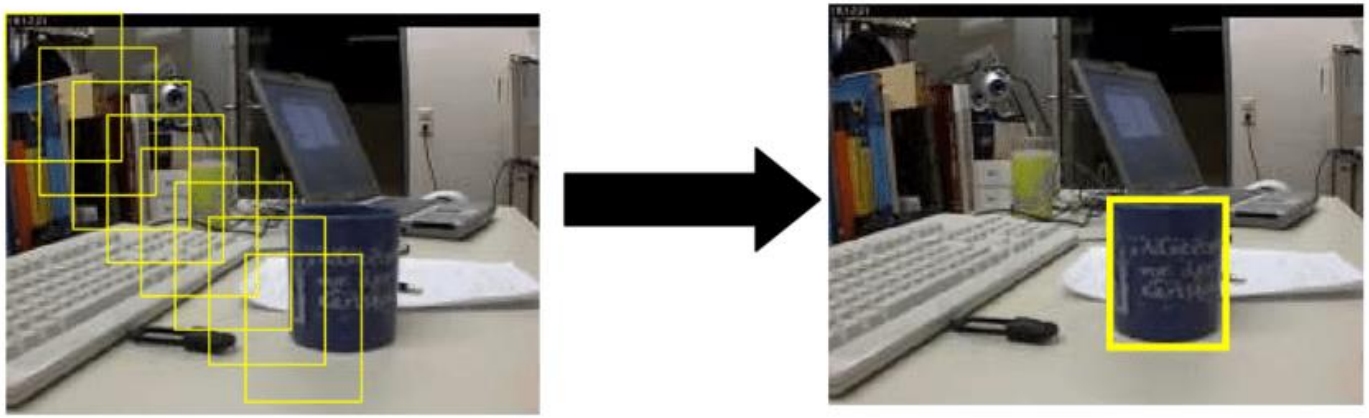




CNN – Detekce

Konvoluční neuronová síť – detekce

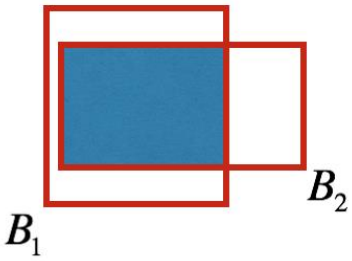
- Základní přístup pro detekci objektů v obraze je použití klasifikátoru na tzv. pohyblivé okno (slidding window)
- Uvažují se tzv. ohraničující obdélníky pro detekci pozice objektu



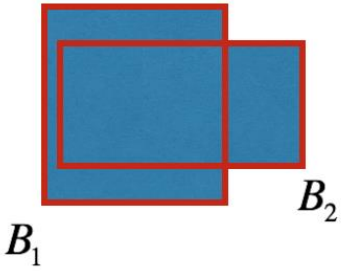
Konvoluční neuronová síť – vyhodnocení

- K vyhodnocení detekčních úloh se používá metrika IoU - Intersecion over Union

Intersection

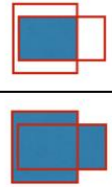


Union



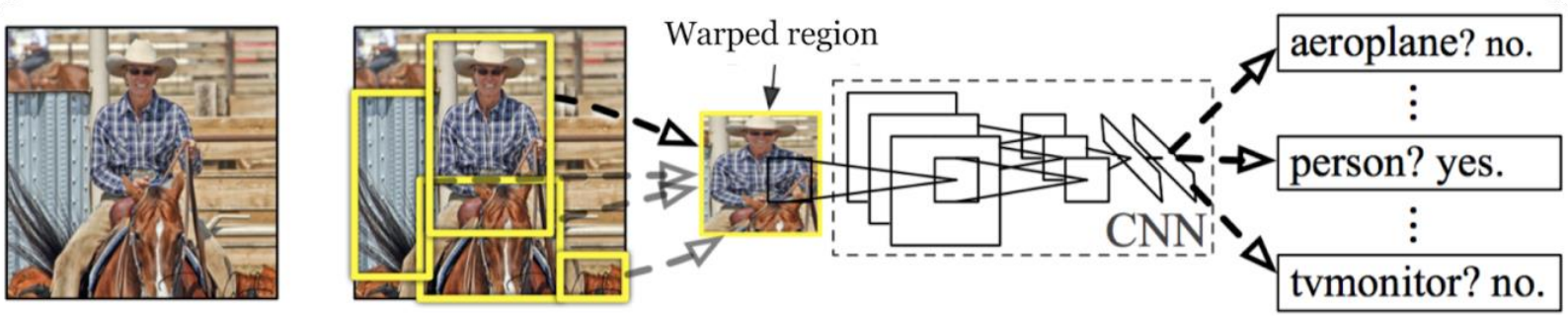
Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} =$$



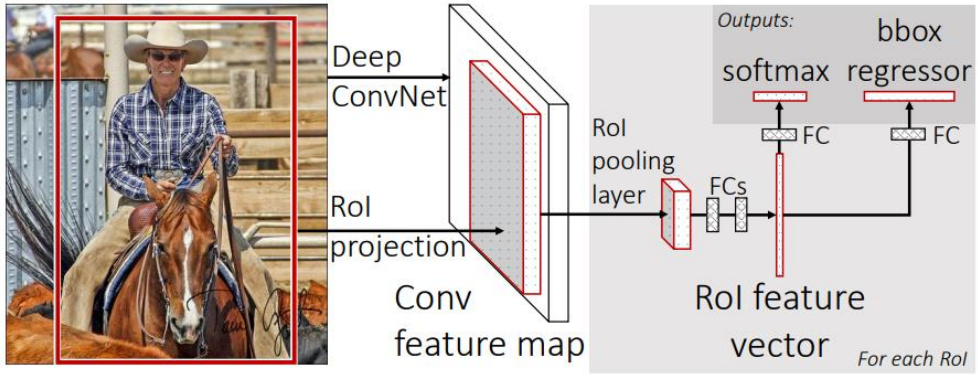
Konvoluční neuronová síť – R-CNN

- 2014 – R-CNN
  - 2 Fázový detektor (přistupuje k problému rozdělením na fáze)
  - Projde obrázek CNN a určí návrh oblasti zájmu, pak každý návrh projde sítí pro klasifikaci



