Neuronové sítě 1 - Jednovrstvá a vícevrstvá neuronová síť

18NES1 - 9. a 10. hodina, LS 2024/25

Zuzana Petříčková

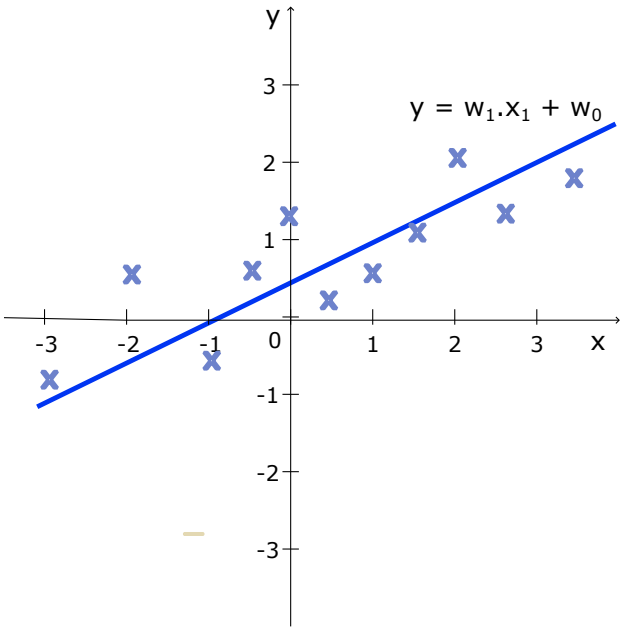
17-20. března 2025

Co jsme probírali minule

1. **Nejznámější spojité přenosové funkce pro perceptron**

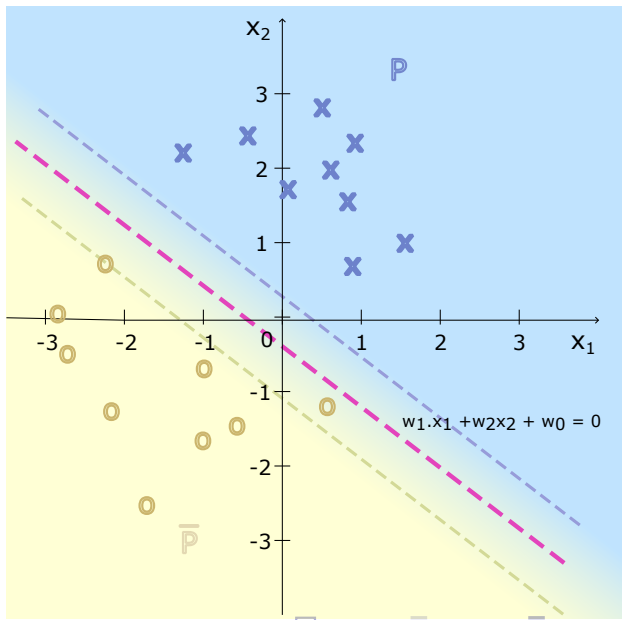
- lineární (identita)

Úloha lineární regrese



- sigmoida, hyperbolický tangens

Úloha lineární klasifikace



Co jsme probírali minule

2. **Gradientní metoda pro obecný perceptoron se spojitou a diferencovatelnou přenosovou funkcí**
   1. Inicializuj váhy malými náhodnými reálnými hodnotami ....
      $\vec{w}(0) = (w_0, w_1, ..., w_n)^T$
      Inicializuj parametr učení α_0.... $1 > \alpha_0 > 0$
   2. Předlož další trénovací vzor $(\vec{x}_t, d_t)$ a spočti potenciál a skutečný výstup neuronu:
      $\xi_t = \vec{x}_t \vec{w}$
      $y_t = f(\xi_t)$
   3. Adaptuj váhy (proti směru gradientu chybové funkce):

$$\vec{w}(t+1) = \vec{w}(t) - \alpha \frac{\partial E_p}{\partial w_i} = \vec{w}(t) + \alpha_t f'(\xi_t)(d_t - y_t)\vec{x}_t^T$$

      (pro chybovou funkci SSE)
   4. Případně aktualizuj parametr učení : $\alpha_t \to \alpha_{t+1}$
   5. Pokud není konec, přejdi ke kroku 2.

Co jsme probírali minule

2. **Gradientní metoda pro obecný perceptoron se spojitou a diferencovatelnou přenosovou funkcí**

Chybová funkce SSE:

$$E(\vec{w}) = \frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2 = \frac{1}{2}\sum_{p=1}^{N}\left(d_p - f(\sum_{i=0}^{n} w_i.x_{pi})\right)^2 = \sum_{p=1}^{N} E_p(\vec{w})$$

Parciální derivace:

$$\frac{\partial E_p}{\partial w_i} = \frac{\partial E_p}{\partial y_p}\frac{\partial y_p}{\partial \xi_p}\frac{\partial \xi_p}{\partial w_i} = -(d_p - y_p)f'(\xi_p)x_{pi}$$

Adaptační pravidlo (po předložení p-tého vzoru v čase t)

$$w_i(t+1) = w_i(t) - \alpha\frac{\partial E_p}{\partial w_i} = w_i(t) + \alpha f'(\xi_p)(d_p - y_p)x_{pi}$$

pro vektor vah:

$$\vec{w}(t+1) = \vec{w}(t) - \alpha\nabla E_p(\vec{w}) = \vec{w}(t) + \alpha(d_p - y_p)f'(\xi_p)\vec{x}_p^T$$

Co jsme probírali minule

Chybová funkce Cross-entropy

- Velmi výhodná pro lineární klasifikaci v kombinaci s přenosovými funkcemi sigmoida nebo tanh.
- Penalizuje nízké pravděpodobnosti správné třídy.
- Např. pro sigmoidu:

$$E = -\sum_p \left(d_p \log y_p + (1 - d_p) \log(1 - y_p)\right)$$

- gradient bude roven: $\frac{\partial E_p}{\partial y_p} = -\frac{d_p}{y_p} + \frac{(1-d_p)}{(1-y_p)}$
- po dosazení do vzorečku se některé členy navzájem výhodně vykrátí a platí:

$$\frac{\partial E_p}{\partial w_i} = \frac{\partial E_p}{\partial y_p}\frac{\partial y_p}{\partial \xi_p}\frac{\partial \xi_p}{\partial w_i} = \left(\frac{1 - d_p}{1 - y_p} - \frac{d_p}{y_p}\right) y_p(1-y_p)x_{pi} = (y_p - d_p)x_{pi}$$

$\rightarrow$ efektivnější učení a snížení rizika saturace neuronů

Co jsme probírali minule

3. **Předzpracování kategoriálních dat**

Ordinal (label) encoding

- Pro kategorie s pořadím, např. *nízký, střední, vysoký* a pro právě dvě kategorie, např. *levý, pravý* hodnoty převedeme na čísla a případně normalizujeme:
  - *Nízký* = -1, *Střední* = 0, *Vysoký* = 1
  - *Levý* = -1, *Pravý* = 1

One-hot encoding

- Pro nenávislé kategorie, např. barva auta, zvíře. Převedeme na binární reprezentaci (z jednoho příznaku vznikne tolik sloupců, kolik je kategorií) a případně normalizujeme:
  - *Červená* $\rightarrow [1, -1, -1]$, *Modrá* $\rightarrow [-1, 1, -1]$, *Zelená* $\rightarrow [-1, -1, 1]$

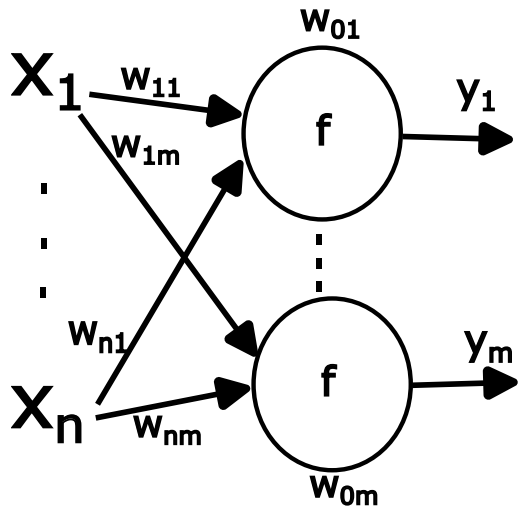
Embeddings

- pro složitější vztahy mezi kategoriálními hodnotami, např. reprezentace slov a vět v přirozeném jazyce

Dnešní hodina

1. Neuronová síť tvořená jednou vrstvou neuronů - dokončení
2. Neuronová síť tvořená více vrstvami neuronů - úvod

Neuronová síť tvořená jednou vrstvou neuronů



1. Neurony s lineární přenosovou funkcí
   → **mnohorozměrná lineární regrese**
   - lineární neuronová síť
2. Neurony s logistickou nebo tanh přenosovou funkcí
   → **lineární klasifikace do více tříd (úloha rozpoznávání vzorů)**
   - jednovrstvý perceptron

Neuronová síť tvořená jednou vrstvou neuronů

Motivační příklad: klasifikace do více tříd a kategoriální

	Velikost	**Srst**	**Mluví?**	**Třída**
hodnoty	Malá	Krátká	Ne	Kočka
	Velká	Dlouhá	Ne	Pes
	Malá	Žádná	Ano	Papoušek
	Střední	Krátká	Ne	Kočka
	...			...

Velikost	**Srst**	**Mluví?**	**Kočka**	**Pes**	**Papoušek**	**Kapr**
-1	0	-1	1	-1	-1	-1
1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1
0	0	-1	1	-1	-1	-1
...	...	...	...		...	

Ukázka: categorical_values.ipynb

Neuronová síť tvořená jednou vrstvou neuronů

Příklad klasifikace do více tříd a kategoriální hodnoty

Velikost	Srst	Mluví?	Kočka	Pes	Papoušek	Kapr
-1	0	-1	1	-1	-1	-1
1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1
0	0	-1	1	-1	-1	-1
...	...	...	...		...	

