

KMI/UMIN Umělá inteligence

Deep Learning

2. Dopředné sítě a backpropagation

Jan Konečný

Katedra informatiky, PřF UP v Olomouci

5. října 2026

Minule...

Perceptron je lineární model se schodkovou aktivací.

Umí právě lineárně separabilní funkce a učí se opravným pravidlem.

Dnes potřebujeme dvě věci navíc:

1. skládat jednotky do *vrstev*, aby vznikla nelineární hranice;
2. umět spočítat gradient složené funkce, protože opravné pravidlo na vícevrstvý model nestačí.

Druhá věc je backpropagation — efektivní postup pro výpočet gradientu, nikoli architektura ani optimalizační metoda.

Skripta: kap. 2 *Dopředné neuronové sítě*

Definice

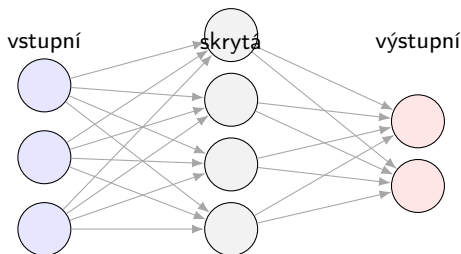
Definice 1 (Dopředná neuronová síť (FFNN))

FFNN je orientovaný acyklický graf s ohodnocenými hranami. Uzel představuje výpočetní jednotku, hrana vážené spojení.

- Uzly bez vstupních hran tvoří **vstupní vrstvu**.
- Uzly bez výstupních hran tvoří **výstupní vrstvu**.
- Ostatní uzly jsou **skryté**.
- Skryté a výstupní uzly mají aktivační funkci $f : \mathbb{R} \rightarrow \mathbb{R}$ a hodnotu b zvanou **bias**.

Informace postupuje od vstupu k výstupu bez rekurentních vazeb; výsledná funkce vzniká skládáním parametrizovaných transformací.

Vrstvená architektura



Model určují tři složky:

- struktura uzlů (agregace a aktivační funkce),
- topologie sítě,
- algoritmus učení, kterým se nastaví váhy.

Na rozdíl od perceptronu připouštíme *libovolnou* aktivační funkci — v dalším výkladu i polynomy a goniometrické funkce.

Vrstvená síť

Definice 2 (Vrstvená FFNN)

Skryté uzly jsou uspořádány do vrstev $H_1, \dots, H_k$ tak, že hrany vedou ze všech uzlů vstupní vrstvy do všech uzlů H_1 , ze všech uzlů H_i do všech uzlů H_{i+1} a ze všech uzlů H_k do všech uzlů výstupní vrstvy.

Pro tento speciální případ lze gradient zapsat maticovými vzorci, což využívají všechny knihovny. Obecná FFNN je ale libovolný acyklický graf — reziduální spojení, větvení i přeskočené vrstvy se do definice vejdou.

Bez nelinearity to nepude..

Nechť jsou vrstvy lineární, tedy $\mathbf{o}^{(1)} = \mathbf{x}\mathbf{W}_1$ a $\mathbf{o}^{(2)} = \mathbf{o}^{(1)}\mathbf{W}_2$. Pak

$$\mathbf{o}^{(2)} = \mathbf{x} (\mathbf{W}_1 \mathbf{W}_2) = \mathbf{x} \mathbf{W},$$

takže dvě vrstvy počítají totéž co jediná vrstva s maticí $\mathbf{W} = \mathbf{W}_1 \mathbf{W}_2$.

Důsledek

Hloubka sama o sobě nepřidává vyjadřovací sílu. Přidává ji až nelineární aktivace mezi vrstvami; hloubka pak umožňuje skládat jednodušší transformace do složitějších.

Proto se v dalším výkladu držíme diferencovatelných nelinearit — nejprve sigmoidy, později ReLU a jejích příbuzných.

Síť jako aproximátor

Klasická úloha teorie aproximace: reprodukovat funkci $F : \mathbb{R} \rightarrow \mathbb{R}$ vyhodnocením sady primitivních funkcí.