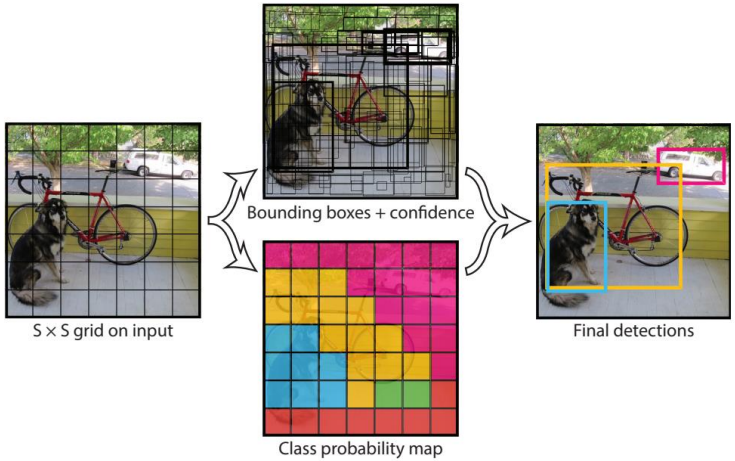
Konvoluční neuronová síť – Fast R-CNN

- 2015 – Fast R-CNN
  - Kombinuje 2 fáze → Jednou projet obrázek CNN + až potom řešit jednotlivé regiony
  - Postup detekce:
    - Celý vstupní obrázek projde jednou hlubokou CNN (např. VGG-16). Výstup = Feature mapa (popis obrázku).
    - Na feature mapě se aplikují tzv. Region Proposal Regions (RoIs) — oblasti, kde by mohly být objekty
    - Pro každý RoI: Použije se speciální vrstva → RoI Pooling: Převzorkuje fixní velikosti (např. 7×7). (Pro FFNN)
    - Výstup z RoI Poolingu jde do plně propojené vrstvy (klasifikační head).



Konvoluční neuronová síť – YOLO

- 2016 – YOLOv1
  - You Only Look Once – Přináší nový přístup kombinující fáze z 2 fázových detektorů
  - Obraz se rozdělí na 7×7 (v originále) buněk a každá buňka předpovídá několik bounding boxů včetně pravděpodobnosti, že tam objekt je, a klasifikaci do tříd.



Konvoluční neuronová síť – YOLO

- 2018 – YOLOv3
  - Navazuje na v2, s vylepšením díky využití tzv. „anchor boxů“ místo bounding boxů

    Bounding box
    - Výsledný obdélník kolem objektu. Vzniká po výpočtu/predikci modelu.

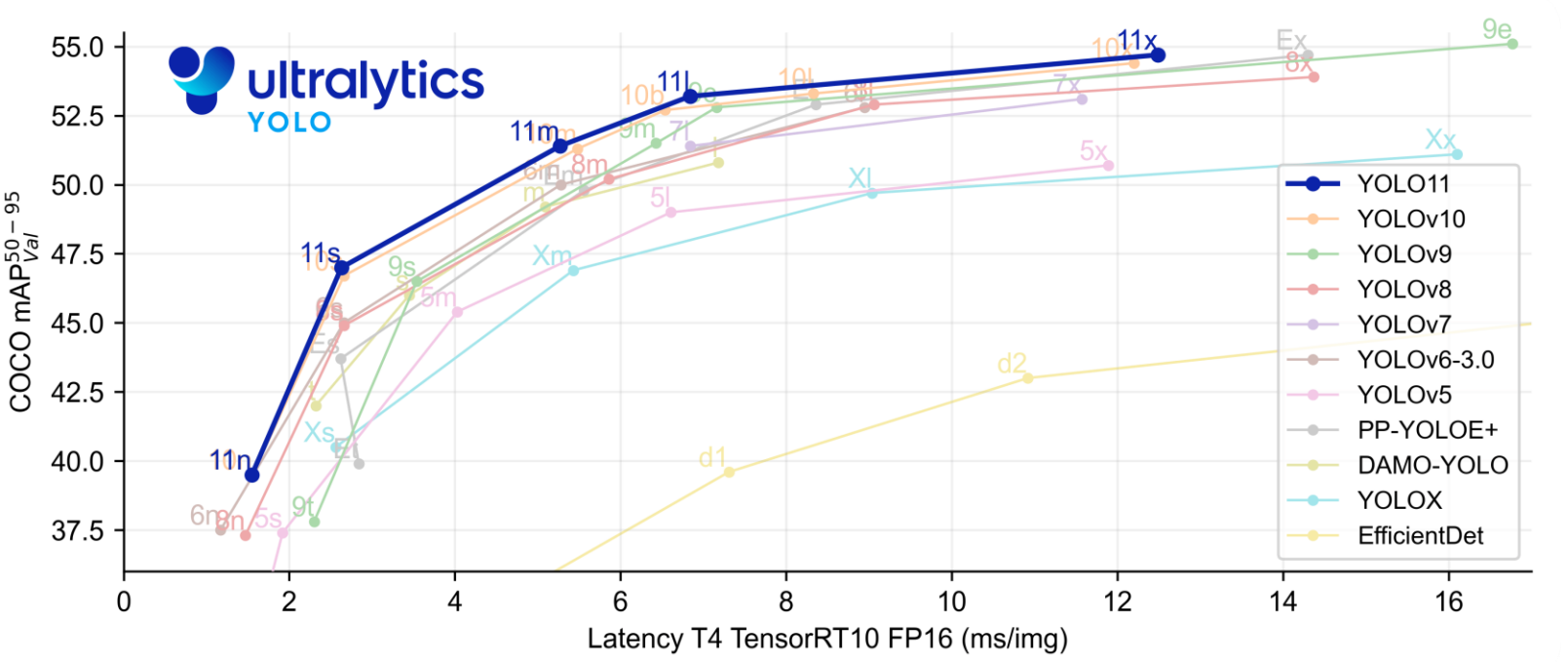
    Anchor Box
    - Předdefinované tvary boxů, které model používá jako „startovní šablony“ pro predikci bounding boxů.