Jak úlohu naučíme?

1. pro každou kategorii naučíme jeden neuron (jeden po druhém) a slepíme výsledky ...

Velikost	Srst	Mluví?	Kočka
-1	0	-1	1
0	1	-1	-1
-1	-1	1	-1
0	0	-1	1
...			...

Neuronová síť tvořená jednou vrstvou neuronů

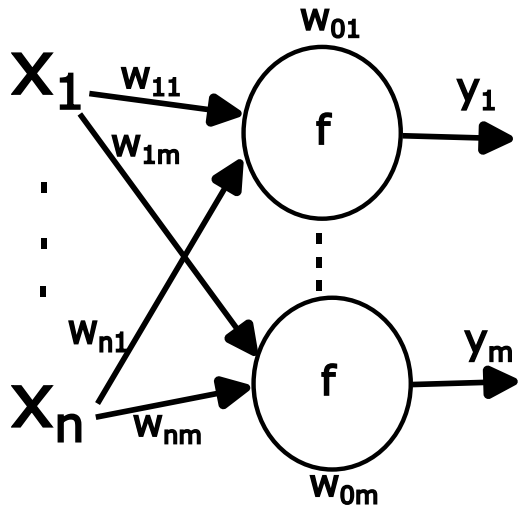
Příklad klasifikace do více tříd a kategoriální hodnoty

Velikost	Srst	Mluví?	Kočka	Pes	Papoušek	Kapr
-1	0	-1	1	-1	-1	-1
1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	-1
0	0	-1	1	-1	-1	-1
...	...	...	...		...	

Jak úlohu naučíme?

2. **lépe:** sestrojíme neuronovou síť s jednou vrstvou neuronů a naučíme ji najednou

Neuronová síť tvořená jednou vrstvou neuronů

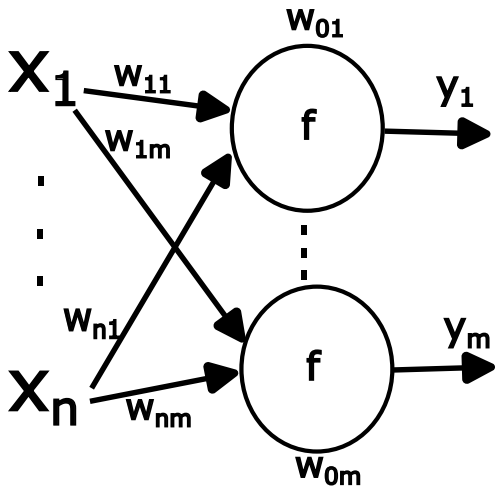


- Model je reprezentován maticí vah

$$W = \begin{pmatrix} w_{01} & w_{02} & \dots & w_{0m} \\ \dots & \dots & & \dots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{pmatrix}$$

  každému neuronu odpovídá jeden sloupec matice
- mají-li jednotlivé neurony přenosovou funkci $f : R \to R$, definujeme $f(\vec{\xi}) = (f(\xi_1), ..., f(\xi_N))^T$

Neuronová síť tvořená jednou vrstvou neuronů



$$W = \begin{pmatrix} w_{01} & w_{02} & \dots & w_{0m} \\ \dots & \dots & & \dots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{pmatrix}$$

perceptron_layer.ipynb

- Výstup modelu:

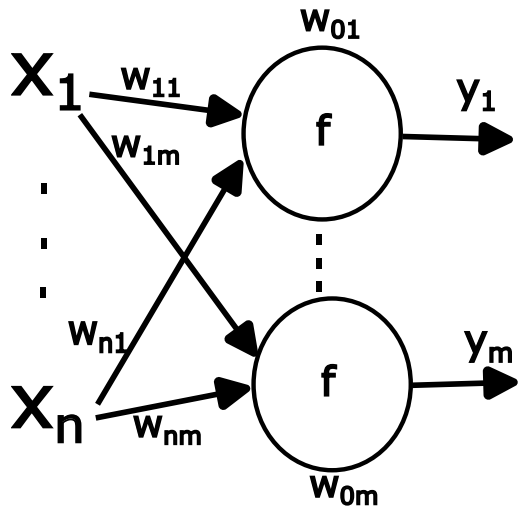
$$\vec{y} = f(\vec{\xi}) = f(\vec{x}W)$$

- Trénovací množina je ve tvaru $T = (X, D)$

$x_{10} = 1$	x_{11}	...	x_{1n}	d_{11}	...	d_{1m}
...	...		...	...		...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_{N1}	...	d_{Nm}

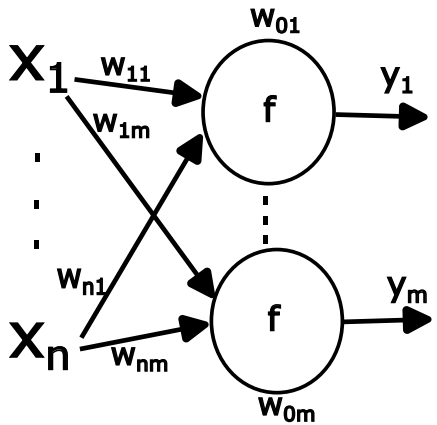
$$Y = f(\Xi) = f(XW)$$

Neuronová síť tvořená jednou vrstvou neuronů



1. Neurony s lineární přenosovou funkcí
   → **mnohorozměrná lineární regrese**
   - lineární neuronová síť
2. Neurony s logistickou nebo tanh přenosovou funkcí
   → **lineární klasifikace do více tříd (úloha rozpoznávání vzorů)**
   - jednovrstvý perceptron

Lineární neuronová síť



- tvořena jednou vrstvou lineárních neuronů (více vrstev by nepřineslo žádný benefit - na rozmyšlenou)
- mnohorozměrná lineární regrese
- výstup modelu:

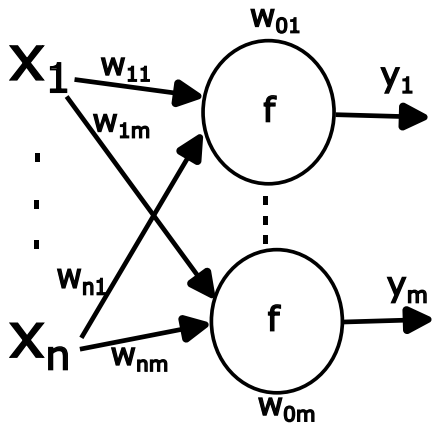
$$Y = XW$$

Učení metodou LSQ

$$W = (X^T X)^{-1} X^T D$$

Učení gradientní metodou

Jednovrstvá neuronová síť se spojitou přenosovou funkcí



Učení gradientní metodou - různé strategie:

1. v každém kroku adaptuji váhy jen jednoho náhodně zvoleného neuronu
2. váhové vektory všech neuronů adaptuji současně

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Učení gradientní metodou ... např. chybová funkce SSE

$$E_{SSE}(W) = \sum_{j=1}^{m} E_{SSE}(\vec{w}_j) = \frac{1}{2}\sum_{j=1}^{m}\sum_{p=1}^{N}(d_{pj} - y_{pj})^2$$

$$= \frac{1}{2}\sum_{j=1}^{m}\sum_{p=1}^{N}\left(d_{pj} - f(\sum_{i=0}^{n} w_{ij}x_{pi})\right)^2 = \sum_{p=1}^{N} E_p(W)$$

$E_p(W) = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$ ... je chybová funkce pro jeden vzor

Parciální derivace:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial y_{pj}}\frac{\partial y_{pj}}{\partial \xi_{pj}}\frac{\partial \xi_{pj}}{\partial w_{ij}} = -(d_{pj} - y_{pj})f'(\xi_{pj})x_{pi}$$

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Učení gradientní metodou

- Adaptační pravidlo pro jednu váhu:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha_t(d_{tj} - y_{tj})f'(\xi_{tj})x_{ti}$$

- Adaptační pravidlo pro jeden neuron s vektorem vah $\vec{w}_j$:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha_t(d_{tj} - y_{tj})f'(\xi_{tj})\vec{x}_t^T$$

- Adaptační pravidlo pro jednu vrstvu neuronů s maticí vah W:

$$W(t+1) = W(t) + \alpha_t\vec{x}_t^T[f'(\vec{\xi}_t) \circ (\vec{d}_t - \vec{y}_t)]$$

  - velmi efektivní výpočet
  - $\circ$ ... Hadamardův (prvkový, element-wise) součin vektorů

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Učení gradientní metodou

- Adaptační pravidlo pro jednu vrstvu neuronů s maticí vah W:

$$W(t+1) = W(t) + \alpha_t \vec{x}_t^T [f'(\vec{\xi}_t) \circ (\vec{d}_t - \vec{y}_t)]$$

  - velmi efektivní výpočet
  - $\circ$ ... Hadamardův (prvkový, element-wise) součin vektorů
- Adaptační pravidlo pro dávkovou variantu:

$$Xi_t = XW$$

$$Y_t = f(Xi_t)$$

$$W(t+1) = W(t) + \alpha_t X_t^T [f'(Xi_t) \circ (D_t - Y_t)]$$

  - ještě efektivnější výpočet

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Obecné schéma gradientního algoritmu (GD, gradient descent)

1. Inicializuj váhy malými náhodnými reálnými hodnotami .... $W(0)$ tvaru $(n+1) \times m$
   Inicializuj parametr učení α_0.... $1 > \alpha_0 > 0$
2. Předlož další trénovací vzor $(\vec{x}_t, \vec{d}_t)$ a spočti potenciál a skutečný výstup modelu: $\vec{\xi}_t = \vec{x}_t W$
   $\vec{y}_t = f(\vec{\xi}_t)$
3. Adaptuj váhy:

$$W(t+1) = W(t) + \alpha_t \vec{x}_t^T [f'(\vec{\xi}_t) \circ (\vec{d}_t - \vec{y}_t)]$$