Příklad 3 (Taylorova řada jako síť)

$$F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \cdots + a_n(x - x_0)^n + \cdots$$

Uzly počítají $z \mapsto 1$, $z \mapsto z$, $\dots$, $z \mapsto z^n$; jedinými volnými parametry jsou koeficienty $a_0, \dots, a_n$. Výstupní uzel přichodí hodnoty sečte.

Váhy lze spočítat analyticky (členy Taylorovy řady), nebo — jak je u neuronových sítí obvyklé — učícím algoritmem.

Fourierova řada jako síť

$$F(x) = \sum_{i=0}^{\infty} (a_i \cos(ix) + b_i \sin(ix))$$

Síť se sinem jako primitivní funkcí realizuje konečný počet členů:

- $k_0, \dots, k_n$ škálují frekvence argumentů,
- $d_0, \dots, d_n$ jsou fázové posuvy (pro $d_0 = \pi/2$ vyjde $\sin(x + d_0) = \cos(x)$, takže kosinus není nutné implementovat zvlášť),
- $w_0, \dots, w_n$ jsou amplitudy.

Síť je obecnější než tradiční rozvoj: dovoluje necelé frekvence i fázové posuvy, které nejsou násobky $\pi/2$.

V čem se úloha učení liší

Zásadní rozdíl oproti Taylorovi a Fourierovi

Aproximovaná funkce F *není zadána explicitně*. K dispozici jsou pouze vstupně-výstupní příklady a algoritmus učení, který nastaví parametry tak, aby síť odpovídala známým hodnotám a zobecnila na nové vstupy.

Odtud plynou dvě otázky, které nás budou provázet celý kurz:

- Jak parametry najít? (optimalizace)
- Proč by měly fungovat i mimo trénovací data? (generalizace)

Trénovací množina a chybová funkce

Funkce sítě má typ $\mathbb{R}^n \rightarrow \mathbb{R}^m$. Trénovací množina je p dvojic $\langle \mathbf{x}_i, \mathbf{t}_i \rangle$, kde $\mathbf{t}_i$ je požadovaný výstup.

Prozatím měříme kvalitu kvadratickou chybou

$$E = \frac{1}{2} \sum_{i=1}^p \|\mathbf{o}_i - \mathbf{t}_i\|^2,$$

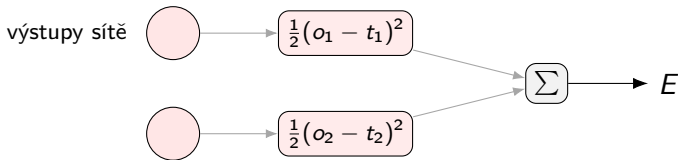
kde $\mathbf{o}_i$ je skutečný výstup sítě pro vstup $\mathbf{x}_i$.

Požadavek na model

Backpropagation potřebuje v každé iteraci gradient E . Funkce sítě proto musí být v místech výpočtu diferencovatelná; protože je složením funkcí jednotek, stačí diferencovatelnost aktivačních funkcí.

Chybu přidáme do grafu

Aby byl gradient chyby jen dalším průchodem sítě, rozšíříme graf o uzly, které chybu počítají:



Funkcí rozšířené sítě je přímo chybová funkce E . Jediné modifikovatelné parametry jsou váhy, takže E je spojitá a diferencovatelná funkce vah $w_1, \dots, w_\ell$ a lze ji minimalizovat gradientním sestupem

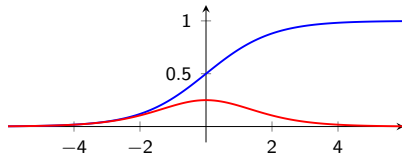
$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}.$$

Celý problém se tím redukuje na výpočet gradientu.

Schodkovou funkci vyměníme za sigmoidu

Schodková funkce podmínku nesplňuje. Nahradí ji **sigmolda** $\sigma : \mathbb{R} \rightarrow (0, 1)$,

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma'(x) = \sigma(x)(1 - \sigma(x)).$$



Modře σ , červeně σ' . Derivace je největší kolem nuly a v obou chvostech se blíží k nule — k tomu se vrátíme u mizejících gradientů.