  - Využití více úrovní map příznaků
  - Použití „multi-scale“ přístupu (tři úrovně detekce) → lepší detekce menších obj.
  - Počátek zavádění tzv. „páteřních“ sítí (Backbone)
    - Část modelu, která extrahuje feature mapy.
    - Typicky předtrénovaná na ImageNet (klasifikace).
    - Detektor pak přidává jen „hlavičku“ (head) pro detekci objektů.

Konvoluční neuronová síť – YOLO

- YOLO, YOLOv8
  - Rozšíření zaměřující se zejména na zlepšení přesnosti a udržení real-time inference
  - Přechod pod Open-source platformu Ultralytics
  - Různé implementace + snadné vlastní implementace (PyTorch, připravené skripty)
  - Různé velikosti sítě (U, S, M, L, X)
  - Testování a optimalizace backbone
  - Pokročilá augmentace pro zlepšení schopnosti generalizovat

Konvoluční neuronová síť – porovnání

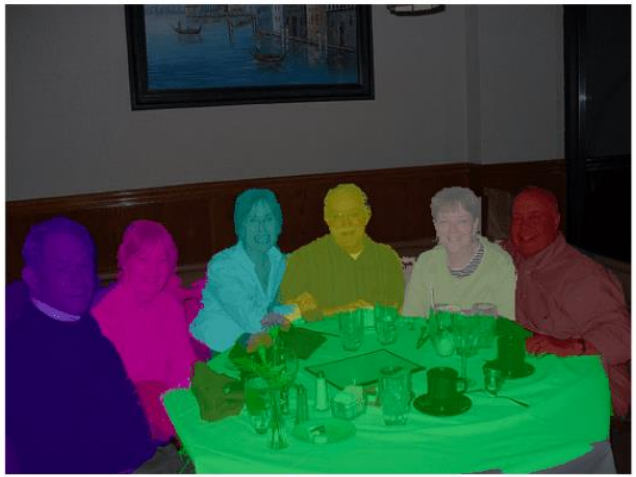


CNN – Segmentace

Segmentace obrazových dat

- Pokročilejší metoda, přiřazující třídu (label) jednotlivým pixelům
- Instanční vs Sémantická

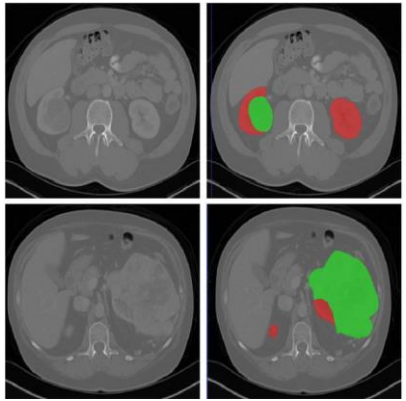
Semantic Segmentation

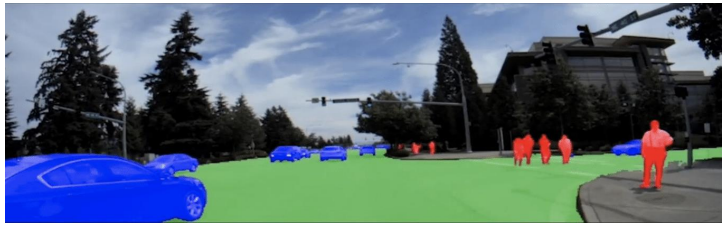


Instance Segmentation

Segmentace obrazových dat – užití

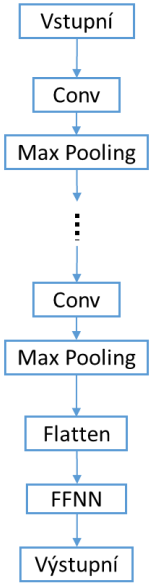
- Medical imaging, Self-driving cars, Remote Sensing