4. Případně aktualizuj parametr učení : $\alpha_t \to \alpha_{t+1}$
5. Pokud není konec, přejdi ke kroku 2.

Jednovrstvá neuronová síť se spojitou přenosovou funkcí

Obecné schéma dávkového gradientního algoritmu (batch GD)

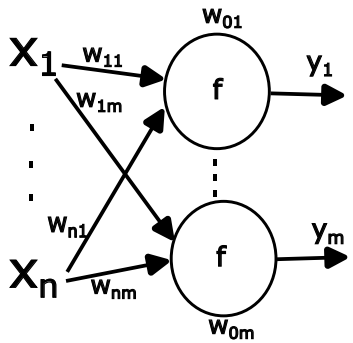
1. Inicializuj váhy malými náhodnými reálnými hodnotami .... $W(0)$ tvaru $(n+1) \times m$
   Inicializuj parametr učení α_0.... $1 > \alpha_0 > 0$
2. Předlož trénovací množinu X a spočti potenciál a skutečný výstup modelu: $Xi_t = XW$
   $Y_t = f(Xi_t)$
3. Adaptuj váhy:

$$W(t+1) = W(t) + \alpha_t X_t^T [f'(Xi_t) \circ (D_t - Y_t)]$$

4. Případně aktualizuj parametr učení : $\alpha_t \rightarrow \alpha_{t+1}$
5. Pokud není konec, přejdi ke kroku 2.

perceptron_layer.ipynb

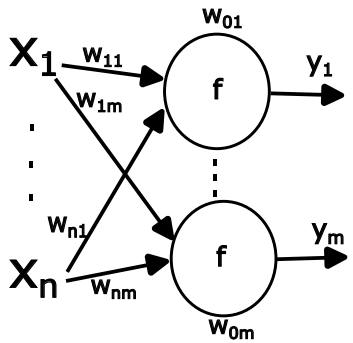
Mnohorozměrná lineární regrese



- model: neuronová síť tvořena jednou vrstvou lineárních neuronů (více vrstev by nepřineslo žádný benefit - na rozmyšlenou)
- chybová funkce pro učení gradientní metodou: SSE(základ), MSE (střední kvadratická chyba), MAE (střední absolutní chyba, pro data s odlehlými vzory)

Atmosferický tlak	**Intenzita větru**	**Teplota**	**Riziko srážek**
-1.3	-0.5	-0.2	0.9
...	...	...	...

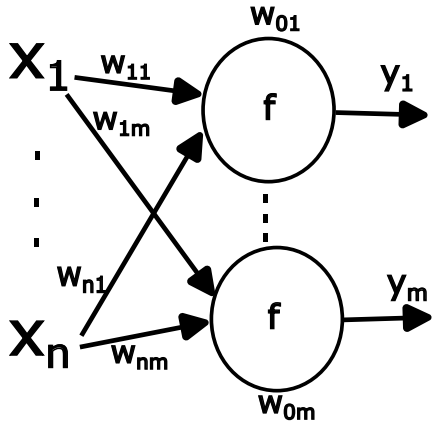
Lineární klasifikace do více tříd (rozpoznávání vzorů)



- model: jedna vrstva neoronů s tanh přenosovou funkcí nebo sigmoidou (pak je třeba požadované výstupy kódovat binárně)
- chybová funkce pro učení gradientní metodou: SSE, cross-entropy (menší riziko saturace)
- vyhodnocovací kriteria: např. přesnost (accuracy) - procento správně rozpoznaných vzorů

Velikost	**Srst**	**Mluví?**	**Kočka**	**Pes**	**Papoušek**	**Kapr**
-1	0	-1	1	-1	-1	-1
1	1	-1	-1	1	-1	-1
...	...	...	...		...	

Lineární klasifikace do více tříd (rozpoznávání vzorů)



- počet neuronů v síti zvolíme stejný jako počet tříd
- každý neuron se během učení učí rozpoznávat vzory z jedné ze tříd

Jak zjistím vítěznou třídu?

- argmax

$$k_{max} = argmax_k y_k$$

- softmax - pro spojitou přenosovou funkci

$$softmax(y_k) = \frac{e^{y_k}}{\sum_{i}^{m} e^{y_i}}$$

Lineární klasifikace do více tříd (rozpoznávání vzorů)

Jak zjistím vítěznou třídu?

- argmax

$$k_{max} = \text{argmax}_k \, y_k$$

  ... index vítězné třídy

- softmax - pro spojitou přenosovou funkci získáme pravděpodobnosti jednotlivých tříd

$$softmax(y_k) = \frac{e^{y_k}}{\sum_{j=1}^{m} e^{y_j}}$$

  - V knihovnách je softmax často implementován jako speciální výstupní plně propojená vrstva.

Cross-entropy pro více tříd

Chybová funkce Cross-entropy pro více tříd:

- Používá se pro klasifikaci do více tříd v kombinaci se softmax výstupem.

$$E = -\sum_{p=1}^{N}\sum_{j=1}^{m} d_{pj} \log y_{pj} \tag{1}$$

- d_{pj} je požadovaný výstup (1 pro správnou třídu, jinak 0),
- y_{pj} je pravděpodobnost třídy j získaná softmaxem.

Gradient chyby vzhledem k vahám:

$$\frac{\partial E_p}{\partial w_{ij}} = (y_{pj} - d_{pj})x_{pi} \tag{2}$$

$\rightarrow$ Efektivnější učení, přirozená interpretace jako negativní log pravděpodobnosti správné třídy.

Příklady

perceptron_layer.ipynb

- Ukázka vlastní implementace jednovrstvé neuronové sítě
- Jednoduchý příklad: žlučník (příklad 1) rozšířený o další výstupní proměnné
  - Pozorování: gradientní hodnota si s úlohami poradí, i když učení trvá déle a je obtížnější nastavit správně hyperparametry
- Obtížnější příklad: písmena
  - Zde se plně projeví výhoda chybové funkce Cross-entropy nad SSE
  - Dávkové učení je výrazně efektivnější než iterativní
  - Pozorování: skoky u chybové funkce a saturace neuronů u SSE

Dnešní hodina

1. Neuronová síť tvořená jednou vrstvou neuronů - dokončení
2. **Neuronová síť tvořená více vrstvami neuronů - úvod**

Vrstevnatá neuronová síť (multi-layer perceptron, MLP, 1980)

- hierarchická -**sekvenční**- architektura, neurony jsou uspořádány do vrstev
- **dense layers (plně propojené vrstvy)**: všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

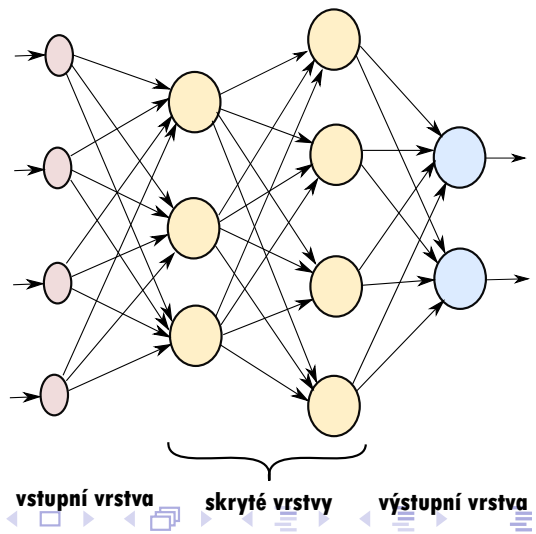
Speciální vstupní vrstva:

- odpovídá vstupům neuronové sítě

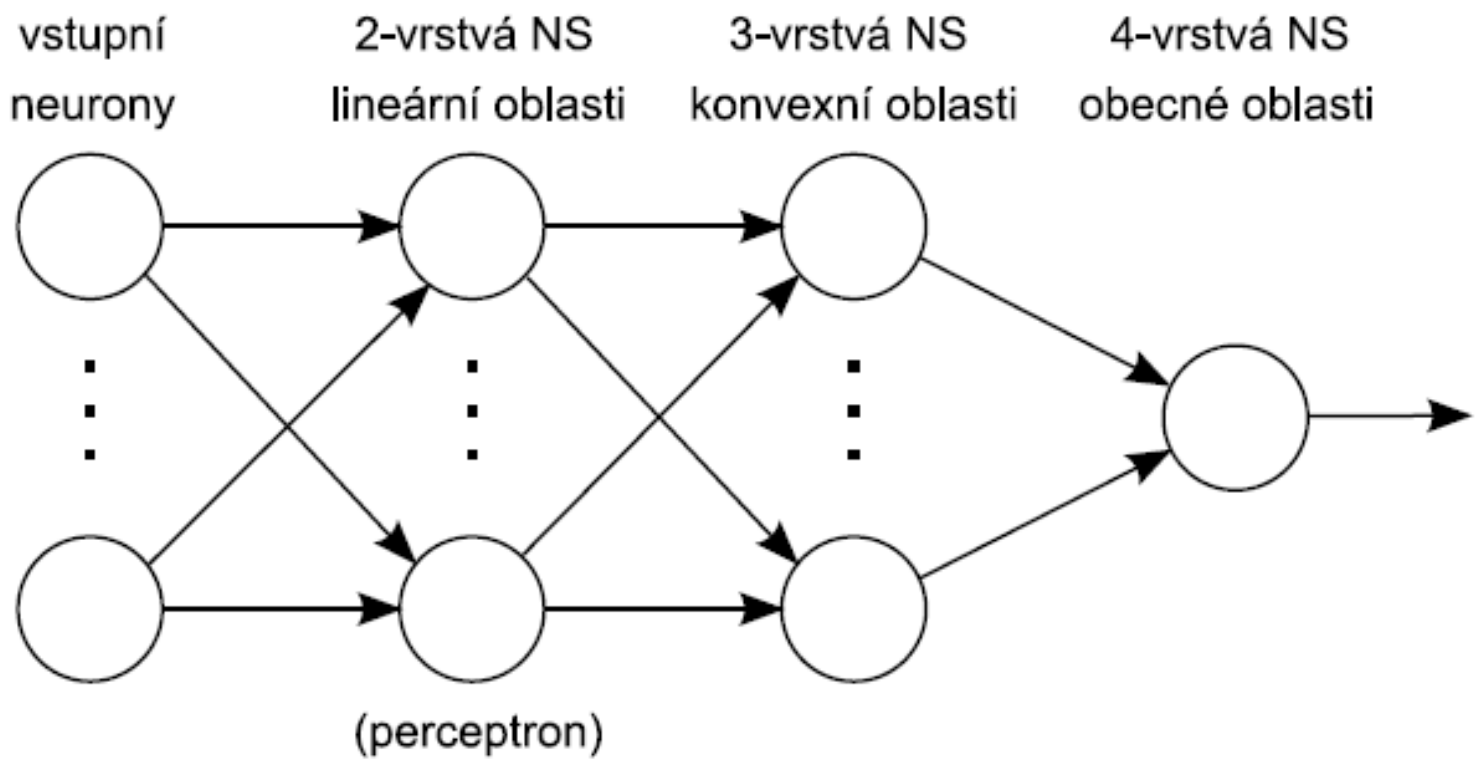
Výstupní vrstva

- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů

Zbylé vrstvy jsou **skryté**.



