Kapitola 2

Dopředné neuronové sítě

Dopředná neuronová síť (feedforward neural network, FFNN) je orientovaný acyklický výpočetní graf. Informace postupuje od vstupu k výstupu bez rekurentních vazeb a výsledná funkce vzniká skládáním parametrizovaných transformací. Tato konstrukce tvoří základ klasifikačních, regresních i mnoha složitějších modelů.

Síť perceptronů řešící XOR v předchozí kapitole je nejjednodušším příkladem FFNN.

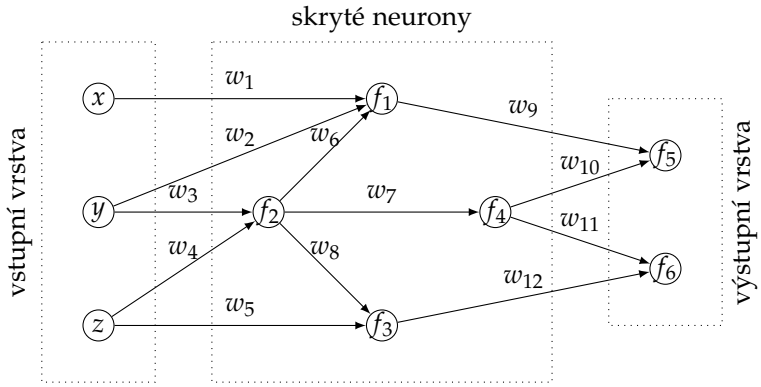
Definice 2.1: Dopředná neuronová síť (FFNN)

FFNN je orientovaný acyklický graf s ohodnocenými hranami. Uzel představuje výpočetní jednotku a hrana vážené spojení mezi dvěma jednotkami.

- Množina uzlů bez vstupních hran se nazývá vstupní vrstva,
- Množina uzlů bez výstupních hran se nazývá výstupní vrstva,
- Uzly, které nejsou ve vstupní ani výstupní vrstvě, se nazývají skryté.
- Skryté uzly a uzly výstupní vrstvy mají dánu aktivační funkci $f : \mathbb{R} \to \mathbb{R}$ a hodnotu b nazývanou bias.

Typické umělé neuronové sítě mají strukturu zobrazenou na obr. 2.1. Tuto síť lze považovat za implementaci funkce F vyhodnocované v bodě (x, y, z). Uzly implementují primitivní funkce $f_1, f_2, \ldots, f_6$, jejichž složením vzniká funkce F, nazývaná funkce sítě. Různé volby vah $w_1, w_2, \ldots, w_{12}$ produkují různé funkce sítě. Každý model umělých neuronových sítí proto určují tři prvky:

- struktura uzlů,
- topologie sítě,



Obrázek 2.1: Typická architektura dopředné neuronové sítě se vstupní, skrytou a výstupní vrstvou.

- algoritmus učení použitý k nalezení vah sítě.

Libovolná aktivační funkce: Na rozdíl od předchozí kapitoly, kde perceptron používal schodkovou funkci, zde připustíme libovolné aktivační funkce – v následující části např. polynomy a goniometrické funkce.

2.1 Aproximace funkcí

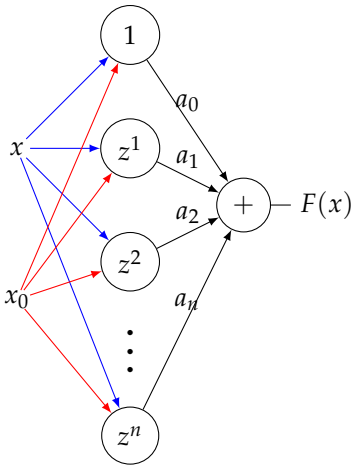
Klasickým problémem teorie aproximace je reprodukovat danou funkci $F : \mathbb{R} \to \mathbb{R}$ přesně nebo přibližně vyhodnocením sady primitivních funkcí – typicky polynomů nebo Fourierových řad.

Příklad 2.1

Taylorova řada pro funkci F, která je aproximována kolem bodu x_0, je

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

kde konstanty $a_0, \ldots, a_n$ závisí na funkci F a jejích derivacích v x_0.
Obr. 2.2 ukazuje polynomiální aproximaci jako síť funkcí. Uzly počítají primitivní funkce $z \mapsto 1, z \mapsto z^1, \ldots, z \mapsto z^n$; jedinými volnými parametry jsou konstanty $a_0, \ldots, a_n$. Výstupní uzel příchozí hodnoty sečte a vydá hodnotu polynomu. Váhy sítě zde lze vypočítat analyticky jako prvních $n + 1$ členů Taylorovy řady funkce F, nebo – jak je v oblasti NN obvyklé – učicím algoritmem.



Obrázek 2.2: Polynomiální aproximace realizovaná sítí primitivních funkcí.

Příklad 2.2

Obr. 2.3 ukazuje, jak lze Fourierovu řadu implementovat jako neuronovou síť. Pokud má být funkce F rozvinuta jako Fourierova řada, má tvar

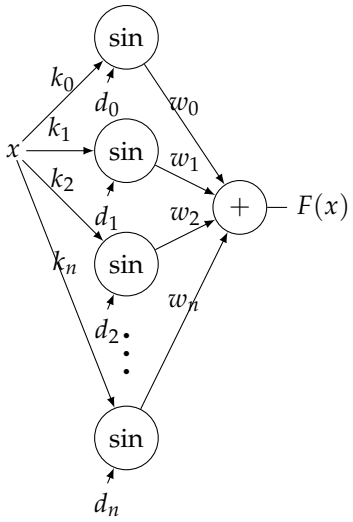
$$F(x) = \sum_{i=0}^{\infty} (a_i \cos(ix) + b_i \sin(ix)).$$

Síť se sinem jako primitivní funkcí realizuje konečný počet členů tohoto výrazu. V obrázku:

- Konstanty $k_0, \ldots, k_n$ představují frekvenční škálovací faktory pro argumenty sinusových funkcí.
- Konstanty $d_0, \ldots, d_n$ jsou fázové posuvy (při $d_0 = \frac{\pi}{2}$ dostáváme $\sin(x + d_0) = \cos(x)$), takže kosinus nemusí být v síti explicitně implementován.
- Konstanty $w_0, \ldots, w_n$ jsou amplitudy Fourierových členů.

Síť je v tomto případě obecnější než tradiční Fourierův rozvoj, protože umožňuje použití ne-celočíselných frekvencí a fázových posuvů, které nejsou jednoduchými násobky $\pi/2$.

Na rozdíl od Taylorovy nebo Fourierovy řady není aproximovaná funkce F zadána explicitně, ale pouze vstupně-výstupními příklady. Algoritmus učení nastavuje parametry tak, aby síť odpovídala známým hodnotám a zobecnila na nové vstupy.



Obrázek 2.3: Realizace Fourierovy řady sítí sinusových jednotek.

2.2 Trénování FFNN

Obecná aktivační funkce a více výstupních jednotek umožňují jiné výstupy než binární hodnotu. Funkce sítě má obecně typ $\mathbb{R}^n \to \mathbb{R}^m$, kde n je počet vstupních a m počet výstupních jednotek. Pro trénování tedy nestačí zadat dvě množiny vstupů (P a N) jako v předchozí kapitole.

Namísto toho zadáváme trénovací množinu sestávající z p dvojic $\langle \boldsymbol{x}_i, \boldsymbol{t}_i \rangle$ (pro $i = 1 \dots p$), kde $\boldsymbol{x}_i \in \mathbb{R}^n$ je vstup sítě a $\boldsymbol{t}_i \in \mathbb{R}^m$ její požadovaný výstup.

Kritérium kvality predikce vyjadřuje chybová funkce, označovaná také jako ztrátová funkce. Prozatím postačí následující definice:

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

kde $\boldsymbol{t}_i$ je požadovaný výstup a $\boldsymbol{o}_i$ je skutečný výstupní vektor sítě pro vstup $\boldsymbol{x}_i$.