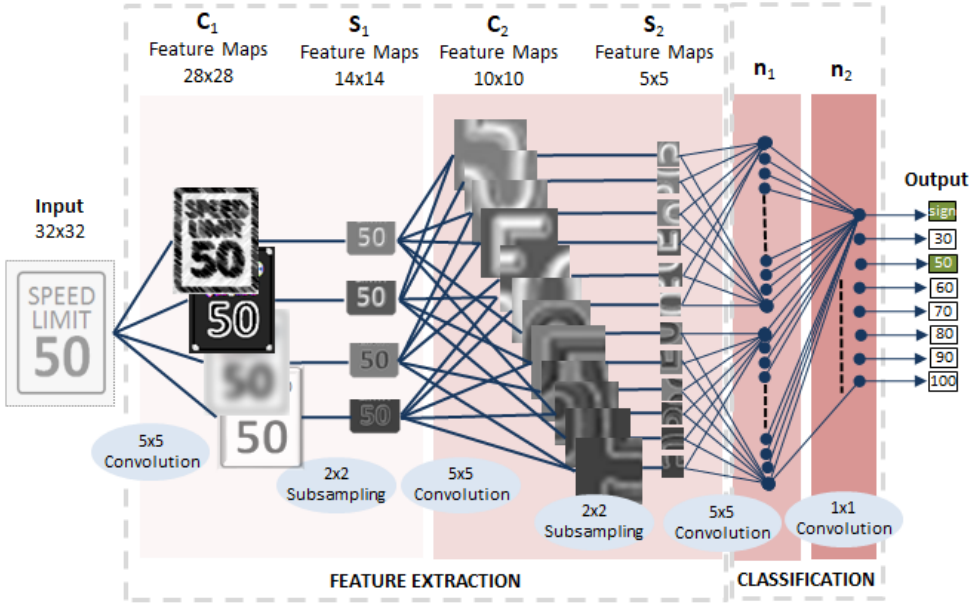




Segmentace pomocí CNN

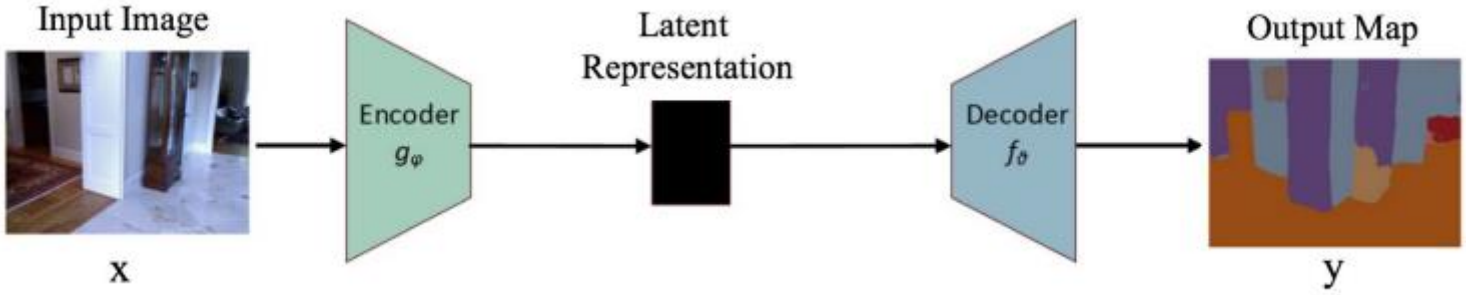
- Konvoluční vrstvy produkují výstupní data – zpracované obrázky





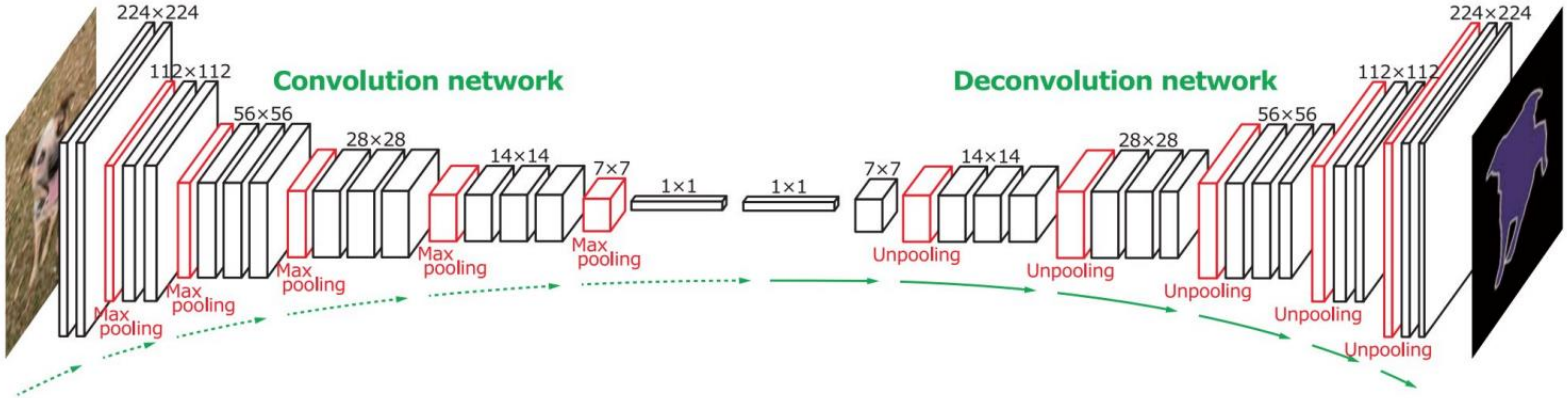
Segmentace pomocí CNN

- U segmentace vyžadujeme na výstupu označený obrázek.
- Modely založené na principu Enkodér-Dekodér nebo Auto-Enkodér.
- Jsou založené na vlastnosti konvolučních sítí kódovat vstupní informaci (obrázek).



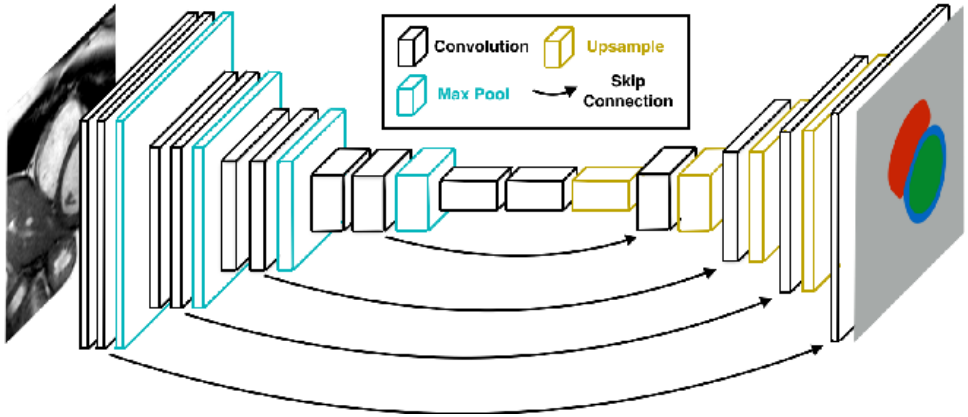
Segmentace pomocí CNN

- Enkodér-Dekodér schéma – postupné kódování vlastností obrázku a následné jeho postupné dekódování pro vygenerování segmentovaného obrázku