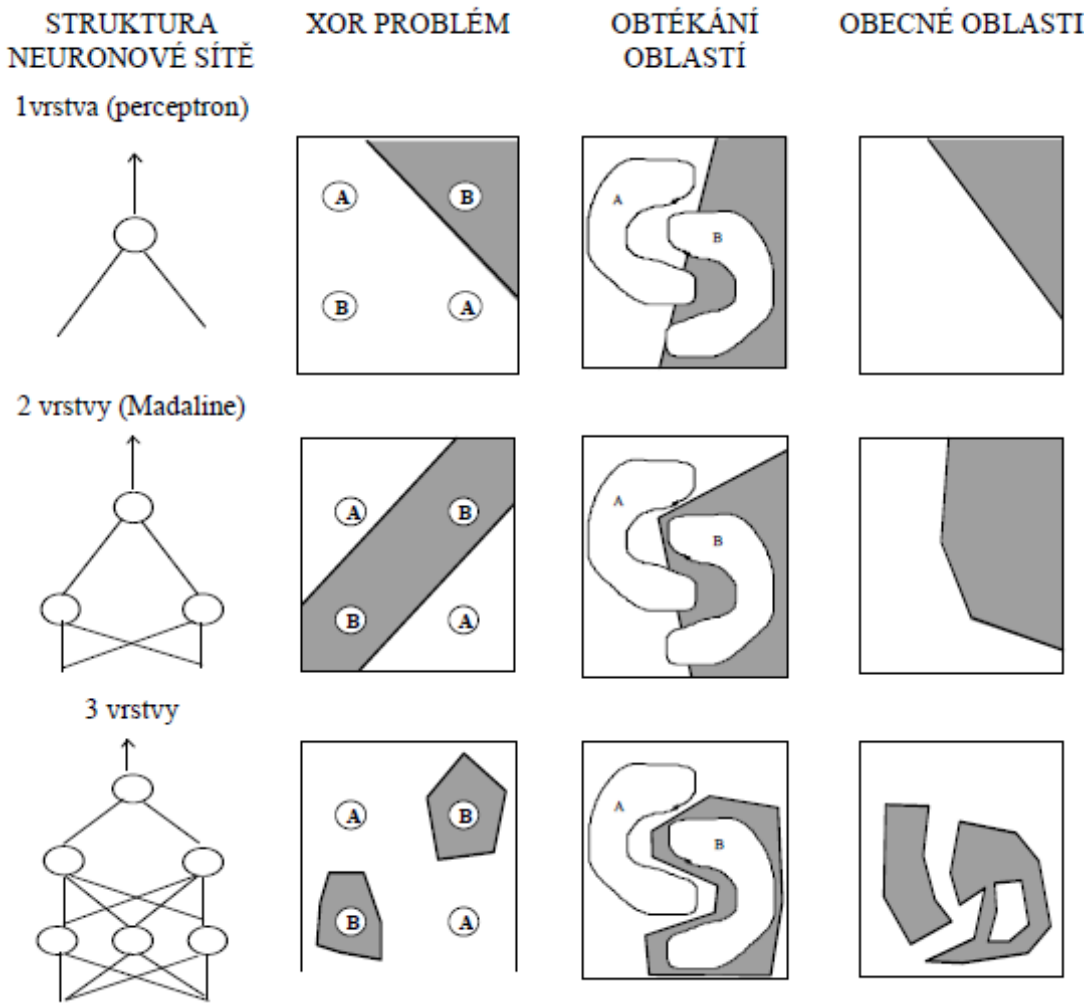
Motivace: Vícevrstvý perceptron



- neurony se skokovou lineární funkcí nebo s její spojitou aproximací (sigmoid, tanh)

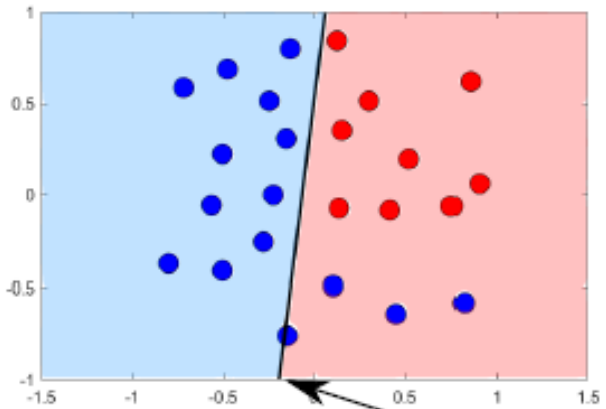
Zdroj: E.Volná: Neuronové sítě 1, Ostrava, 2008