Její derivace má tvar

$$E' = \sum_{i=1}^{p} \|\boldsymbol{o}_i - \boldsymbol{t}_i\|.$$

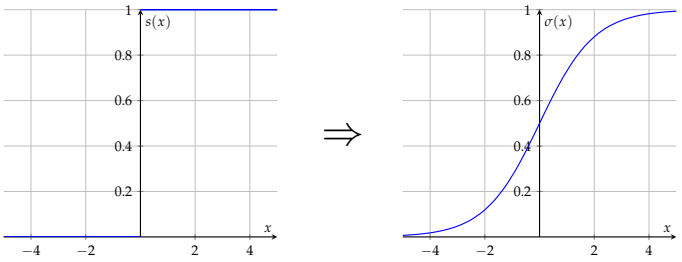
Algoritmus backpropagation potřebuje v každé iteraci gradient ztrátové funkce. Použité funkce proto musí být v místech výpočtu diferencovatelné. Protože E je sumace kvadrátů rozdílů zadaných

vektorů $\boldsymbol{t}_i$ a výsledků funkce sítě, musí tyto vlastnosti mít funkce sítě. A protože funkce sítě je složením funkcí jednotlivých jednotek, stačí, aby spojité a diferencovatelné byly aktivační funkce těchto jednotek.

Schodková aktivační funkce tento požadavek nesplňuje. Ve zbytku kapitoly ji proto nahrazuje sigmoidální funkce (též sigmoida) $\sigma : \mathbb{R} \to (0,1)$

$$\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}, \qquad \text{pro } x \in \mathbb{R}. \tag{2.1}$$

Sigmoida $\sigma(x)$ je hladkou aproximací schodkové funkce $s_0(x)$ z rovnice (1.2) v kapitole 1.



Derivace sigmoidy vzhledem k x, potřebná při zpětném průchodu, je

$$\frac{d}{dx}\sigma(x) = \frac{\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^2} = \sigma(x)(1-\sigma(x)). \tag{2.2}$$

Příklad 2.3: Řetězové pravidlo v síti s jedním neuronem

Pro $x = 1$, 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 $E = \frac{1}{2}(y-t)^2$. Gradient výstupní váhy je

$$\frac{\partial E}{\partial v} = (y-t)h \doteq 0{,}1182,$$

zatímco pro skrytou váhu

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

Stejný chybový signál se tedy zpětně násobí lokálními derivacemi jednotlivých operací.

Připomínka
Funkce f je spojitá v bodě a, pokud pro každé $\varepsilon > 0$ existuje $\delta > 0$ tak, že pro všechna x splňující $|x-a| < \delta$ platí $|f(x) - f(a)| < \varepsilon$.
Funkce f je spojitá, pokud je spojitá v každém bodě svého definičního oboru.
Funkce f je diferencovatelná v bodě a, pokud existuje limita

$$\lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$$

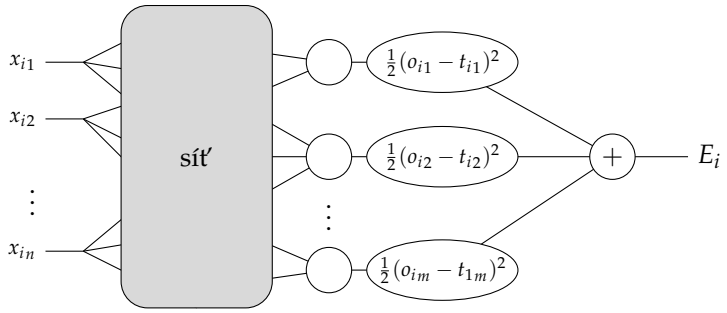
a tato limita je konečná. Tato limita se nazývá derivace funkce f v bodě a a značí se $f'(a)$. Funkce f je diferencovatelná, pokud je diferencovatelná v každém bodě svého definičního oboru.

Geoffrey Hinton (1947)
Spolu s Davidem Rumelhartem a Ronaldem Williamsem ukázal, jak prakticky trénovat vícevrstvé sítě zpětným šířením chyby. Později zásadně přispěl k obnově zájmu o hluboké učení a k úspěchu AlexNetu.

[Kelley1960] H. J. Kelley.
Gradient theory of optimal flight paths.
ARS Journal, 30, 947–954. (1960)

[Bryson1962] A. E. Bryson.
A gradient method for optimizing multi-stage allocation processes.
Harvard Symposium on Digital Computers and Their Applications. (1962)

[Rumelhart1986] D. E. Rumelhart, G. E. Hinton, R. J. Williams.
Learning representations by back-propagating errors.
Nature 323.6088 (1986): 533-536.



Obrázek 2.4: Rozšíření sítě pro výpočet chybové funkce

2.3 Backpropagation

Backpropagation (BP) je algoritmus pro trénování neuronových sítí: šíří chybu z výstupní vrstvy zpět do skrytých vrstev a podle ní upravuje váhy tak, aby minimalizoval celkovou chybu. Byl objeven v 60. letech [Kelley1960], [Bryson1962] a znovu-objeven v 80. letech [Rumelhart1986].

Sada vah minimalizující chybovou funkci je řešením problému učení.

Prvním krokem minimalizace je rozšíření sítě tak, aby chybovou funkci počítala automaticky (obrázek 2.4). Každý j-tý výstupní uzel sítě připojíme k uzlu vyhodnocujícímu funkci $\frac{1}{2}(o_{ij} - t_{ij})^2$, kde o_{ij} a t_{ij} označují j-tou komponentu výstupního vektoru $\boldsymbol{o}_i$ a cíle $\boldsymbol{t}_i$. Výstupy těchto přidaných m uzlů sečte další uzel s výstupem E_i. Stejné rozšíření provedeme pro každý vzor $\boldsymbol{t}_i$. Poslední uzel sečte všechny kvadratické chyby na $E_1 + \cdots + E_p$. Funkcí takto rozšířené sítě je chybová funkce E původní sítě.

Máme tak síť počítající celkovou chybu pro danou trénovací množinu. Jedinými modifikovatelnými parametry pro snížení kvadratické chyby E jsou váhy. Protože rozšířená síť počítá E čistě složením funkcí uzlů, je E spojitá a diferencovatelná funkce ℓ vah $w_1, w_2, \ldots, w_\ell$ v síti. Lze ji tedy minimalizovat gradientním sestupem, který vyžaduje gradient

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_\ell} \right).$$

Každá váha je upravena inkrementem

$$\delta w_i = -\gamma \frac{\partial E}{\partial w_i}, \quad \text{pro } i = 1, \ldots, \ell,$$

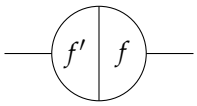
kde učící konstanta γ určuje velikost kroku každé iterace v opačném směru gradientu.

Celý problém se tak redukuje na výpočet gradientu funkce sítě vzhledem k jejím váhám. Po jeho výpočtu lze iterativní aktualizací vah hledat minimum chybové funkce, kde $\nabla E = 0$.

2.4 Derivace funkce sítě

Nyní na chvíli zapomeňme na trénovací množiny a učení sítí a představme metodu efektivního výpočtu gradientu *jednodimenzionální* sítě vzhledem k jejím vahám.

Síť je ekvivalentní řetězci složených funkcí, hlavní roli proto bude hrát řetězcové pravidlo diferenciálního kalkulu. Zohledníme to kompozitní strukturou uzlů: každý uzel sestává z levé a pravé strany, jak ukazuje obr. 2.5.

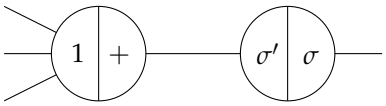


Obrázek 2.5: Rozdělení výpočetního uzlu na hodnotu funkce a její lokální derivaci.

Tento typ reprezentace nazýváme B-diagram (jako *backpropagation diagram*).

Pravá strana počítá primitivní funkci přiřazenou uzlu, levá strana derivaci této funkce pro tentýž vstup.

Agregační funkci (sumaci) lze oddělit od aktivační funkce rozdělením každého uzlu na dvě části, jak ukazuje obr. 2.6. První uzel počítá součet příchozích vstupů, druhý aktivační funkci σ. Derivace σ je σ' a parciální derivace součtu n argumentů vzhledem k libovolnému z nich je 1. Toto rozdělení zjednodušuje diskusi: na každém uzlu se počítá jen jedna funkce, nikoliv dvě.



Obrázek 2.6: Oddělení agregační funkce od aktivační funkce.