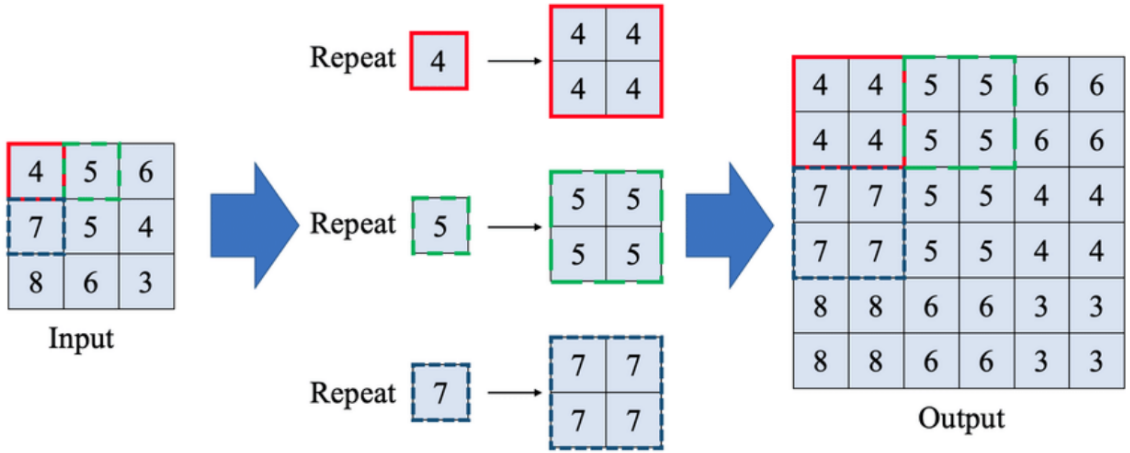
Segmentace pomocí CNN

- Skip spojení – přenášejí určité klíčové vlastnosti mezi enkodérem a dekodérem v dané hloubce



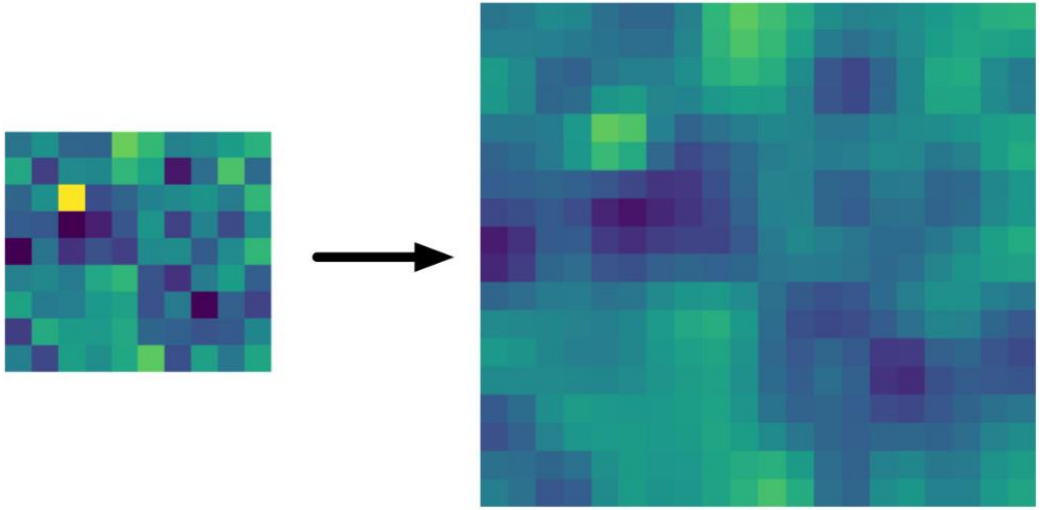
Segmentace pomocí CNN

- UpSampling vrstva – zvětšuje původní rozměr vstupu
- Metody: **replikace**, průměrování, Unpooling



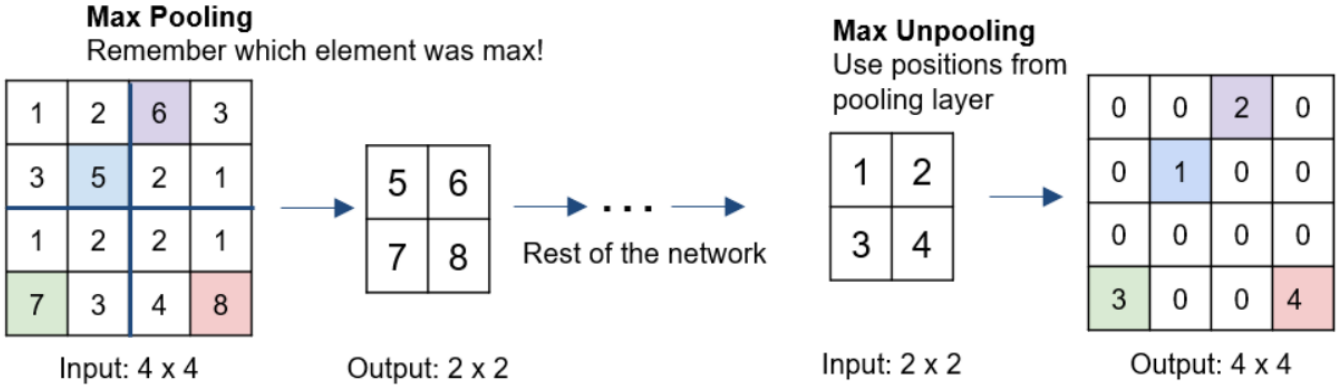
Segmentace pomocí CNN

- UpSampling vrstva – zvětšuje původní rozměr vstupu
- Metody: replikace, **průměrování**, Unpooling