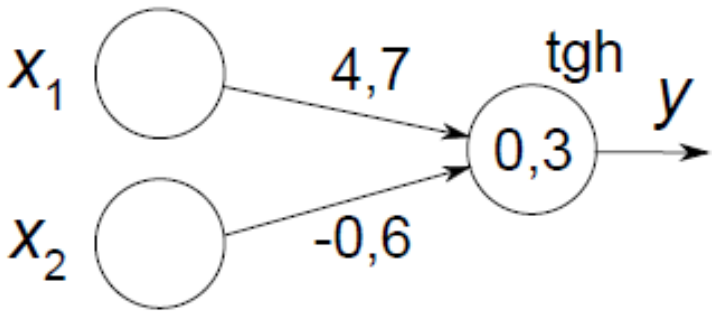
Motivace: Vícevrstvý perceptron



Zdroj: E.Volná: Neuronové sítě 1, Ostrava, 2008

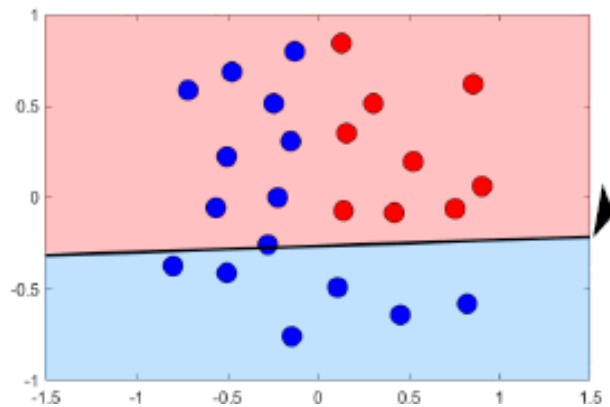
Motivace: Vícevrstvý perceptron

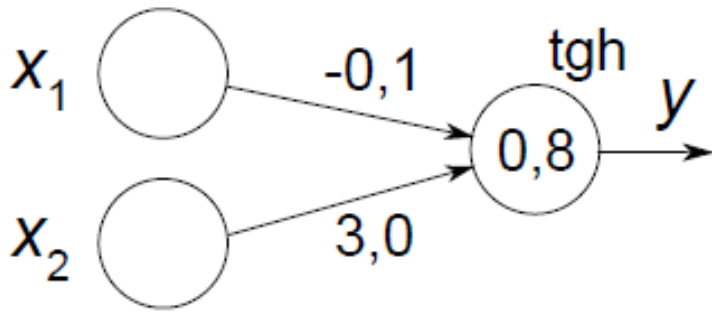




$$0{,}3 + 4{,}7x_1 - 0{,}6\,x_2 = 0$$

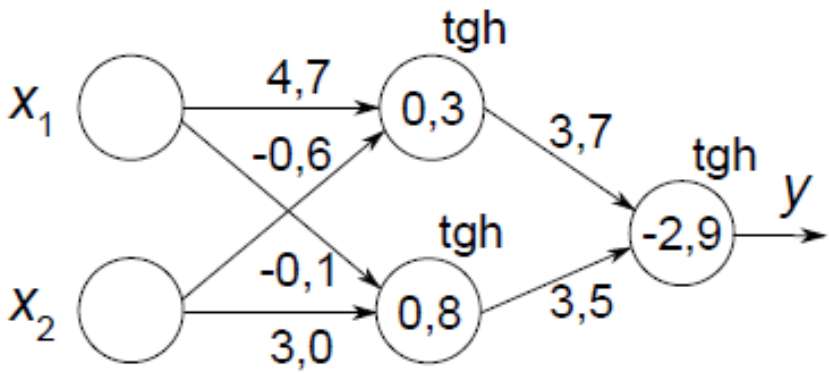
$$0{,}8 - 0{,}1x_1 + 3{,}0\,x_2 = 0$$

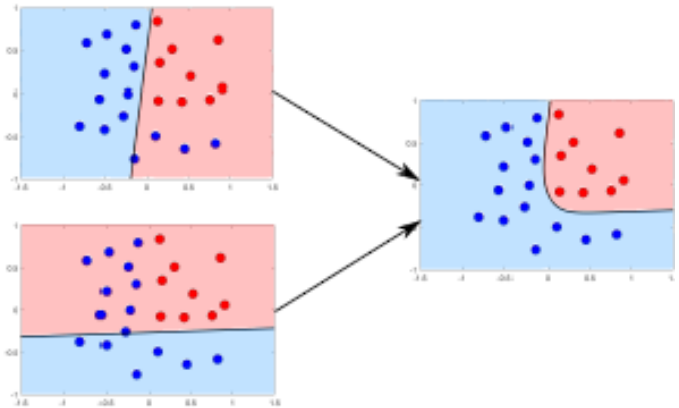




Zdroj: Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

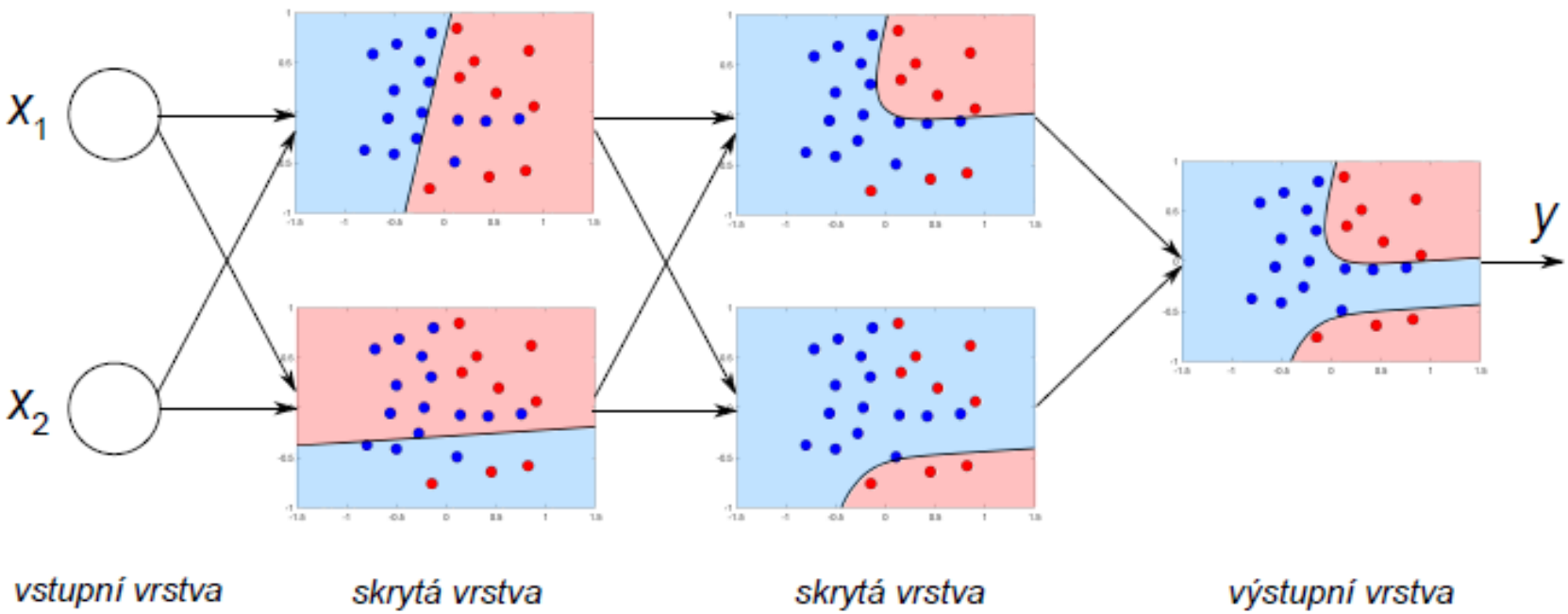
Motivace: Vícevrstvý perceptron





Zdroj: Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

Motivace: Vícevrstvý perceptron



Zdroj: Kateřina Horaisová: slidy k předmětu Neuronové sítě 2, FJFI ČVUT Děčín

Motivace: Vícevrstvý perceptron

→ pro libovolné klasifikační úlohy je dostačující mít neuronovou síť se dvěma skrytými vrstvami a jednou výstupní

A co vícevrstvá lineární síť?

- Skládat lineární vrstvy za sebe nemá smysl - více lineárních vrstev za sebou je ekvivaletních jedné lineární vrstvě

Lze MLP použít pro řešení regresních úloh?

- Ano, ale pouze výstupní vrstva bývá lineární.
- Pro zachycení nelineárních závislostí ve skrytých vrstvách je nutné použít nelineární aktivační funkce (např. ReLU, sigmoid, tanh).

Motivace: Vícevrstvý perceptron

Univerzální aproximační věta (Cybenko, 1989; Hornik et al., 1989)

- Vrstevnatá neuronová síť s jednou skrytou vrstvou s dostatečným počtem neuronů a nelineární aktivační funkcí stačí k aproximaci libovolné spojité funkce na kompaktní množině.
- V praxi se však více osvědčují sítě s více skrytými vrstvami:
  - snazší a rychlejší konvergence při učení
  - efektivnější reprezentace složitějších funkcí s méně parametry (hlubší sítě obvykle potřebují méně neuronů)
  - lepší schopnost sítě zobecňovat.

Vrstevnatá neuronová síť (multi-layer perceptron, MLP, 1980)

- hierarchická **sekvenční** architektura: neurony jsou uspořádány do vrstev
- **dense layers (plně propojené vrstvy)**: všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

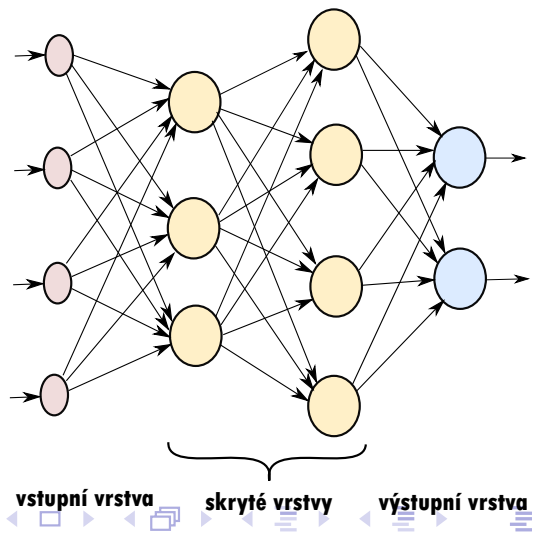
Speciální vstupní vrstva:

- odpovídá vstupům neuronové sítě

Výstupní vrstva

- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů

Zbylé vrstvy jsou **skryté**.



Vrstevnatá neuronová síť (multi-layer perceptron, MLP)

Jak ji implementovat?

- posloupnost (seznam) vrstev $L_0, ..., L_{max}$
- jednu vrstvu již implementovat umíme

Výpočet odezvy (výstupu) u vrstevnaté neuronové sítě:

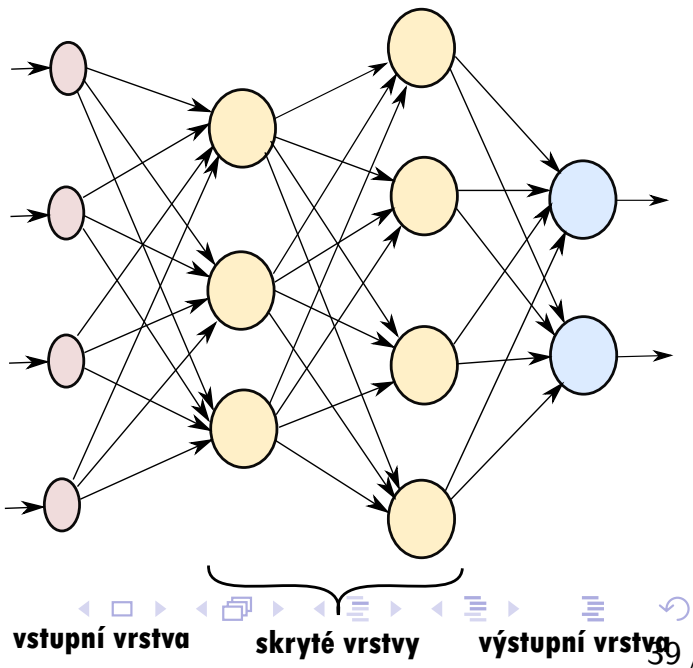
- dopředným průchodem (**forward pass**)
- postupujeme po vrstvách směrem od vstupní vrstvy po výstupní:
  - aktuální vrstvě předložíme její vstup
  - spočteme výstup vrstvy
  - ten poslouží jako vstup následující vrstvy

Vrstevnatá neuronová síť (multi-layer perceptron, MLP)

Forward pass (dopředný průchod): výpočet odezvy u vrstevnaté neuronové sítě:

- pro daný vstupní vektor $\vec{x}$ délky n model spočítá výstupní vektor $\vec{y}$ délky m

1. výstup neuronů ve vstupní vrstvě:
   $y_i = x_i$
2. Postupujeme ve směru od první skryté vrstvy k výstupní a pro každý neuron j spočteme (a zapamatujeme si) jeho výstup y_j (využijeme k tomu aktivity neuronů v předchozí vrstvě):
   $y_j = f(\xi_j) = f(\sum_i w_{ij} y_i + b_i)$ (i je index přes neurony ve vrstvě předcházející neuronu j)