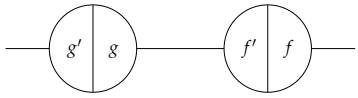
Síť je vyhodnocována ve dvou krocích:

- krok feed-forward: informace přichází zleva a každý uzel vyhodnotí primitivní funkci f na pravé straně a derivaci f' na levé straně. Oba výsledky se uloží v uzlu, ale doprava se přenáší jen výsledek pravé strany.

- krok backpropagation: graf se prochází v opačném směru a využívá hodnoty uložené při dopředném průchodu. Stačí rozlišit tři základní případy.

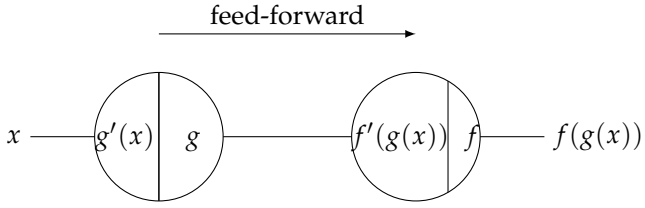
První případ: složení funkcí

B-diagram na obr. 2.7 obsahuje pouze dva uzly.



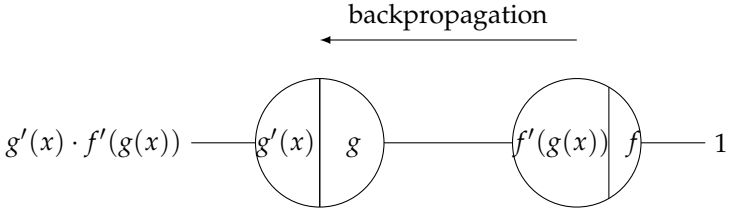
Obrázek 2.7: B-diagram sítě pro složení dvou funkcí.

V kroku feed-forward slouží příchozí informace jako argument pro vyhodnocení primitivní funkce uzlu a její derivace; síť tak počítá složení funkcí f a g. Obrázek níže ukazuje stav sítě po tomto kroku.



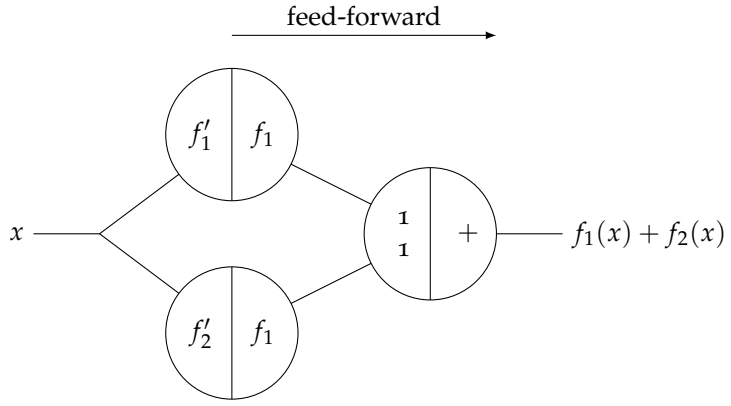
Výstupní jednotka vydala výsledek složení funkcí a každá jednotka má na levé straně uloženou informaci.

V kroku zpětné propagace je vstupem z pravé strany sítě konstanta 1. Příchozí informace do uzlu se násobí hodnotou uloženou na jeho levé straně a výsledek se přenáší do další jednotky vlevo; nazýváme jej procházející hodnota uzlu. Obrázek níže ukazuje konečný výsledek kroku zpětné propagace $f'(g(x))g'(x)$, tedy derivaci složení $f(g(x))$.

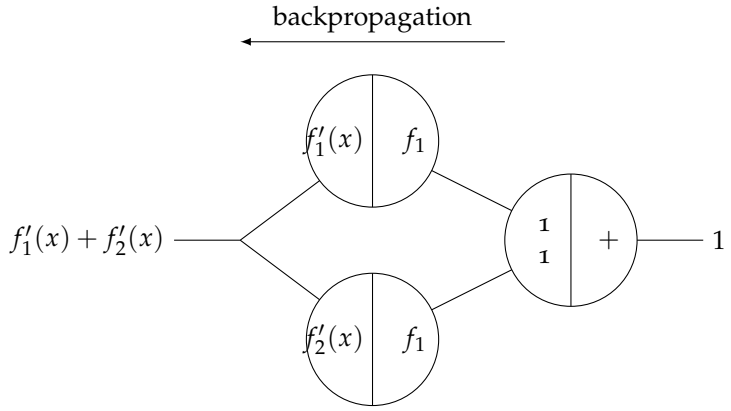


Druhý případ: sčítání funkcí

Dalším případem je sčítání dvou primitivních funkcí. Obrázek níže ukazuje síť pro výpočet součtu funkcí f_1 a f_2; sčítání obstarává přídavný uzel.



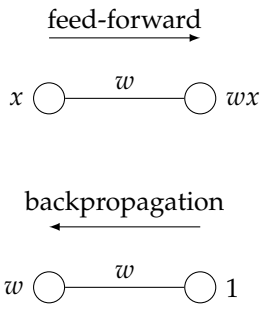
Parciální derivace sčítací funkce vzhledem k libovolnému ze dvou vstupů je 1. V kroku feed-forward síť počítá $f_1(x) + f_2(x)$. V kroku zpětné propagace je zprava přivedena konstanta 1 a distribuuje se všemi příchozími hranami do jednotek vlevo; kde se setkají dvě cesty, procházející hodnoty se sčítají. Obrázek níže ukazuje výsledek $f_1'(x) + f_2'(x)$, tedy derivaci součtu $f_1 + f_2$ v bodě x. Jednoduchou indukcí lze stejně zpracovat součet libovolného počtu funkcí.



Třetí případ: váhové hrany

Váhové hrany by šlo zpracovat jako složení funkcí, ale jde to jednodušeji. V kroku feed-forward je příchozí informace x násobena

váhou hrany w; výsledkem je wx. Při zpětné propagaci se procházející derivace násobí stejnou vahou. Oba směry zachycuje obr. 2.8.



Obrázek 2.8: Tok hodnoty a derivace přes hranu s vahou w.

Následující formulace úplného algoritmu zároveň umožňuje indukční důkaz pro libovolnou dopřednou síť s diferencovatelnými aktivačními funkcemi. Nejprve se uvažuje síť s jedním vstupním a jedním výstupním uzlem.

2.5 Algoritmus Backpropagation

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

Feed-forward: vstup x je zaveden do sítě. Primitivní funkce u uzlů a jejich derivace jsou vyhodnoceny na každém uzlu. Derivace jsou uloženy.

Backpropagation: konstanta 1 je zavedena do výstupního uzlu a síť je provozována pozpátku. Příchozí informace k uzlu je doplněna a výsledek je násoben hodnotou uloženou v levé části uzlu. Výsledek je přenášen doleva od uzlu. Výsledek shromážděný u vstupního uzlu je derivace síťové funkce F vzhledem k x.

Následující tvrzení ukazuje, že algoritmus je správný.

Věta 2.4

Algoritmus Backpropagation správně počítá derivaci funkce sítě F vzhledem k vstupu x.