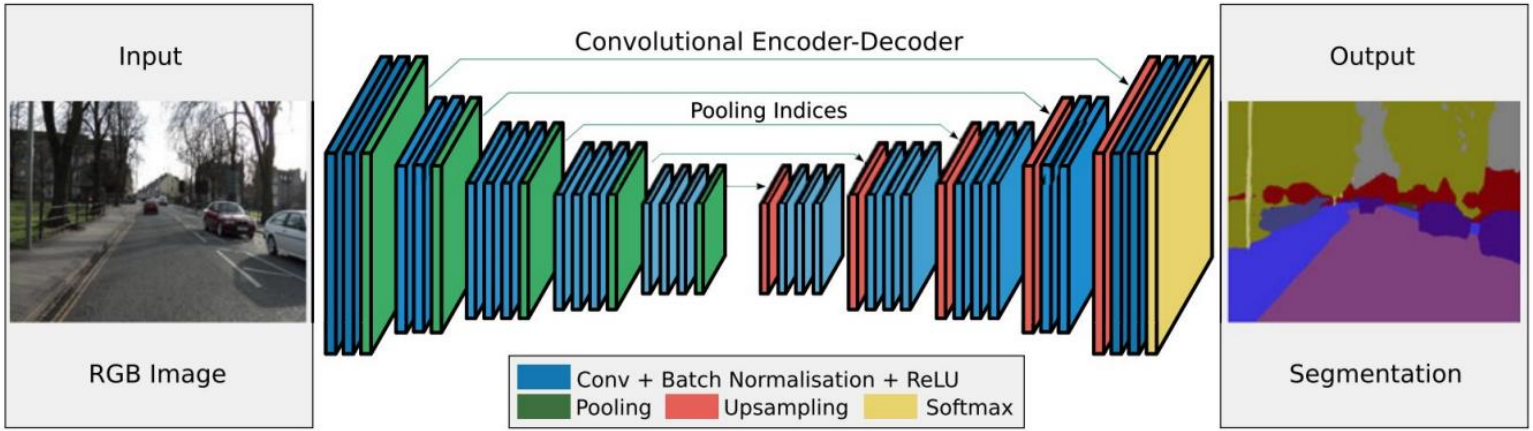
Segmentace pomocí CNN

- UpSampling vrstva – zvětšuje původní rozměr vstupu
- Metody: replikace, průměrování, **Unpooling**