Derivace sigmoidy a co z ní plyne

Maximum σ' je v nule a má hodnotu $\frac{1}{4}$:

$$\sigma'(0) = \sigma(0)(1 - \sigma(0)) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$

Při zpětném průchodu se derivace jednotlivých vrstev *násobí*. Pro L vrstev je tedy příspěvek aktivací nejvýše

$$\left(\frac{1}{4}\right)^L,$$

což pro $L = 10$ dává řádově 10^{-6} .

Mizející gradient

Vrstvy blízko vstupu proto dostávají zanedbatelný učicí signál. Je to jeden z důvodů, proč se od sigmoidy ve skrytých vrstvách ustoupilo; podrobně v přednášce o reprezentacích a aktivacích.

Řetězové pravidlo v síti s jedním neuronem

Příklad 4 (Ruční výpočet obou gradientů)

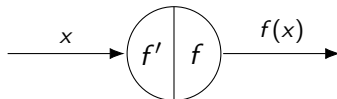
Pro vstup $x = 1$, skrytou aktivaci $h = \sigma(wx)$ s $w = 0,4$, lineární výstup $y = vh$ s $v = 2$ a cíl $t = 1$ vychází $h \doteq 0,5987$, $y \doteq 1,1974$ a $E = \frac{1}{2}(y - t)^2$. Pak

$$\frac{\partial E}{\partial v} = (y - t) h \doteq 0,1182, \quad \frac{\partial E}{\partial w} = (y - t) v h(1 - h) x \doteq 0,0949.$$

Týž chybový signál $(y - t)$ se zpětně násobí lokálními derivacemi jednotlivých operací. Backpropagation tento postup jen systematizuje pro libovolný graf.

B-diagram: uzel má dvě strany

Uzel rozdělíme na pravou a levou část:



- **pravá strana** počítá primitivní funkci uzlu,
- **levá strana** její derivaci pro tentýž vstup a uloží ji.

Agregaci (sumaci) a aktivaci je vhodné rozdělit do dvou uzlů: na každém uzlu se pak počítá jen jedna funkce.

Grafické podání podle R. Rojas: *Neural Networks: A Systematic Introduction*. Springer (1996), kap. 7.

Dva průchody sítí

Feed-forward

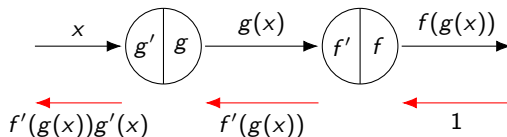
Informace přichází zleva. Každý uzel vyhodnotí primitivní funkci f (pravá strana) i derivaci f' (levá strana). Obojí uloží; doprava posílá jen $f(x)$.

Backpropagation

Graf se prochází v opačném směru. Do výstupního uzlu vstupuje konstanta 1; přichozí hodnota se v každém uzlu vynásobí uloženou derivací a posílá dál doleva.

Stačí rozlišit tři případy: složení funkcí, sčítání a vážené hrany.

První případ: složení funkcí



Zpětný průchod násobí procházející hodnotu uloženou derivací; ve vstupním uzlu tak vznikne derivace složené funkce. To je řetězové pravidlo zapsané graficky.

Druhý a třetí případ

Sčítání funkcí

Parciální derivace součtu podle každého vstupu je 1. Při zpětném průchodu se procházející hodnota *rozešle* beze změny do všech větví; tam, kde se cesty setkají, se hodnoty *sečtou*.

$$(f_1 + f_2)'(x) = f_1'(x) + f_2'(x)$$

Vážená hrana

V dopředném průchodu se informace násobí vahou w . Při zpětném průchodu se procházející derivace násobí toutéž vahou.

$$x \mapsto wx, \quad \delta \mapsto w\delta$$

Tři pravidla stačí na libovolnou dopřednou síť: skládání, větvení se sčítáním a vážené hrany.

Formulace

Nechť má síť jeden reálný vstup x a síťovou funkci F . Derivaci $F'(x)$ spočteme ve dvou fázích:

Feed-forward

Vstup x se zavede do sítě. Každý uzel vyhodnotí primitivní funkci i její derivaci; derivace se uloží.