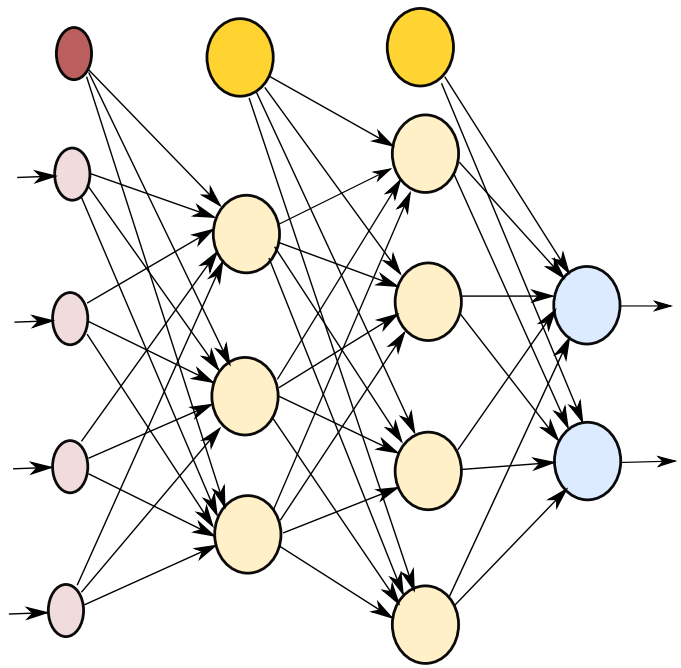
Vrstevnatá neuronová síť (multi-layer perceptron, MLP)

Forward pass (dopředný průchod): výpočet odezvy u vrstevnaté neuronové sítě:

- opět si zjednodušíme a zefektivnníme výpočet pomocí maticových operací a fiktivních vstupů (zde neuronů)

Fiktivní neurony

- ke každé skryté vrstvě (podobně jako ke vstupní) přidáme jeden fiktivní neuron s konstantním výstupem 1, který bude reprezentovat prahy (biasy) neuronů v následující vrstvě

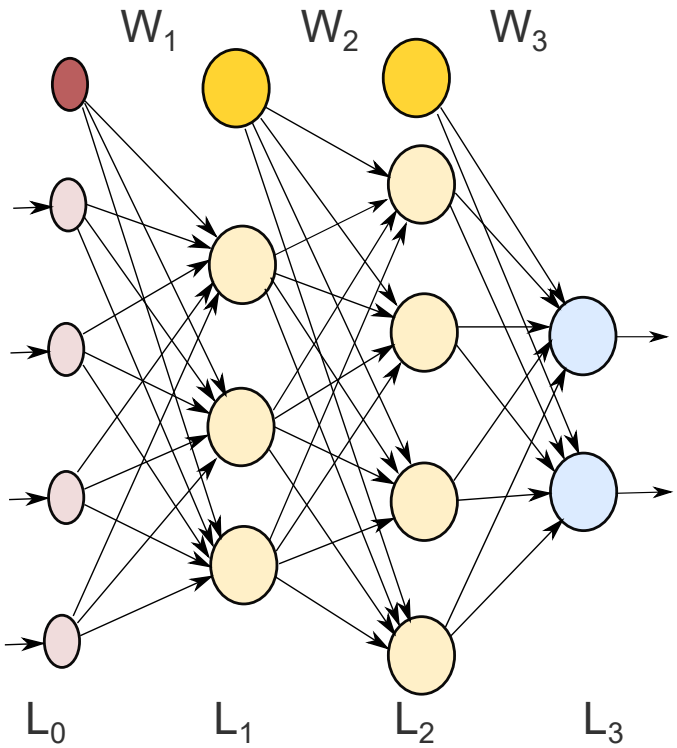


Vrstevnatá neuronová síť (multi-layer perceptron, MLP)

Maticová reprezentace vrstevnaté neuronové sítě:

- mějme vrstevnatou neuronovou síť s vrstvami L_0 (vstupní),...,L_{max} (výstupní)
- váhy všech neuronů můžeme reprezentovat pomocí matic $W_1, ..., W_{L_{max}}$
- W_L ... matice vah mezi vrstvou $L-1$ a L o rozměrech $(n_{L-1}+1) \times n_L$

(podobně jako pro jednovrstvou neuronovou síť)



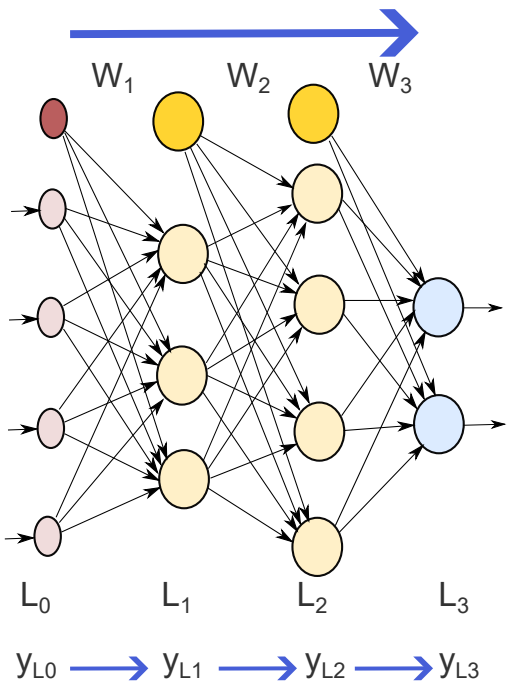
Výpočet odezvy neuronové sítě maticově I.

1. Předložíme vstupní vzor $\vec{x}$
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme vektory výstupů těchto vrstev $\vec{y}_0, ..., \vec{y}_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $\vec{y}_0 = \vec{x}$
   - Pro $L = L_1 ..., L_{max}$:

$$\vec{z}_L = f(\vec{\xi}_L) = f(\vec{y}_{L-1} W_L)$$

$$\vec{y}_L = (1|\vec{z}_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $\vec{y} = (y_1, ..., y_m) = \vec{y}_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě



$\rightarrow$ velmi efektivní

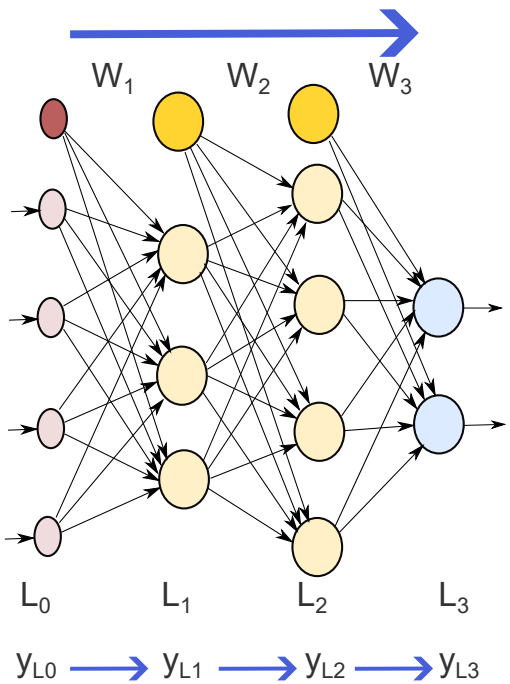
Výpočet odezvy neuronové sítě maticově II.

1. Předložíme matici vstupních vzorů X
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme matice výstupů těchto vrstev $Y_0, ..., Y_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $Y_0 = X$
   - Pro $L = L_1, .., L_{max}$:

$$Z_L = f(\xi_L) = f(Y_{L-1} W_L)$$

$$Y_L = (1|Z_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $Y = Y_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě



→ ještě efektivnější

Učení vrstevnaté neuronové sítě

Konfigurace neuronové sítě:

- vektor vah (a biasů) všech neuronů v síti $\vec{w} \sim$ vektor všech parametrů modelu

Jak vrstevnatou neuronovou síť budeme učit?

- můžeme použít gradientní metodu pro zvolenou chybovou funkci E a vektor všech parametrů modelu $\vec{w}$:

$$\vec{w}(t+1) = \vec{w}(t) - \alpha \nabla E(\vec{w})$$

- výpočet parciálních derivací a proces adaptace vah budou složitější než u jedné vrstvy neuronů
- výpočet značně zjednoduší algoritmus zpětného šíření chyby (backpropagation)

Algoritmus zpětného šíření (Backpropagation)

(Werbos, Rumelhart, 1974-1986)

Máme k dispozici

- Trénovací množina T s N trénovacími vzory $(\vec{x}_p, \vec{d}_p)$.
  - $x^p = (x_1^p, ..., x_n^p)$ ... vstupní vzor
  - $d^p = (d_1^p, ..., d_m^p)$ ... požadovaný výstup

$x_{10} = 1$	x_{11}	...	x_{1n}	d_{11}	...	d_{1m}
...	...		...	...		...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_{N1}	...	d_{Nm}

- Vrstevnatá neuronová síť s danou architekturou a s $n + 1$ vstupními a m výstupními neurony. Neurony musí mít spojitou, diferencovatelnou přenosovou funkci.

Cíl

- Nastavit váhy všech neuronů v síti tak, aby byl skutečný výstup sítě stejný jako požadovaný.