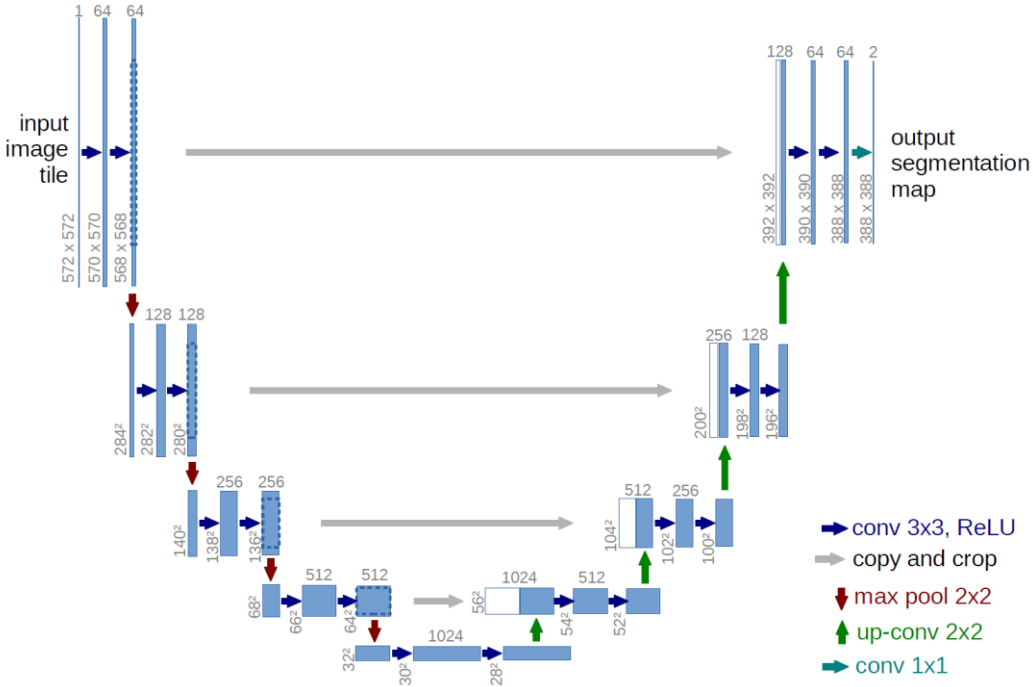
Segmentace pomocí CNN – architektury

- SegNet



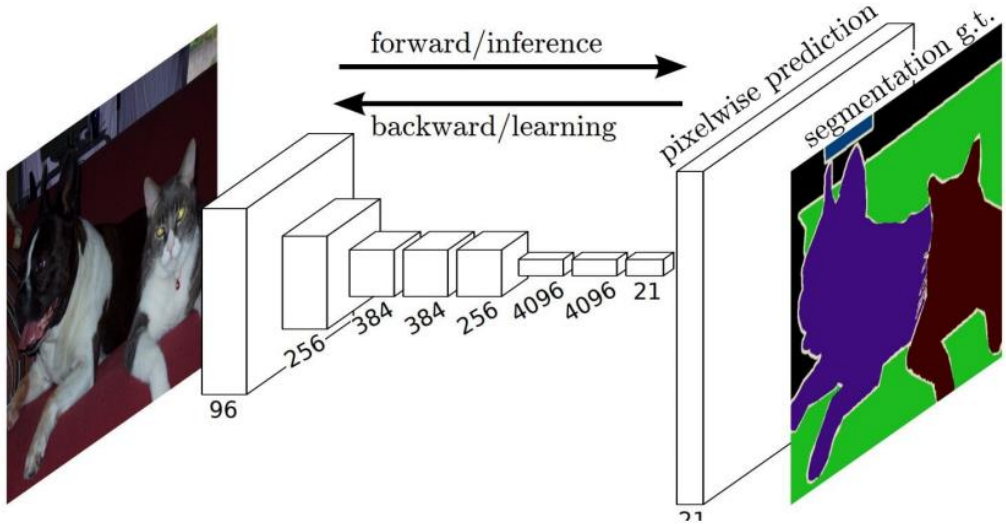
Segmentace pomocí CNN – architektury

- U-Net



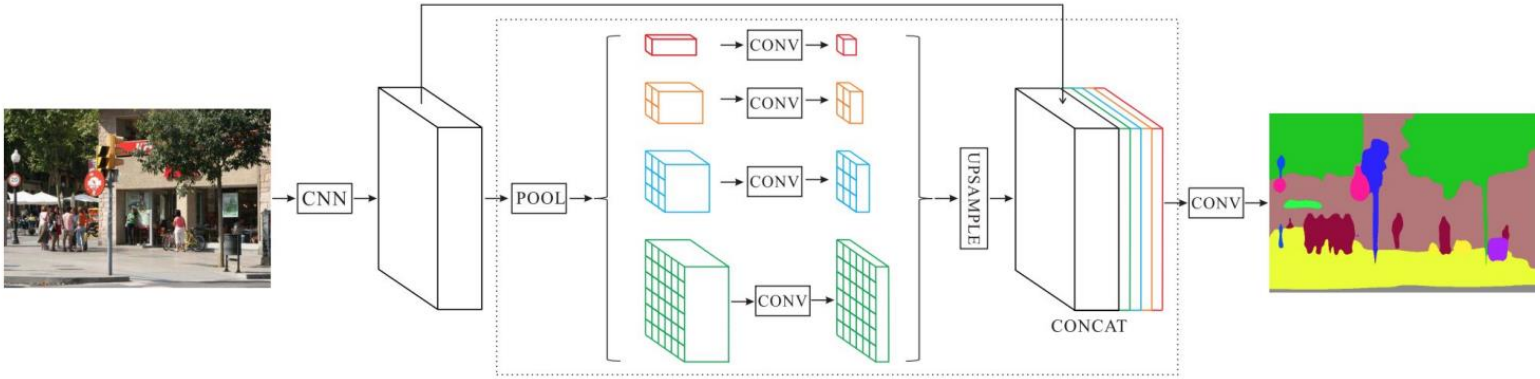
Segmentace pomocí CNN – architektury

- FCN – Fully Convolutional Network



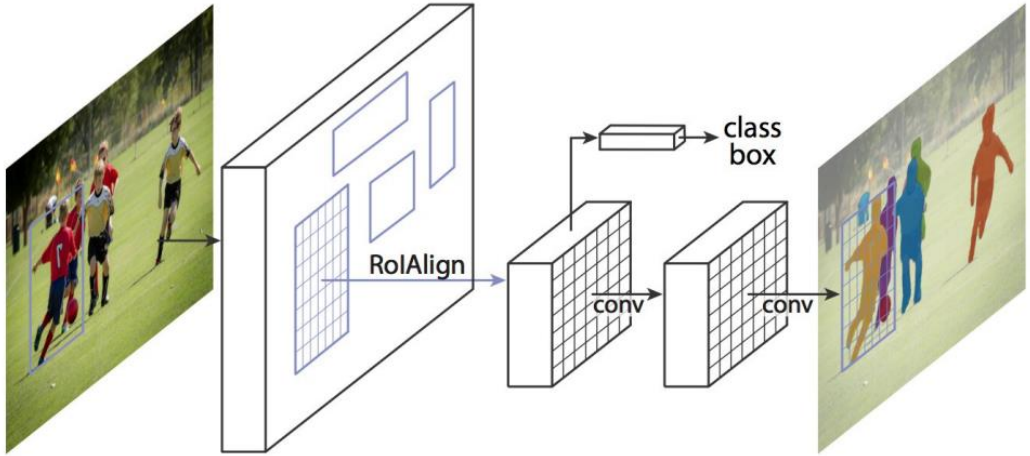
Segmentace pomocí CNN – architektury

- Pyramid Network modely



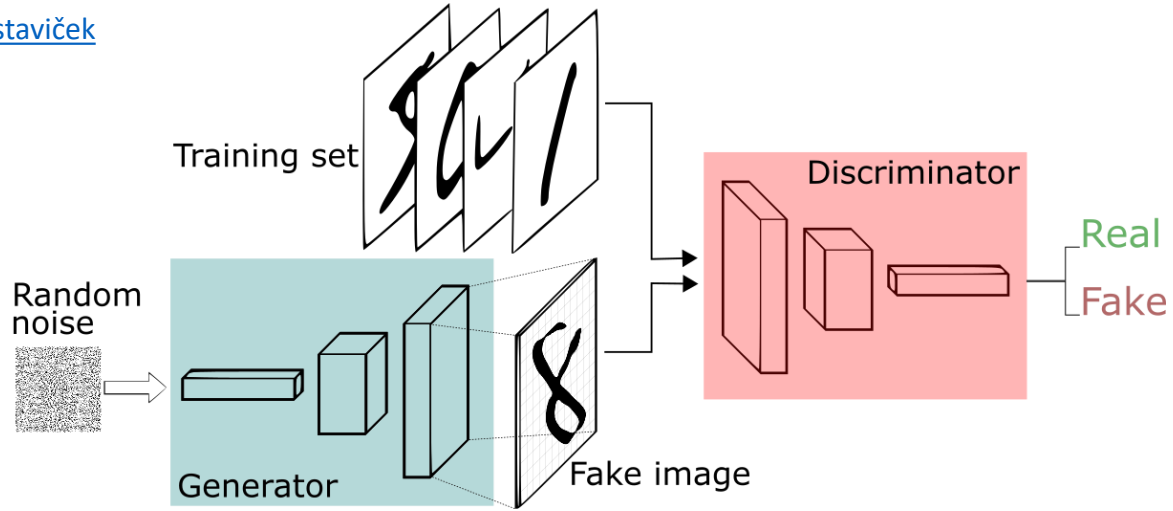
Segmentace pomocí CNN – architektury

- Mask R-CNN
  - Objekty jsou klasifikovány a lokalizovány pomocí ohraničujícího boxu a sémantické segmentace, která klasifikuje každý pixel do daných kategorií. Každá oblast zájmu dostane segmentační masku.



Segmentace pomocí CNN – architektury

- Segmentační úloha vede k zisku nového obrázku, který nemusí zachycovat pouhé přiřazení do třídy, ale může vytvářet nový obrázek
- GANs - Generative Adversarial Networks
  - Generování kreslených postaviček
  - Stárnutí obličeje
  - Vybarvování obrázků



Datasety

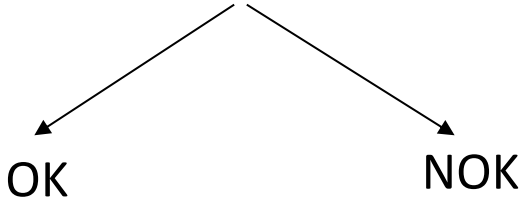
- Datasety jsou nedílnou součástí při práci s neuronovými sítěmi.
- V obrazovém zpracování mohou vstupní obrázky nabývat 2D, 2.5D (RGB+D), 3D rozměrů.
- Podle dané úlohy existují různé datasety, na kterých se provádí testování vlastních topologií vzhledem k ostatním. Přehledné shrnutí dostupných datasetů bylo např. popsáno v článku z 2020.
- Kromě vytvořených „obecných“ datasetů je ale často pro specifickou úlohu vytvořit datasety vlastní.
  - Vytvoření obrázků trénovací a testovací množiny.
  - Ruční označení dat (zisk tzv. ground truth) - bounding boxy, labelling pro segmentaci (přehled 2022), vlastní označování (vlastní aplikace)