Důkaz: pro zájemce

Algoritmus již byl ověřen pro sériové a paralelní zapojení uzlů i pro vážené hrany. Indukční předpoklad nyní říká, že pracuje pro každou FFNN s nejvýše n uzly. B-diagram na obr. 2.9 obsahuje $n+1$ uzlů. Po dopředném průchodu je výstupem hodnota síťové funkce $F(x)$. Nechť je k poslednímu uzlu připojeno m uzlů s výstupy $F_1(x), \ldots, F_m(x)$ a nechť jeho primitivní funkci označuje φ. Potom

$$F(x) = \varphi(w_1 F_1(x) + w_2 F_2(x) + \cdots + w_m F_m(x)).$$

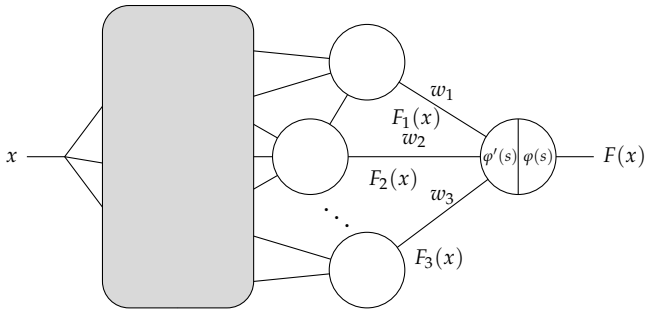
Derivace F v bodě x je tedy takováto:

$$F'(x) = \varphi'(s)(w_1 F_1'(x) + w_2 F_2'(x) + \cdots + w_m F_m'(x)),$$

kde $s = w_1 F_1(x) + \cdots + w_m F_m(x)$. Podgraf všech cest od vstupu k uzlu s výstupem $F_i(x)$ má nejvýše n uzlů, takže podle indukčního předpokladu zpětný průchod spočítá $F_i'(x)$. Po inicializaci posledního uzlu hodnotou $\varphi'(s)$ přispěje i-tá větev výrazem $w_i F_i'(x)\varphi'(s)$. Součet všech větví proto dává

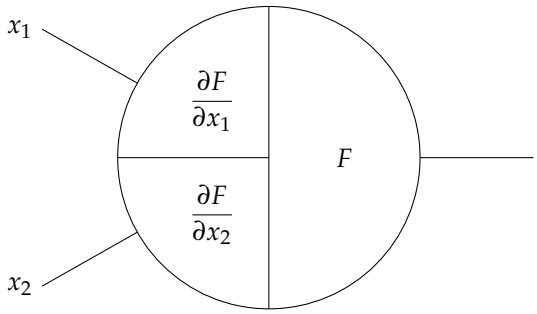
$$\varphi'(s)(w_1 F_1'(x) + w_2 F_2'(x) + \ldots + w_m F_m'(x)),$$

což je derivace F vyhodnocená v x. Faktory $w_1\varphi'(s), \ldots, w_m\varphi'(s)$ vzniknou tak, že se výstupní uzel inicializuje hodnotou 1, ta se v uzlu vynásobí uloženou derivací $\varphi'(s)$ a výsledek se rozešle zpět po hranách s vahami $w_1, \ldots, w_m$. Zpětný průchod tedy provádí právě operace požadované řetězovým pravidlem. Algoritmus je proto správný i pro síť s $n+1$ uzly, čímž je indukční důkaz uzavřen.



Obrázek 2.9: Poslední uzel v indukčním důkazu správnosti backpropagation.

Z předchozí analýzy vyplývá, že všechny vstupy do uzlu jsou sečteny před výpočtem jednorozměrné aktivační funkce. Lze uvažovat i aktivační funkce f více proměnných; levá strana jednotky pak uchovává parciální derivace f vzhledem ke každé proměnné. Obr. 2.10 ukazuje příklad pro funkci f dvou proměnných x_1 a x_2, dodaných dvěma hranami. V kroku zpětné propagace se každá uložená parciální derivace násobí procházející hodnotou uzlu a přenáší doleva vlastní hranou. Zpětná propagace tedy funguje i v tomto obecnějším případě.



Obrázek 2.10: Uložení parciálních derivací v uzlu s více vstupy.

Backpropagation funguje také pro více vstupních proměnných. Pro síťovou funkci $F(x_1, x_2)$ obsahuje graf všechny cesty od obou vstupů k výstupu. Jediný zpětný průchod přes celý graf vrátí v prvním vstupním uzlu $\partial F/\partial x_1$ a ve druhém $\partial F/\partial x_2$. Není nutné spouštět samostatný výpočet pro každou proměnnou.

2.6 Učení se s Backpropagation

Při učení se minimalizuje chybová funkce E závislá na vahách sítě. Dopředný průchod proto ukládá také výstup každého uzlu. Zpětný průchod rozšířeným grafem pak poskytne derivace podle všech vah. Pro jednu konkrétní váhu w_{ij}, jejíž hrana vede od i-tého k j-tému

uzlu, platí následující vztah. Tuto váhu lze považovat za vstupní kanál podsítě obsahující všechny cesty od w_{ij} k jedinému výstupnímu uzlu sítě. Do podsítě byla ve fázi feed-forward přivedena informace $o_i w_{ij}$, kde o_i je uložený výstup jednotky i. Krok backpropagation vypočítá gradient chybové funkce E vzhledem k tomuto vstupu, tzn. $\partial E / \partial o_i w_{ij}$. Protože ve fázi zpětné propagace je o_i konstanta, dostáváme

$$\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}}.$$

Shrnutí: krok backpropagation probíhá obvyklým způsobem; všechny pod-sítě definované jednotlivými vahami lze zpracovat současně. Navíc u každého uzlu i ukládáme výstup o_i z fáze feed-forward a kumulativní výsledek zpětného výpočtu až do tohoto uzlu – tzv. zpětně propagovanou chybu.

Označí-li se zpětně šířená chyba v uzlu j symbolem δ_j, má parciální derivace E podle w_{ij} tvar

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

Po spočítání všech parciálních derivací provedeme gradientní sestup korekcí každé váhy w_{ij} podle vztahu $\Delta w_{ij} = -\gamma o_i \delta_j$. Tento korekční krok mění algoritmus backpropagation v metodu učení NN.

Uvedený grafický důkaz platí pro libovolné feed-forward topologie.

2.7 Vrstvené FFNN

Důležitým speciálním případem dopředných sítí jsou vrstvené sítě s jednou nebo více skrytými vrstvami. Pro ně lze odvodit explicitní maticové vzorce pro gradienty vah. Zpětně šířená chyba přiřazená každému uzlu přitom zabrání opakování stejných mezivýpočtů.

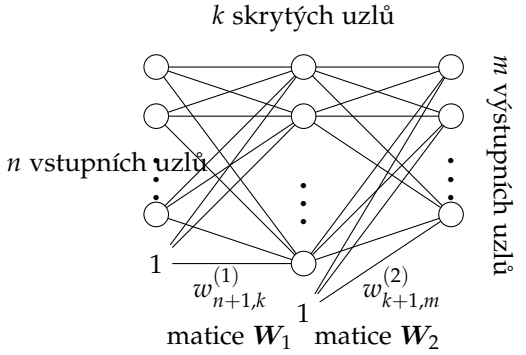
Definice 2.2

Vrstvená FFNN je FFNN, kde jsou skryté uzly organizovány do tzv. skrytých vrstev $H_1, H_2, \ldots, H_k$.

- Hrany ze všech uzlů vstupní vrstvy vedou právě do všech uzlů první skryté vrstvy H_1,
- hrany ze všech uzlů první skryté vrstvy H_1 vedou právě do všech uzlů druhé skryté vrstvy H_2, atd.,
- hrany ze všech uzlů poslední skryté vrstvy H_k vedou právě do všech uzlů výstupní O.

2.7.1 Rozšířená síť

Uvažme síť s n vstupními uzly, k skrytými uzly a m výstupními jednotkami. Váhu mezi vstupním místem i a skrytou jednotkou j označíme $w_{ij}^{(1)}$, váhu mezi skrytou jednotkou i a výstupní jednotkou j označíme $w_{ij}^{(2)}$. Bias každého uzlu je implementován jako váha dodatečné hrany: vstupní vektory i výstupní vektor skryté vrstvy jsou rozšířeny o komponentu 1, jak ukazuje obr. 2.11. Váha mezi konstantou 1 a skrytým uzlem j je $w_{n+1,j}^{(1)}$, váha mezi konstantou 1 a výstupním uzlem j je $w_{k+1,j}^{(2)}$.



Obrázek 2.11: Dvouvrstvá síť s biasy zapsanými jako váhy hran z konstantního vstupu.