Backpropagation

Do výstupního uzlu se zavede konstanta 1 a síť se prochází pozpátku. Příchozí hodnota se v uzlu násobí uloženou derivací a posílá doleva. Hodnota shromážděná ve vstupním uzlu je $F'(x)$.

Pro více vstupních proměnných platí totéž: jediný zpětný průchod vrátí v i -tém vstupním uzlu $\partial F / \partial x_i$. Není nutné spouštět výpočet zvlášť pro každou proměnnou.

Správnost algoritmu

Věta 5 (Korektnost backpropagation)

Algoritmus backpropagation správně počítá derivaci funkce sítě F vzhledem ke vstupu x .

Idea důkazu (indukcí podle počtu uzlů)

Nechť je k poslednímu uzlu s primitivní funkcí φ připojeno m uzlů s výstupy $F_1(x), \dots, F_m(x)$, takže $F(x) = \varphi(w_1 F_1(x) + \dots + w_m F_m(x))$ a

$$F'(x) = \varphi'(s)(w_1 F'_1(x) + \dots + w_m F'_m(x)), \quad s = \sum_i w_i F_i(x).$$

Podgraf ke každému F_i má nejvýše n uzlů, takže podle indukčního předpokladu vrátí $F'_i(x)$. Poslední uzel se inicializuje hodnotou $\varphi'(s)$ a rozešle ji po hranách s vahami w_i ; součet větví dává právě $F'(x)$.

Gradient podle vah

Rozšíříme graf o uzly počítající chybu, takže funkcí rozšířené sítě je přímo E . Dopředný průchod navíc ukládá výstup o_i každého uzlu.

Pro váhu w_{ij} na hraně z uzlu i do uzlu j platí

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j,$$

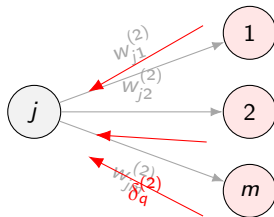
kde δ_j je zpětně šířená chyba v uzlu j . Sestup pak provede korekci

$$\Delta w_{ij} = -\gamma o_i \delta_j.$$

Tímto krokem se z backpropagation stává *metoda učení*: samotný algoritmus jen počítá gradient.

Zpětné šíření do skryté vrstvy

Skrytý uzel j ovlivňuje chybu přes *všechny* výstupní uzly, do nichž vede hrana. Příspěvky se proto sčítají:



$$\delta_j^{(1)} = o_j^{(1)}(1 - o_j^{(1)}) \sum_{q=1}^m w_{jq}^{(2)} \delta_q^{(2)}$$

Je to druhý případ z B-diagramu (sčítání větví) aplikovaný na uzel, z něhož vede více hran dopředu.

Vrstvená síť maticově

Pro síť s n vstupy, k skrytými a m výstupními jednotkami platí

$$\mathbf{o}^{(1)} = \sigma(\bar{\mathbf{o}} \overline{\mathbf{W}}_1), \quad \mathbf{o}^{(2)} = \sigma(\bar{\mathbf{o}}^{(1)} \overline{\mathbf{W}}_2),$$

kde pruh značí rozšíření o konstantní jedničku (bias jako váha).

Zpětný průchod používá diagonální matice uložených derivací $\mathbf{D}_1, \mathbf{D}_2$ a vektor odchylek $\mathbf{e} = \mathbf{o}^{(2)} - \mathbf{t}$:

$$\delta^{(2)} = \mathbf{D}_2 \mathbf{e}, \quad \delta^{(1)} = \mathbf{D}_1 \mathbf{W}_2 \delta^{(2)},$$

$$\Delta \overline{\mathbf{W}}_2^T = -\gamma \delta^{(2)} \mathbf{o}^{(1)}, \quad \Delta \overline{\mathbf{W}}_1^T = -\gamma \delta^{(1)} \bar{\mathbf{o}}.$$

Pro ℓ vrstev platí též rekurze

$$\delta^{(\ell)} = \mathbf{D}_\ell \mathbf{e}, \quad \delta^{(i)} = \mathbf{D}_i \mathbf{W}_{i+1} \delta^{(i+1)}, \quad i = 1, \dots, \ell - 1.$$

Potřebné operace jsou jen násobení matice vektorem — proto se síť počítají dobře na GPU.