Augmentace

- Vlastní datasety jsou často vytvořeny z několika stovek, tisíců obrázků, které nepokrývají dostačující část pro správné natrénování neuronové sítě.
- Pomocí augmentace můžeme dataset uměle rozšířit (původní obrázky orotovat, ztmavit, posunout, …)
- Použití více dat pro trénování vede k lepší generalizaci sítě.
- Tensorflow Data Generator

Průmyslová aplikace

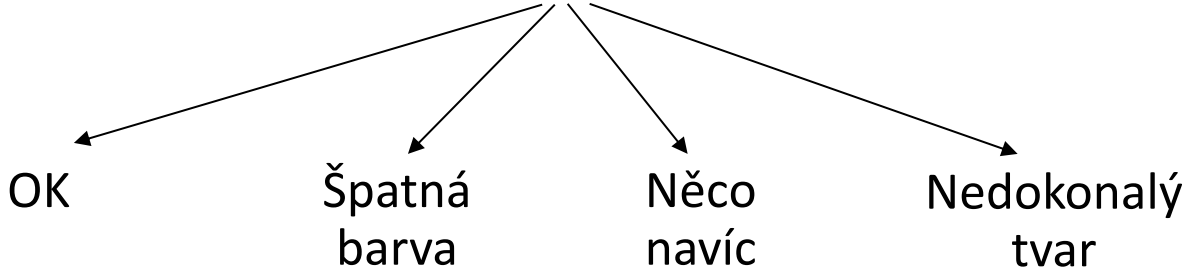
Úkol – klasifikace výrobků



- Pečivo; kontrola kvality

Úkol – klasifikace výrobků

- Pečivo; kontrola kvality



Úkol – klasifikace výrobků

OK

Špatná barva

Něco navíc

Nedokonalý tvar

Úkol – klasifikace výrobků

- Podmínky:
  - Přesnost (Accuracy, Precision, Recall)
  - Výpočetní náročnost
  - Na jakém datasetu?

Úkol – klasifikace výrobků

- Dataset:
  - OK – 1896 vzorků
  - Špatná barva – 512 vzorků
  - Něco navíc – 632 vzorků
  - Nedokonalý tvar – 1860 vzorků

Úkol – klasifikace výrobků

- Pro trénovaní je třeba rozdělit (70:15:15):
  - OK – 1327 Train, 284 Val, 285 Test
  - Špatná barva – 358 Train, 76 Val, 78 Test
  - Něco navíc – 442 Train, 94 Val, 96 Test
  - Nedokonalý tvar – 1302 Train, 279 Val, 279 Test

Úkol – klasifikace výrobků

- Návrh neuronového modelu
  - Vstupní velikost obrázků
  - Normalizace dat
  - Použití konvolučních a pooling vrstev
  - Dropout vrstva
  - Plně propojené vrstvy
  - Výstupní vrstva
  - Chybová funkce a trénovací algoritmus

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Green Deal UPCE**, reg. č. NPO_UPCE_MSMT-2142/2024-4.

NÁRODNÍ PLÁN OBNOVY
Financováno Evropskou unií
NextGenerationEU