Mezi vstupními místy a skrytými jednotkami je celkem $(n+1) \times k$ vah. Nechť W_1 označuje matici $(n+1) \times k$ s komponentou $w_{ij}^{(1)}$ v i-tém řádku a j-tém sloupci; podobně W_2 matici $(k+1) \times m$ s komponentami $w_{ij}^{(2)}$. Zvýrazněná notace připomíná, že poslední řádek obou matic odpovídá biasům výpočetních uzlů; matice vah bez tohoto řádku bude potřebná ve fázi backpropagation. Rozšířený n-rozměrný vstupní vektor $o = (o_1, \ldots, o_n)$ je transformován na $\bar{o} = (o_1, \ldots, o_n, 1)$. Excitační hodnota net_j j-tého skrytého uzlu je

$$\text{net}_j = \sum_{i=1}^{n+1} w_{ij}^{(1)} \bar{o}_i.$$

Aktivační funkce je sigmoida a výstup $o_j^{(1)}$ tohoto uzlu je tedy

$$o_j^{(1)} = \sigma\left(\sum_{i=1}^{n+1} w_{ij}^{(1)} \bar{o}_i\right).$$

Excitace všech skrytých jednotek lze vypočítat součinem $\bar{\boldsymbol{o}}\boldsymbol{W}_1$. Vektor $\boldsymbol{o}^{(1)}$ jejich výstupů je pak

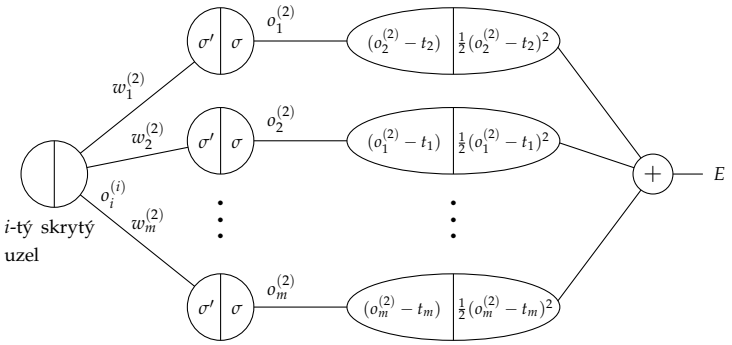
$$\boldsymbol{o}^{(1)} = \sigma(\bar{\boldsymbol{o}}\boldsymbol{W}_1),$$

s konvencí, že sigmoida se aplikuje na každou složku argumentového vektoru. Excitace jednotek výstupní vrstvy se počítá z rozšířeného vektoru $\bar{\boldsymbol{o}}^{(1)} = (o_1^{(1)}, \ldots, o_k^{(1)}, 1)$. Výstup sítě je m-rozměrný vektor

$$\boldsymbol{o}^{(2)} = \sigma(\bar{\boldsymbol{o}}^{(1)} \boldsymbol{W}_2).$$

Tyto vzorce lze zobecnit na libovolný počet vrstev; tok informací v síti se tak počítá jednoduchými maticovými operacemi.

Obr. 2.12 ukazuje rozšířenou síť pro výpočet chybové funkce.



Obrázek 2.12: Rozšířená vícevrstvá síť pro výpočet chybové funkce E.

Nejprve stačí jeden pár vstup–výstup $(\boldsymbol{o}, \boldsymbol{t})$. Síť je rozšířena o vrstvu jednotek, jejichž pravé části počítají kvadratickou odchylku $\frac{1}{2}(o_i^{(2)} - t_i)^2$ pro i-tou složku výstupu a levé části ukládají derivaci $o_i^{(2)} - t_i$. Každá původní výstupní jednotka i počítá sigmoidu σ a produkuje hodnotu $o_i^{(2)}$. Součet odchylek dává chybovou funkci E. Pro p tréninkových příkladů se stejný graf použije na každý pár a dílčí chyby se sečtou.

Po náhodné volbě vah sítě provede potřebné korekce algoritmus backpropagation, který lze rozložit do čtyř kroků:

i) Výpočet feed-forward

ii) Backpropagation do výstupní vrstvy

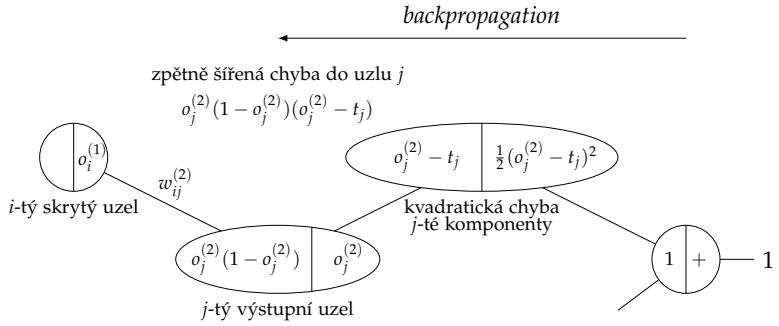
iii) Backpropagation do skryté vrstvy

iv) Aktualizace vah

Algoritmus končí, když hodnota chybové funkce je dostatečně malá.

První krok: výpočet směrem dopředu. Vektor $\boldsymbol{o}$ je předložen síti. Vektory $\boldsymbol{o}^{(1)}$ a $\boldsymbol{o}^{(2)}$ jsou vypočítány a uloženy, u každé jednotky i vyhodnocené derivace aktivačních funkcí.

Druhý krok: Backpropagation do výstupní vrstvy Hledáme první sadu parciálních derivací $\partial E/\partial w_{ij}^{(2)}$. Cesta zpětné propagace od výstupu sítě až k výstupní jednotce j je zobrazena v B-diagramu na obr. 2.13.



Obrázek 2.13: Cesta zpětného šíření chyby k výstupní jednotce j.

Z této cesty jednoduchou inspekcí shromáždíme všechny činitele zpětně propagované chyby $\delta_j^{(2)}$:

$$\delta_j^{(2)} = o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j),$$

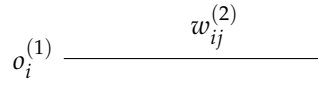
a hledaná parciální derivace je

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = o_j^{(2)}(1-o_j^{(2)})(o_j^{(2)}-t_j)o_i^{(1)} = \delta_j^{(2)}o_i^{(1)}.$$

V tomto posledním kroku považujeme váhu $w_{ij}^{(2)}$ za proměnnou a její vstup $o_i^{(1)}$ za konstantu.

Obrázek 2.14 ukazuje obecnou situaci během algoritmu zpětné propagace: na vstupní straně hrany s váhou w_{ij} je $o_i^{(1)}$, na výstupní straně zpětně propagovaná chyba $\delta_j^{(2)}$.

Třetí krok: Backpropagation do skryté vrstvy. Nyní počítáme parciální derivace $\partial E/\partial w_{ij}^{(1)}$. Každá jednotka j skryté vrstvy je spojena s každou jednotkou q výstupní vrstvy hranou s váhou $w_{jq}^{(2)}$, pro $q = 1, \ldots, m$. Zpětně šířenou chybu až k uzlu j skryté vrstvy je



Obrázek 2.14: Vstup a zpětně šířená chyba na hraně

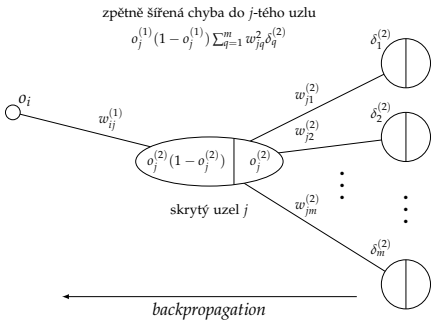
nutné vypočítat přes všechny možné cesty zpět, jak znázorňuje obr. 2.15 . Zpětně šířená chyba je pak

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

Hledaná parciální derivace je tedy

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_i.$$

Stejným způsobem lze zpětně propagovanou chybu vypočítat pro libovolný počet vrstev; výraz pro parciální derivace E si zachovává stejnou analytickou formu.



Obrázek 2.15: Všechny cesty zpětného šíření chyby ke skryté jednotce.

Čtvrtý krok: aktualizace vah. Po výpočtu všech parciálních derivací jsou váhy sítě aktualizovány ve směru záporného gradientu; délku kroku určuje konstanta učení γ. Korekce vah jsou:

$$\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)} \delta_j^{(2)}, \text{ pro } i = 1, \ldots, k+1; j = 1, \ldots, m,$$

a

$$\Delta w_{ij}^{(1)} = -\gamma o_i \delta_j^{(1)}, \text{ pro } i = 1, \ldots, n+1; j = 1, \ldots, k,$$

kde používáme konvenci $o_{n+1} = o_{k+1}^{(1)} = 1$. Korekce vah je nutné provést až po výpočtu zpětně propagované chyby pro všechny jednotky v síti – jinak by korekce neodpovídaly zápornému směru gradientu.