Algoritmus zpětného šíření (Backpropagation)

Cílová (chybová) funkce

- místo SSE se často používá MSE (mean squared error) ... průměrná chyba přes všechny vzory $E_{MSE} = E_{SSE}/N$
  - pro jeden trénovací vzor:

$$E_p(\vec{w}) = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$$

  - pro celou trénovací množinu:

$$E(\vec{w}) = \frac{1}{N}\sum_{p=1}^{N} E_p = \frac{1}{2N}\sum_{p=1}^{N}\sum_{j=1}^{m}(d_{pj} - y_{pj})^2$$

- ale používají se i jiné chybové funkce (např. cross-entropy)

Cíl algoritmu zpětného šíření

- minimalizace chybové funkce E na dané trénovací množině T

Algoritmus zpětného šíření (Backpropagation)

Základní princip: je to standardní gradientní metoda

1. náhodně inicializuj parametry modelu (váhy a biasy)
2. opakuj trénovací cyklus:
   - připrav dávku (batch) trénovacích vzorů X a odpovídajících požadovaných výstupů D
   - spočti skutečný výstup (odezvu) modelu Y
   - spočti chybu modelu (jak moc se liší Y a D)
   - aktualizuj parametry (váhy a biasy) tak, aby se chyba modelu o něco zmenšila (tj. proti směru gradientu chybové funkce)

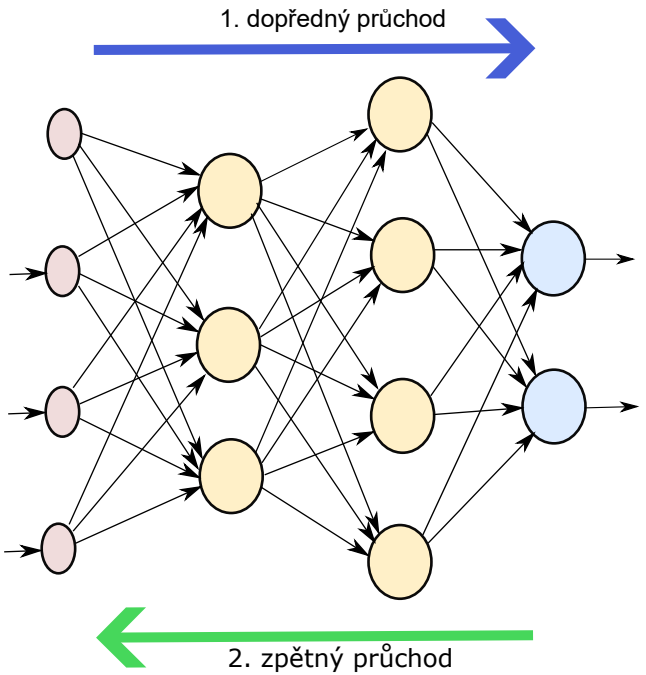
$$w_i(t+1) = w_i(t) - \alpha_t \frac{\partial E_t}{\partial w_i}$$

Hezké vizualizace hladin chybové funkce:
https://jithinjk.github.io/blog/nn_loss_visualized.md.html
https://izmailovpavel.github.io/curves_blogpost/

Algoritmus zpětného šíření (Backpropagation)

Základní princip (backpropagation)

1. Spočteme skutečnou odezvu sítě pro daný trénovací vzor (nebo dávku vzorů).
   - jedním průchodem od vstupní vrstvy směrem k výstupní (forward pass)
2. Porovnáme skutečnou a požadovanou odezvu sítě.
3. Adaptujeme váhy a prahy:
   - proti směru gradientu chybové funkce
   - jedním průchodem od výstupní vrstvy směrem ke vstupní (backward pass)



Backpropagation - Adaptační pravidla

- Chybová funkce pro jeden trénovací vzor $(\vec{x}_t, \vec{d}_t)$ a skutečný výstup sítě $\vec{y}_t$:

$$E_t = \frac{1}{2}\sum_{j=1}^{m}(d_{tj} - y_{tj})^2$$

- Parciální derivace:

$$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial y_{tj}}\frac{\partial y_{tj}}{\partial \xi_{tj}}\frac{\partial \xi_{tj}}{\partial w_{ij}}$$

- Adaptační pravidlo pro váhu z neuronu i do neuronu j v čase t:

$$w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t),$$

- $\triangle w_{ij}(t)$ je přírůstek váhy w_{ij} přispívající k minimalizaci E_t

$$\triangle w_{ij}(t) = -\alpha\frac{\partial E_t}{\partial w_{ij}} = -\alpha\frac{\partial E_t}{\partial y_{tj}}\frac{\partial y_{tj}}{\partial \xi_{tj}}\frac{\partial \xi_{tj}}{\partial w_{ij}}$$

- α je parametr učení (learning rate)

Backpropagation - Adaptační pravidla

Myšlenka

- Není třeba počítat parciální derivace $\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \frac{\partial \xi_{tj}}{\partial w_{ij}}$ pro každou váhu nezávisle (to by bylo extrémně neefektivní)
- Abychom spočítali $\frac{\partial E_t}{\partial w_{ij}}$ pro nějakou váhu, konkrétně její člen $\delta_j = \frac{\partial E_t}{\partial \xi_{tj}}$, můžeme využít chybové členy $\delta_k = \frac{\partial E_t}{\partial \xi_{tk}}$ spočítané pro neurony k v následující vrstvě.
- $\rightarrow$ Jedním zpětným průchodem (vrstvu po vrstvě) spočítáme tyto chybové členy δ_j pro každý neuron j
- Dopočítat $\frac{\partial E_t}{\partial w_{ij}} = \delta_j \frac{\partial \xi_{tj}}{\partial w_{ij}}$ pro každou váhu již pak je snadné

Backpropagation - Adaptační pravidla

Označme

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = \frac{\partial E_t}{\partial \xi_{tj}} \text{ ... chybový člen pro nejuron } j$$

$\rightarrow$ to je chybový člen, který budeme propagovat
Pak

$$\triangle w_{ij}(t) = -\alpha \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} \frac{\partial \xi_{tj}}{\partial w_{ij}} = -\alpha \delta_{tj} \frac{\partial \xi_{tj}}{\partial w_{ij}} = -\alpha \delta_{tj} y_{ti}$$

kde:

$$\frac{\partial \xi_{tj}}{\partial w_{ij}} = \frac{\partial (\sum_k w_{kj} y_{tk})}{\partial w_{ij}} = y_{ti}$$

(k je index přes všechny neurony ve vrstvě předcházející j)

Backpropagation - Adaptační pravidla

I. Pro neurony j ve výstupní vrstvě:

$$E_t = \frac{1}{2}\sum_{j=1}^{m}(d_{tj} - y_{tj})^2$$

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}}\frac{\partial y_{tj}}{\partial \xi_{tj}} = -(d_{tj} - y_{tj})f'(\xi_{tj})$$

Pro váhy w_{ij} mezi poslední skrytou vrstvou a výstupní:

$$\triangle w_{ij}(t) = -\alpha\delta_{tj}y_{ti} = \alpha(d_{tj} - y_{tj})f'(\xi_{tj})y_{ti}$$

Back-propagation - Adaptační pravidla

Pro neurony j v poslední skryté vrstvě:

$$\begin{aligned} E_t &= \frac{1}{2}\sum_{k=1}^{m}(d_{tk} - y_{tk})^2 = \frac{1}{2}\sum_{k=1}^{m}(d_{tk} - f(\xi_{tk}))^2 \\ &= \frac{1}{2}\sum_{k=1}^{m}(d_{tk} - f(\sum_{j} w_{jk} y_{tj}))^2 \end{aligned}$$

(k je index přes všechny výstupní neurony, j je index přes všechny neurony v poslední skryté vrstvě)

$$\begin{aligned} \frac{\partial E_t}{\partial y_{tj}} &= \sum_{k=1}^{m} \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial y_{tj}} = \sum_{k=1}^{m} \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial \xi_{tk}} \frac{\partial \xi_{tk}}{\partial y_{tj}} \\ &= \sum_{k=1}^{m} \delta_{tk} \frac{\partial \xi_{tk}}{\partial y_{tj}} = \sum_{k=1}^{m} \delta_{tk} w_{jk} \end{aligned}$$

Back-propagation - Adaptační pravidla

Pro neurony j v poslední skryté vrstvě:

$$\frac{\partial E_t}{\partial y_{tj}} = \sum_{k=1}^{m} \delta_{tk} w_{jk}$$

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = \left(\sum_{k=1}^{m} \delta_{tk} w_{jk} \right) f'(\xi_{tj})$$

Pro váhy w_{ij} mezi předposlední a poslední skrytou vrstvou:

$$\triangle w_{ij}(t) = -\alpha \delta_{tj} y_{ti} = -\alpha \left(\sum_{k=1}^{m} \delta_{tk} w_{jk} \right) f'(\xi_{tj}) y_{ti}$$

(k je index přes všechny výstupní neurony)

Back-propagation - Adaptační pravidla

Pro neurony j v libovolné skryté vrstvě:

$$\frac{\partial E_t}{\partial y_{tj}} = \sum_k \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial y_{tj}} = \sum_k \frac{\partial E_t}{\partial y_{tk}} \frac{\partial y_{tk}}{\partial \xi_{tk}} \frac{\partial \xi_{tk}}{\partial y_{tj}}$$

$$= \sum_k \delta_{tk} \frac{\partial \xi_{tk}}{\partial y_{tj}} = \sum_k \delta_{tk} w_{jk}$$

(index k jde přes všechny neurony v následující vrstvě)

$$\delta_{tj} = \frac{\partial E_t}{\partial y_{tj}} \frac{\partial y_{tj}}{\partial \xi_{tj}} = (\sum_k \delta_{tk} w_{jk}) f'(\xi_{tj})$$

Back-propagation - Adaptační pravidla

Celkem:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha\delta_{tj}y_{ti},$$

kde

- pro výstupní neuron j:

$$\delta_{tj} = f'(\xi_{tj})(y_{tj} - d_{tj}).$$

- pro skrytý neuron j:

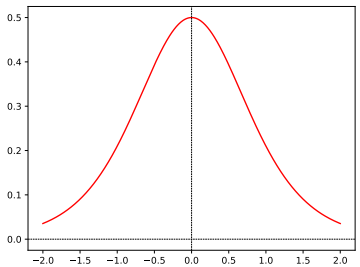
$$\delta_{tj} = f'(\xi_{tj})\sum_{k}(\delta_{tk}w_{jk}).$$

Připomenutí: Výpočet derivace přenosové funkce

Sigmoidální ... logsig

$$y = f(\xi) = \frac{1}{1 + e^{-\lambda\xi}}$$

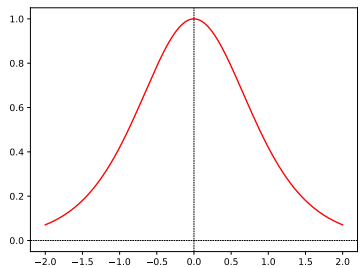
$$f'(\xi) = \lambda y(1 - y)$$



Hyperbolický tangens ... tanh

$$y = f(\xi) = \frac{1 - e^{-2\lambda\xi}}{1 + e^{-2\lambda\xi}}$$

$$f'(\xi) = \lambda^2(1 + y)(1 - y)$$



Algoritmus zpětného šíření (iterativní varianta)