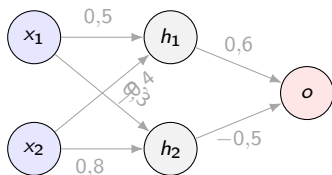
Čtyři kroky učení vrstvené sítě

1. **Feed-forward:** vypočítat a uložit $\mathbf{o}^{(1)}$, $\mathbf{o}^{(2)}$ i derivace aktivací.
2. **Zpětný průchod do výstupní vrstvy:**
$$\delta_j^{(2)} = o_j^{(2)}(1 - o_j^{(2)})(o_j^{(2)} - t_j).$$
3. **Zpětný průchod do skryté vrstvy:**
$$\delta_j^{(1)} = o_j^{(1)}(1 - o_j^{(1)}) \sum_q w_{jq}^{(2)} \delta_q^{(2)}.$$
4. **Aktualizace vah** ve směru záporného gradientu.

Pořadí operací

Korekce se provádí až po výpočtu všech zpětně šířených chyb. Jinak by aktualizace neodpovídaly zápornému gradientu.

Demonstrace: jeden krok v síti 2-2-1



biasy $b_1 = 0,1$, $b_2 = -0,2$, $b_o = 0,2$;

aktivace σ ve všech jednotkách

Vstup $\mathbf{x} = (1, 0)$, cíl $t = 1$, učicí krok $\gamma = 0,5$.

Dopředný průchod

$$\text{net}_1 = 0,6 \quad h_1 = \sigma(0,6) \doteq 0,6457$$

$$\text{net}_2 = 0,1 \quad h_2 = \sigma(0,1) \doteq 0,5250$$

$$\text{net}_o = 0,3249 \quad o \doteq 0,5805$$

Chyba je $E = \frac{1}{2}(o - t)^2 \doteq 0,0880$.

Demonstrace: zpětný průchod a korekce

Zpětně šířené chyby

$$\delta_o = (o - t) o(1 - o) \doteq -0,1022$$

$$\delta_1 = h_1(1 - h_1) v_1 \delta_o \doteq -0,0140$$

$$\delta_2 = h_2(1 - h_2) v_2 \delta_o \doteq 0,0127$$

Gradienty vah do výstupu

$$\partial E / \partial v_1 = \delta_o h_1 \doteq -0,0660$$

$$\partial E / \partial v_2 = \delta_o h_2 \doteq -0,0536$$

Nové váhy po kroku $\gamma = 0,5$

váha	před	po
v_1	0,6000	0,6330
v_2	-0,5000	-0,4732
b_o	0,2000	0,2511
w_{11}	0,5000	0,5070
w_{21}	-0,4000	-0,4000
w_{12}	0,3000	0,2936
w_{22}	0,8000	0,8000

Chyba klesla z 0,0880 na 0,0791.

Váhy w_{21} a w_{22} se nezměnily: jejich vstup byl nulový, takže $\partial E / \partial w = o_i \delta_j = 0$. Nulový vstup nenese učicí signál.

Dávkové a online učení

Při p vzorech lze korekce sečíst přes celou množinu (*dávkové*, offline učení), nebo aktualizovat váhy po každém vzoru (*online* učení).

Dávkově

Přesný směr gradientu, ale jedna epocha znamená mnoho dopředných průchodů před jedinou korekcí.

Online

Kroky nesledují přesný gradient, ale při náhodném pořadí vzorů kolem něj oscilují. Šum navíc může pomoci opustit mělké lokální minimum.

Kompromisem je mini-batch; podrobně v příští přednášce o optimalizaci.

Jeden trénovací krok v pseudokódu

Algorithm 1: Krok učení vrstvené sítě

Input: dávka dvojic $(\mathbf{x}, \mathbf{t})$, váhy $\mathbf{W}$, učicí krok γ

foreach dvojice $(\mathbf{x}, \mathbf{t})$ v dávce **do**

 dopředný průchod: spočti a ulož výstupy vrstev i derivace aktivací;

 spočti chybu E a odchylku $\mathbf{e} = \mathbf{o} - \mathbf{t}$;

 zpětný průchod: δ ve výstupní vrstvě, pak rekurzivně vrstvy zpět;

 akumuluj gradienty $\partial E / \partial w_{ij} = o_i \delta_j$;

end

$\mathbf{W} \leftarrow \mathbf{W} - \gamma \nabla E$;

Rozdíl mezi dávkovým a online učním je jen v tom, kde stojí poslední řádek — zda po celé množině, po dávce, nebo po každém vzoru.