Některé knihy definují zpětně propagovanou chybu se záporným znaménkem; rovnice pro aktualizaci vah pak záporné znaménko nemají (je absorbováno deltami). Jde čistě o konvenci.

2.7.2 Více než jeden tréninkový vzor

Při více vstupně-výstupních vzorech počítá rozšířená síť chybovou funkci pro každý vzor samostatně. Korekce vah se počítají pro každý vzor zvlášť; pro váhu w_{ij} tak získáváme korekce

$$\Delta w_{ij}^{(1)}, \Delta w_{ij}^{(2)}, \ldots, \Delta w_{ij}^{(p)}.$$

Nezbytná aktualizace ve směru gradientu je potom

$$\Delta w_{ij}^{(1)} = \Delta w_{ij}^{(1)} + \Delta w_{ij}^{(2)} + \ldots + \Delta w_{ij}^{(p)}.$$

Takto prováděným korekcím říkáme dávkové nebo offline aktualizace. Často se však váhy aktualizují po každém představení vzoru (online trénink). Korekce pak přesně nesledují směr záporného gradientu, ale při náhodném výběru tréninkových vzorů směr vyhledávání osciluje kolem přesného gradientu a algoritmus v průměru implementuje formu sestupu v chybové funkci. Šum přidaný do směru gradientu navíc může pomoci předejít uváznutí v lokálním minimu. Při tisících tréninkových vzorů je výpočet přesného gradientu nákladný – každá epocha (jedno kolo prezentace všech vzorů) zahrnuje mnoho průchodů dopředu – a online trénink je efektivnější.

2.7.3 Backpropagation v maticové podobě

Přístup označování grafů v důkazu algoritmu zpětné propagace je zcela obecný, neomezuje se na pravidelné vrstevnaté architektury. Tento speciální případ však lze zformulovat vhodně pro vektorové procesory či speciální stroje pro lineárně algebraické operace.

V síti s n vstupy, k skrytými a m výstupními jednotkami platí $\boldsymbol{o}^{(2)} = \sigma(\bar{\boldsymbol{o}}^{(1)}\overline{\boldsymbol{W}}_2)$, kde $\boldsymbol{o}^{(1)} = \sigma(\bar{\boldsymbol{o}}\overline{\boldsymbol{W}}_1)$. Zpětný průchod potřebuje pouze prvních n řádků matice $\overline{\boldsymbol{W}}_1$; tato matice rozměru $n \times k$ se označí $\boldsymbol{W}_1$. Analogicky $\boldsymbol{W}_2$ obsahuje prvních k řádků $\overline{\boldsymbol{W}}_2$. Do konstantních vstupů odpovídajících biasům se gradient nešíří.

Derivace uložené ve fázi feed-forward u k skrytých jednotek a m výstupních jednotek lze zapsat jako dvě diagonální matice

$$\boldsymbol{D}_2 = \begin{pmatrix} o_1^{(2)}(1-o_1^{(2)}) & 0 & \cdots & 0 \\ 0 & o_2^{(2)}(1-o_2^{(2)}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_m^{(2)}(1-o_m^{(2)}) \end{pmatrix},$$

a

$$\boldsymbol{D}_1 = \begin{pmatrix} o_1^{(1)}(1-o_1^{(1)}) & 0 & \ldots & 0 \\ 0 & o_2^{(1)}(1-o_2^{(1)}) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & o_k^{(1)}(1-o_k^{(1)}) \end{pmatrix},$$

Vektor $\boldsymbol{e}$ uložených derivací kvadratických odchylek definujeme jako

$$\boldsymbol{e} = \begin{pmatrix} o_1^{(2)} - t_1 \\ o_2^{(2)} - t_2 \\ \vdots \\ o_m^{(2)} - t_m \end{pmatrix}$$

Pak m-rozměrný vektor $\boldsymbol{\delta}^{(2)}$ zpětně šířené chyby až k výstupním uzlům je

$$\boldsymbol{\delta}^{(2)} = \boldsymbol{D}_2 \cdot \boldsymbol{e}.$$

a k-rozměrný vektor zpětně šířené chyby až ke skryté vrstvě

$$\boldsymbol{\delta}^{(1)} = \boldsymbol{D}_1 \boldsymbol{W}_2 \boldsymbol{\delta}^{(2)}.$$

Korekce matic $\overline{\boldsymbol{W}}_1$ a $\overline{\boldsymbol{W}}_2$ jsou potom

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

Jediné potřebné operace jsou vektor-maticová, maticově-vektorová a vektorová násobení. Rovnice lze snadno zobecnit na ℓ vrstev výpočetních jednotek. Označme $\boldsymbol{W}_{i+1}$ matici spojení mezi vrstvou i a $i+1$ (vrstva 0 je vrstva vstupních míst). Zpětně šířená chyba do výstupní vrstvy je pak $\boldsymbol{\delta}^{(\ell)} = \boldsymbol{D}_\ell \boldsymbol{e}$.

a zpětně šířená chyba do i-té výpočetní vrstvy je definována rekurzivně

$$\boldsymbol{\delta}^{(i)} = \boldsymbol{D}_i \boldsymbol{W}_{i+1} \boldsymbol{\delta}^{(i+1)}, \qquad \text{pro } i = 1, \ldots, \ell - 1.$$

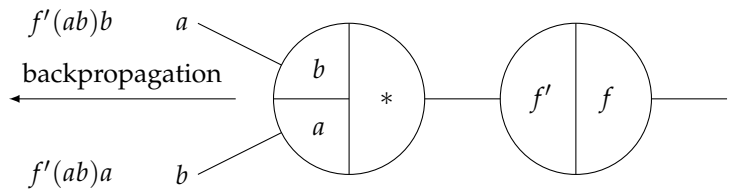
nebo alternativně $\boldsymbol{\delta}^{(i)} = \boldsymbol{D}_i \boldsymbol{W}_{i+1} \ldots \boldsymbol{W}_{\ell-1} \boldsymbol{D}_{\ell-1} \boldsymbol{W}_\ell \boldsymbol{D}_\ell \boldsymbol{e}$.

Korekce matic vah se počítají stejně jako pro dvě vrstvy výpočetních jednotek.

2.7.4 Lokalita backpropagation

Pomocí B-diagramu nyní dokážeme, že sčítání je jedinou integrační funkcí zachovávající při zpětné propagaci lokalitu učení.

V dosud uvedených sítích využívá algoritmus zpětné propagace pouze lokální informace: informace příchozí na vstupní hranu ve fázi feed-forward je uložena u jednotky a odeslána zpět touž hranou ve fázi zpětné propagace. Uvažme příklad, kde integrační funkcí jednotky je násobení a aktivační funkcí f. Obr. 2.16 ukazuje rozdělený pohled na výpočet: dva vstupy a a b přicházejí dvěma vstupními linkami, integrační funkce dá součin ab a ten je předán jako argument funkci f.



Obrázek 2.16: Násobení jako integrační funkce výpočetního uzlu.

Zpětná propagace spočítá parciální derivace $f(ab)$ podle a a b. Hodnota b se však musí vrátit horní hranou a hodnota a spodní hranou. Protože každá z nich přišla jinou cestou, výpočet již není lokální. Integrační funkce zachovávající lokalitu charakterizuje následující tvrzení.

Věta 2.5

V uzlu s n vstupy $x_1, \ldots, x_n$ pouze integrační funkce ve tvaru

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

kde C je konstanta, zaručují lokalitu algoritmu backpropagation v tom smyslu, že na hraně $i \neq j$ nemusí být explicitně uložena žádná informace o x_j.

Důkaz: pro zájemce

Nechť integrační funkcí jednotky je diferencovatelná funkce I s n argumenty. Požadavek lokality říká, že derivace odeslaná po i-té hraně smí záviset pouze na x_i. Pro vhodnou jednorozměrnou funkci f_i tedy

$$\frac{\partial I}{\partial x_i} = f_i(x_i), \text{ pro } i = 1, ..., n.$$

Integrací první rovnice podle x_1 dostaneme

$$I(x_1, \ldots, x_n) = F_1(x_1) + G_1(x_2, \ldots, x_n).$$

Derivace podle x_2 je $f_2(x_2)$, takže část G_1 má tvar $F_2(x_2) + G_2(x_3, \ldots, x_n)$. Opakováním téhož argumentu pro zbývající proměnné plyne

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

kde $F_i' = f_i$ a C je konstanta. Obrácená implikace je přímá: pro funkci tohoto tvaru je $\partial I / \partial x_i = F_i'(x_i)$, a derivace po i-té hraně tedy nepotřebuje hodnotu žádného jiného vstupu.