1. **Inicializace sítě**
   Inicializujeme váhy a prahy malými náhodnými hodnotami
2. **Předložíme další trénovací vzor** ve tvaru $(\vec{x}_t, \vec{d}_t)$
3. **Dopředný výpočet:**
   - Postupujeme ve směru od vstupní vrstvy k výstupní a pro každý neuron j spočteme (a zapamatujeme si) jeho výstup y_{tj} (využijeme k tomu aktivity neuronů v předchozí vrstvě, včetně fiktivního)
   $$y_{tj} = f(\xi_{tj}) = f(\sum_i w_{ij} y_{ti})$$
   (i je index přes neurony ve vrstvě předcházející neuronu j)
   - Výpočet provádíme samozřejmě maticově po vrstvách

Algoritmus zpětného šíření (iterativní varianta)

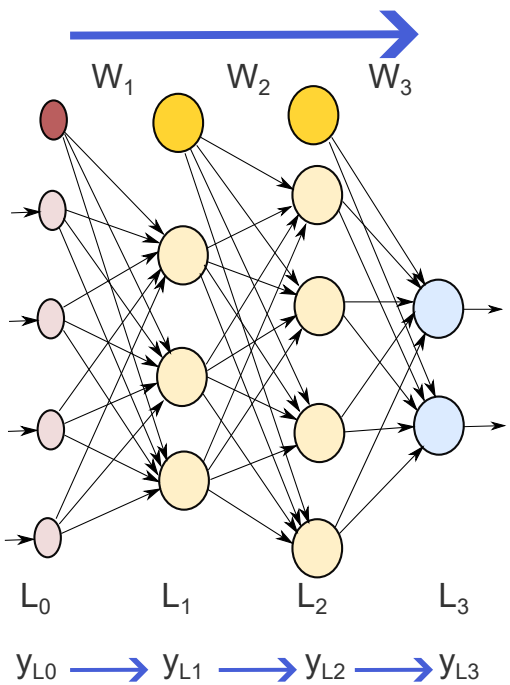
3. **Dopředný výpočet maticově:**
   Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme vektory výstupů těchto vrstev $\vec{y}_{L_0}, ..., \vec{y}_{L_{max}}$:

- Pro $L = L_0$ (vstupní vrstva): $\vec{y}_{L_0} = \vec{x}$
- Pro $L = L_1..., L_{max}$:

$$\vec{z}_L = f(\vec{\xi}_L) = f(\vec{y}_{L-1} W_L)$$

$$\vec{y}_L = (1|\vec{z}_L)$$

W_L je rozšířená matice vah mezi vrstvou (L-1) a L



- Výstup sítě $\vec{y} = (y_1, ..., y_m) = \vec{y}_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě

Algoritmus zpětného šíření (batch varianta)

Dopředný výpočet maticově pro batch GD:

1. Předložíme matici vstupních vzorů X
2. Pro vrstvy $L = L_0, L_1, ..., L_{max}$ postupně spočítáme matice výstupů těchto vrstev $Y_0, ..., Y_{L_{max}}$:
   - Pro $L = L_0$ (vstupní vrstva): $Y_0 = X$
   - Pro $L = L_1, .., L_{max}$:

$$Z_L = f(\xi_L) = f(Y_{L-1} W_L)$$

$$Y_L = (1|Z_L)$$

   W_L je rozšířená matice vah mezi vrstvou (L-1) a L
3. Výstup sítě $Y = Y_{L_{max}}$ tvoří výstupy neuronů ve výstupní vrstvě

Algoritmus zpětného šíření (iterativní varianta)

4. **Zpětný výpočet** Postupujeme ve směru od výstupní vrstvy k první skryté a pro každý neuron j spočteme (a zapamatujeme si) chybový člen δ_{tj} pro každý neuron j a aktualizujeme všechny váhy w_{ij} vedoucí do vrcholu j (ve směru od výstupní vrstvy k vstupní):
   - Pro výstupní neurony j spočteme:
   $$\delta_{tj} = f'(\xi_{tj})(y_{tj} - d_{tj})$$
   - Pro skryté neurony j spočteme:
   $$\delta_{tj} = f'(\xi_{tj}) \sum_k (\delta_{tk} w_{jk}).$$
   (k je index přes neurony ve vrstvě následující po vrstvě obsahující neuron j)
   - Pro váhu každé hrany (z nějakého neuronu i včetně fiktivního) do neuronu j:
   $$w_{ij}(t+1) = w_{ij}(t) - \alpha \delta_{tj} y_{ti}$$

Algoritmus zpětného šíření (iterativní varianta)

4. **Zpětný výpočet maticově**
   Pro vrstvy $L = L_{max}, ..., L_1$ postupně spočítáme chybové členy $\vec{\delta_L}$ a aktualizujeme matice vah W_L mezi vrstvami (L-1) a L:
   - Pro výstupní vrstvu L_{max} spočteme:

$$\vec{\delta}_{L_{max}} = f'(\vec{\xi}_{L_{max}}) \circ (\vec{y}_t - \vec{d}_t)$$

   - Pro skryté vrstvy L spočteme:

$$\vec{\delta}_L = (f'(\vec{\xi}_L) \circ \vec{\delta}_{L+1}) W_{L+1}^T.$$

   - Aktualizujeme matici vah W_L mezi vrstvami (L-1) a L:

$$W_L(t+1) = W_L(t) - \alpha \vec{y}_{L-1}^T \vec{\delta}_L$$

Algoritmus zpětného šíření (batch varianta)

Zpětný výpočet maticově pro batch GD:

- Pro vrstvy $L = L_{max}, ..., 1$ postupně spočítáme chybové členy Δ_L a aktualizujeme matice vah W_L mezi vrstvami (L-1) a L:
  - Pro výstupní vrstvu L_{max} spočteme:

$$\Delta_{L_{max}} = f'(\xi_{L_{max}}) \circ (Y - D)$$

  - Pro skryté vrstvy L spočteme:

$$\Delta_L = (f'(\xi_L) \circ \delta_{L+1}) W_{L+1}^T$$

  - Aktualizujeme matici vah W_L mezi vrstvami (L-1) a L:

$$W_L(t+1) = W_L(t) - \alpha Y_{L-1}^T \Delta_L$$

Algoritmus zpětného šíření (iterativní varianta)

5. **Ukončující podmínka**
   Pokud není splněna ukončující podmínka, vrátíme se zpět ke kroku 2.
   - Předem daný maximální počet epoch.
   - Časový limit.
   - Jakmile je průměrná chyba na trénovací množině dostatečně malá ... $E < E_{min}$
   - Jakmile přestane klesat průměrná chyba na validační množině dat .... early stopping
   - Jakmile je přírůstek vah Δw moc malý ... $|\Delta w| < \Delta_{min}$

Algoritmus zpětného šíření

Jak předkládat trénovací vzory? .... různé strategie (už známe):

1. **iterativně po epochách (online GD)**: během jedné epochy se každý vzor předloží právě jednou, v rámci každé epochy vzory náhodně uspořádáme
   - maximální počet epoch .... kolikrát se předloží celá trénovací množina
2. **dávkově po epochách (batch GD)**:
   - celá trénovací množina se předloží najednou a váhy se adaptují najednou pro celou trénovací množinu
3. **dávkově po mini-batchích (SGD, stochastic gradient descend)**
   - trénovací množina se náhodně rozdělí na malé podmnožiny vzorů (mini-batch) a ty se iterativně se v náhodném pořadí předloží dávkově

Algoritmus zpětného šíření

Diskuse učící strategie

- Online GD
  - rychlé učení, ale poměrně nestabilní (algoritmus snižuje chybu pro aktuální vzor $\rightarrow$ chyba se může z zvýšit pro ostatní vzory)
  - vyšší citlivost na odlehlé vzory a na volbu hyperparametrů, náhodnost (ale možnost úniku z lokálních minim)
- Batch GD
  - stabilnější, efektivní pro malá data
  - výpočetně a paměťově náročný pro velká data
  - vyšší citlivost na šum
- Mini-batch SGD
  - spojuje výhody obou předchozích metod
  - používá se pro velké datové sady a hluboké sítě

Hlavní frameworky pro hluboké učení v Pythonu

- **TensorFlow**: Open-source knihovna od Googlu.
  - Výkonný framework pro AI aplikace (mobil, server).
  - Umožňuje jak dynamické, tak statické grafy výpočtů.
- **PyTorch**: Open-source knihovna od Meta (Facebooku).
  - Flexibilní a intuitivní, ideální pro výzkum a akademickou sféru.
  - Dynamické výpočtové grafy, snadné debugování.
- **Keras**: Univerzální High-level API.
  - Srozumitelné a přístupné začátečníkům.
  - Ideální pro rychlé prototypování, běží nad TensorFlow, JAX nebo PyTorch.
- **PyTorch Lightning**: High-level rozšíření PyTorchu.
  - Odstraňuje opakující se kód při trénování modelů.
  - Podporuje více GPU, škálování modelů a snadnou reprodukovatelnost.
- **JAX**: Optimalizovaný pro výpočetní rychlost a výzkumné experimenty.
- Dříve populární **Theano**, dnes již nepodporovaný.

Příklady

NN_libraries.ipynb

- Okomentované ukázky předních pythonovských frameworků pro (hluboké) neuronové sítě (Keras, Tensorflow, PyTorch, Lightning) a jejich srovnání na jednoduché úloze binární klasifikace
- Automatické symbolické derivování tensorů v Tensorflow a PyTorch
- Frameworky a podpora GPU