Proč se v uzlu sčítá

Dosud byla informace při zpětném průchodu *lokální*: hodnota, která přišla hranou dopředu, se toutéž hranou vrací zpět.

Zkusme integrační funkci nahradit součinem, tedy $f(ab)$ pro vstupy a a b . Zpětný průchod pak potřebuje po horní hraně poslat b a po dolní a — každá hodnota přišla jinou cestou a lokalita je pryč.

Věta 6 (Lokalita vynucuje sčítání)

V uzlu s n vstupy zaručují lokalitu právě integrační funkce tvaru

$$I(x_1, \dots, x_n) = F_1(x_1) + F_2(x_2) + \dots + F_n(x_n) + C.$$

Vážený součet tedy není libovolná volba: je to (až na jednorozměrné transformace vstupů) jediná možnost, jak udržet učení lokální.

Cena za rychlost: paměť

Zpětný průchod potřebuje hodnoty uložené při dopředném průchodu — výstupy uzlů a derivace aktivací.

- Dopředný a zpětný průchod mají srovnatelnou výpočetní cenu.
- Paměť ale roste s počtem uložených aktivací, tedy s dávkou, hloubkou i délkou sekvence.
- Odtud techniky jako přepočítávání aktivací (*activation checkpointing*), o nichž bude řeč v přednášce o praktickém trénování.

Historicky byl algoritmus popsán několikrát nezávisle a pro trénování sítí jej prosadila až práce z roku 1986.

- H. J. Kelley: Gradient Theory of Optimal Flight Paths. *ARS Journal* 30(10), 947–954 (1960). —
A. E. Bryson: A Gradient Method for Optimizing Multi-Stage Allocation Processes (1962). —
D. E. Rumelhart, G. E. Hinton, R. J. Williams: Learning Representations by Back-Propagating Errors. *Nature* 323, 533–536 (1986).

Numerická kontrola gradientu

Implementaci gradientu ověříme centrální diferencí: pro jednu váhu w

$$\frac{\partial E}{\partial w} \approx \frac{E(w + \varepsilon) - E(w - \varepsilon)}{2\varepsilon}, \quad \varepsilon \approx 10^{-5}.$$

Porovnáváme *relativní* rozdíl

$$\frac{|g_{\text{analyt}} - g_{\text{num}}|}{\max(|g_{\text{analyt}}|, |g_{\text{num}}|, 10^{-8})},$$

který by měl vyjít řádově 10^{-6} a menší.

Na co si dát pozor

Test provádějte v dvojité přesnosti, s vypnutým dropoutem a mimo body nespojitě derivace (například přesně v nule u ReLU).

Cvičení k přednášce 2

1. ★ Nakreslete výpočetní graf sítě se dvěma vstupy, dvěma skrytými neurony a jedním výstupem.
2. ★ Určete počet vah a biasů sítě s vrstvami 10–20–5–1.
3. ★ Odvoďte derivaci sigmoidy ve tvaru $\sigma(x)(1 - \sigma(x))$.
4. ★★ Odvoďte gradient $E = \frac{1}{2}(wx - t)^2$ podle w dvěma způsoby a porovnejte je.
5. ★★ Pro příklad z přednášky proveďte jeden krok gradientního sestupu s $\eta = 0,1$.
6. ★★ Vysvětlete, proč složení několika lineárních vrstev bez aktivace zůstává lineární funkcí.
7. ★★ Sestavte graf funkce $f(a, b) = (ab + a)^2$ a spočtěte gradienty podle a a b .
8. ★★★ Navrhněte numerickou kontrolu gradientu jediné váhy a zvolte vhodné ε .
9. ★★★ Porovnejte cenu dopředného a zpětného průchodu a uveďte, co musí implementace uchovávat v paměti.