Kapitola 3

Úlohy neuronových sítí a reprezentace dat

Typ úlohy určuje podobu výstupní vrstvy a s ní i volbu ztrátové funkce: predikce ceny bytu má na výstupu jediné číslo, rozpoznání jednoho z tisíce druhů ptáků tisíc hodnot. Na druhém konci sítě je třeba rozhodnout, jak data vůbec zapsat jako čísla. Tato kapitola prochází obě strany sítě, od vstupu k výstupu.

3.1 Úlohy klasifikace a regrese v DM

Obě úlohy jsou starší než neuronové sítě a řeší se i jinými prostředky. Připomeňme si je proto nejdřív obecně a teprve pak se ptejme, co z toho pro síť plyne.

3.1.1 Klasifikace

Klasifikace je jedna z hlavních úloh strojového učení a často se řeší pomocí neuronových sítí. Spočívá v přiřazení vstupních dat (např. obrázků nebo textů) do jedné z předem definovaných kategorií na základě jejich vlastností.

Například klasifikace obrázků přiřazuje obrázku podle jeho vizuálních rysů třídu, třeba „kočka“ nebo „pes“.

Formální definice klasifikačního problému:

Cílem klasifikace je najít funkci, která mapuje vstupní data na množinu tříd. Máme-li trénovací množinu S s vstupními atributy $A = \{a_1, a_2, ..., a_n\}$ a cílovým atributem y, úkolem je vytvořit kla-

sifikátor minimalizující chybu zobecnění na nových, neznámých datech.

Chyba zobecnění je procento nesprávně klasifikovaných příkladů na datech, která nejsou součástí tréninkové množiny. Pro její výpočet potřebujeme distribuci D, tedy rozdělení dat v reálném světě.

Chybu zobecnění lze definovat pro tréninkovou množinu S a klasifikátor $\mathrm{Cl}(S)$ jako:

$$E(\mathrm{Cl}(S), D) = \sum_{x,y \in U} D(x,y) \cdot L(y, \mathrm{Cl}(S)(x))$$

Kde $L(y, \mathrm{Cl}(S)(x))$ je ztrátová funkce, definovaná např. jako:

$$L(y, \mathrm{Cl}(S)(x)) = \begin{cases} 0, & \text{pokud } y = \mathrm{Cl}(S)(x), \\ 1, & \text{pokud } y \neq \mathrm{Cl}(S)(x). \end{cases}$$

Chyba zobecnění u neuronových sítí

Nechť $\mathrm{NN}(S)$ je neuronová síť natrénovaná na datové množině S. Její chyba zobecnění je pravděpodobnost, že síť nesprávně klasifikuje příklad vybraný dle distribuce D z prostoru instancí; přesnost klasifikace je jedna minus chyba zobecnění. Tréninková chyba je procento příkladů z tréninkové množiny, které síť klasifikovala nesprávně. Formálně:

$$\hat{E}(\mathrm{NN}(S), S) = \sum_{x,y \in S} L(y, \mathrm{NN}(S)(x))$$

kde $L(y, \mathrm{NN}(S)(x))$ je ztrátová funkce nula-jedna, která je definována jako:

$$L(y, \mathrm{NN}(S)(x)) = \begin{cases} 0, & \text{pokud } y = \mathrm{NN}(S)(x) \\ 1, & \text{pokud } y \neq \mathrm{NN}(S)(x) \end{cases}$$

Skutečnou hodnotu chyby zobecnění však známe jen zřídka (snad v syntetických případech), protože distribuce D většinou není k dispozici.

Teoretický odhad chyby zobecnění ponechávám jiným kurzům, jako jsou `KMI/MLDM1` a `KMI/MLDM2`.

Tréninková chyba jako odhad chyby zobecnění bývá optimisticky zkreslená, zejména pokud se síť příliš přizpůsobí tréninkovým datům (overfitting). Přesnější odhad poskytují dva hlavní přístupy: *teoretický* a *empirický*. Budeme se zabývat jen tím empirickým.

Empirický odhad chyby

Základní holdout postup dělí data na tři disjunktní části. Na *trénovací množině* se upravují parametry sítě, na *validační množině* se volí hyperparametry a okamžik zastavení a *testovací množina* přijde na řadu až úplně na konci. Kdo se do testovací množiny dívá průběžně a podle výsledku mění postup, přeučí se i na ni, jen pomaleji než na trénovací data.

Při malém datasetu lze použít k-násobnou křížovou validaci: data rozdělíme na k částí, model vždy trénujeme na $k-1$ z nich a vyhodnotíme na zbývající. Uvádíme průměr i rozptyl metrik mezi foldy. Samostatný testovací soubor, pokud jej máme, do tohoto cyklu nevstupuje.

Stratifikace přibližně zachová zastoupení tříd. Závislé příklady však musí zůstat ve stejné části: snímky jednoho pacienta, věty jednoho dokumentu nebo navazující časové záznamy dělíme jako skupinu. Jinak může model tutéž entitu vidět při trénování i testu a výsledek bude nadhodnocený.

Bootstrap není zvláštním případem křížové validace. Z datasetu o n prvcích vytváříme nový vzorek o n prvcích výběrem s opakováním. Některé původní příklady se proto objeví vícekrát a jiné vůbec; právě nevybrané (*out-of-bag*) příklady lze využít pro odhad chyby. Podrobnější rozdělení dat a regularizační postupy probírá kapitola 5.

Příklad 3.1: Accuracy u nevyvážených tříd

Dataset obsahuje 990 bezvadných a 10 vadných výrobků. Model, který vždy predikuje „bez vady“, má accuracy $990/1000 = 99\,\%$, přestože nenajde jedinou vadu. Samotná accuracy tu o kvalitě modelu nevypovídá. Užitečnější je confusion matrix, precision a recall; pomůže i vážená ztráta nebo posun rozhodovacího prahu.

3.1.2 Regrese

Regrese je další hlavní úlohou v oblasti strojového učení, který se často řeší pomocí neuronových sítí. Na rozdíl od klasifikace, kde se předpovídají diskrétní třídy, je cílem regresního problému predikovat spojité (číslené) hodnoty. Například predikce ceny nemovitosti na základě jejích atributů, jako jsou plocha, poloha a stáří, je typickým příkladem regresního problému.

Formální definice regresního problému:
Cílem regrese je nalézt funkci f, která mapuje vstupní data $X = \{x_1, x_2, ..., x_n\}$ na spojité cílové hodnoty $y \in \mathbb{R}$. Pokud máme tréninkovou množinu S, která obsahuje vstupní atributy X a cílové hodnoty y, úkolem neuronové sítě je vytvořit model f, který mi-

nimalizuje rozdíl mezi predikovanými hodnotami a skutečnými hodnotami v tréninkových datech.

Ztrátová funkce u regresních problémů:
Pro regresní problémy se obvykle používá **kvadratická ztrátová funkce** (Mean Squared Error, MSE), která je definována jako:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

kde N je počet tréninkových vzorků, y_i je skutečná hodnota a $f(x_i)$ je hodnota predikovaná neuronovou sítí. Cílem je minimalizovat tuto ztrátovou funkci, což znamená minimalizovat průměrný čtvercový rozdíl mezi predikovanými a skutečnými hodnotami.

Overfitting a underfitting v regresi:
Podobně jako u klasifikačních úloh může neuronová síť u regresních problémů trpět *overfittingem* nebo *underfittingem*. Overfitting nastává, když se neuronová síť příliš přizpůsobí tréninkovým datům a nedokáže dobře zobecnit na nová, neznámá data. Naopak podfitting znamená, že model není dostatečně složitý na to, aby zachytil vzory v tréninkových datech.

Metody hodnocení výkonu regresních modelů:
Pro hodnocení výkonu regresních modelů se kromě MSE používají i další metriky, jako například:

Mean Absolute Error (MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)|$$

MAE měří průměrnou absolutní chybu mezi predikcemi a skutečnými hodnotami.

R-squared (R^2):
Tato metrika vyjadřuje, jak dobře model vysvětluje variabilitu v datech. Hodnota R^2 se pohybuje mezi 0 a 1, kde 1 znamená, že model dokonale vysvětluje všechna data, a 0 znamená, že model nevysvětluje nic.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - f(x_i))^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

kde $\bar{y}$ je průměrná hodnota cílových hodnot v tréninkové množině.

3.1.3 Další úlohy pro neuronové sítě

Kromě klasifikace a regrese existuje celá řada dalších úloh, kde neuronové sítě nacházejí uplatnění. Mezi tyto úlohy patří:

1. **Segmentace:**
   Segmentace rozděluje vstupní data (často obrázky nebo signály) na různé části nebo regiony. Například v medicíně se používá pro identifikaci částí těla nebo nádorů na MRI snímcích.

2. **Generativní modely:**
   Neuronové sítě mohou generovat nová data podobná těm, na kterých byly trénovány — obrázky, text, hudbu i video. Nejznámější rodiny jsou Generative Adversarial Networks (GAN) a Variational Autoencoders (VAE); mimo multimédia se stejné modely používají třeba k návrhu molekul.

3. **Zpracování přirozeného jazyka (NLP):**
   Strojový překlad (Google Translate), sumarizace, analýza sentimentu, dialogové systémy i generování textu dnes stojí prakticky výhradně na neuronových sítích.

4. **Rozpoznávání obrazu a zvuku:**
   Neuronové sítě se využívají k rozpoznávání a klasifikaci obrazů a zvuků. Typickými příklady jsou rozpoznávání obličejů, hlasů, gest nebo analýza zvukových signálů.

5. **Reinforcement Learning:**
   V reinforcement learningu se neuronové sítě používají pro optimalizaci akcí agentů v prostředí za účelem maximalizace dlouhodobého zisku. Používá se v robotice, autonomních vozidlech nebo pro hraní her.

6. **Rekonstrukce a restaurování dat:**
   Neuronové sítě mohou rekonstruovat poškozená data, například obrázky, zvuky nebo videa. Typické aplikace zahrnují odstraňování šumu, vyplňování chybějících částí nebo zlepšování kvality dat.

Pro každou z těchto úloh se vyvinula jiná architektura — konvoluční sítě pro obraz, rekurentní a později Transformer pro sekvence.

3.2 Aktivační funkce

Funkce perceptronu z kapitoly 1 se skládá z agregační a aktivační funkce. Schodkovou funkci později nahradila sigmoida; tato část porovnává další běžné možnosti.

Aktivační funkce musí být **nelineární**: složením lineárních vrstev vznikne zase jen lineární zobrazení. Nelinearita dává síti větší vyjadřovací sílu. Věta o univerzální aproximaci říká, že síť s jednou nelineární skrytou vrstvou a lineární výstupní vrstvou může na kompaktu aproximovat libovolnou spojitou funkci s libovolnou přesností.

Vět o univerzální aproximaci bylo publikováno několik verzí, z nichž jedna z nejznámějších je věta 3.2.

Ve větě se mluví o nepolynomických funkcích, ale v textu o nelineárních. Je to proto, že věta se týká sítí s jednou skrytou vrstvou (přičemž výstupní vrstva je lineární). Síť s polynomiální aktivací pevného stupně a jednou skrytou vrstvou nemůže s libovolnou přesností aproximovat polynom vyššího stupně. Další vrstvy stupeň zvyšují skládáním polynomů, ale nemění základní omezení dané polynomiální strukturou.

Věta 3.2: Věta o univerzální aproximaci

Nechť $C(X, \mathbb{R}^m)$ označuje množinu spojitých funkcí z podmnožiny X prostoru Euklidovského prostoru $\mathbb{R}^n$ do Euklidovského prostoru $\mathbb{R}^m$. Nechť $\phi \in C(\mathbb{R}, \mathbb{R})$.
Pak ϕ není polynom, právě když pro každé $n \in \mathbb{N}$, $m \in \mathbb{N}$, kompaktní $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m)$, $\varepsilon > 0$ existují $k \in \mathbb{N}$, $\boldsymbol{A} \in \mathbb{R}^{k \times n}$, $\boldsymbol{b} \in \mathbb{R}^k$, $\boldsymbol{C} \in \mathbb{R}^{m \times k}$ tak, že

$$\sup_{\boldsymbol{x} \in K} \|f(\boldsymbol{x}) - g(\boldsymbol{x})\| < \varepsilon$$

kde $g(\boldsymbol{x}) = \boldsymbol{C} \cdot (\phi \circ (\boldsymbol{A} \cdot \boldsymbol{x} + \boldsymbol{b}))$.

Historie věty a její důkazy

Formulace uvedená ve větě 3.2 je výsledkem postupného zobecňování. Historicky první důkaz [Cybenko1989] podal v roce 1989 George Cybenko, a to pro sítě s jednou skrytou vrstvou a sigmoidní aktivační funkcí: ukázal, že takové sítě dokáží aproximovat libovolnou spojitou funkci na kompaktní množině s libovolnou přesností. Kurt Hornik [Hornik1991] v roce 1991 tento výsledek zobecnil a ukázal, že univerzální aproximační schopnost není důsledkem konkrétního tvaru sigmoidy, nýbrž samotné vícevrstvé dopředné architektury — stačí, aby aktivační funkce byla spojitá, omezená a nekonstantní. Leshno a kol. [Leshno1993] dokázali, že nutnou a postačující podmínkou univerzální aproximace je nepolynomiálnost aktivační funkce. Tuto formulaci uvádí věta 3.2.

Důkaz se opírá o netriviální výsledky funkcionální analýzy; Cybenkova verze využívá například Hahnovu–Banachovu a Rieszovu větu. Věta tedy zaručuje, že hledaná síť *existuje*. Neříká však, kolik bude mít jednotek (může jich být astronomicky mnoho) ani jak k jejím vahám dojít učením z dat. Obojí je v praxi obtížnější než samotná otázka existence.

[Cybenko1989]
G. Cybenko:
Approximation by Superpositions of a Sigmoidal Function.
Mathematics of Control, Signals and Systems 2(4), 303–314 (1989).

[Hornik1991]
K. Hornik:
Approximation Capabilities of Multilayer Feedforward Networks.
Neural Networks 4(2), 251–257 (1991).

[Leshno1993]
M. Leshno, V. Ya. Lin, A. Pinkus, S. Schocken:
Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function.
Neural Networks 6(6), 861–867 (1993).

3.3 Vstupy

Kódování je přímočaré tehdy, když data již tvoří vektory číselných atributů.

Booleovské atributy se obvykle kódují takto:

$$\begin{aligned} \text{pravda} &\mapsto 1 \\ \text{nepravda} &\mapsto 0 \end{aligned}$$

Toto už známe z předchozích lekcí

alternativně

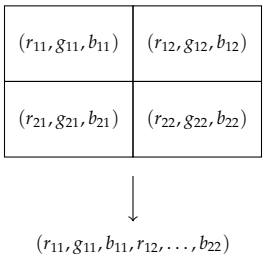
$$\text{nepravda} \mapsto -1.$$

Numerické atributy obvykle škálujeme do zvoleného intervalu nebo standardizujeme na nulový průměr a jednotkovou směrodatnou odchylku. Bez této úpravy může atribut s velkým číselným rozsahem nepřiměřeně ovlivnit gradienty.

Pokud se hodnoty numerických atributů mezi jednotlivými příklady značně liší (například řádově), může být užitečné zobrazit je na logaritmickou škálu. Tím se redukuje vliv extrémně velkých hodnot a distribuce se stane lépe zvládnutelnou pro modely, které očekávají zhruba homogenní vstupy.

Má-li veličina přirozené intervaly, lze ji také kvantizovat do diskrétních kategorií. Ztrácíme tím přesnou hodnotu i uspořádání uvnitř intervalu.

Obr. 3.1 ukazuje, jak zploštění převede malý RGB obraz na posloupnost čísel.



Obrázek 3.1: Obraz o velikosti 2×2 pixely zapsaný jako vektor dvanácti číselných atributů.

Obrázek o velikosti $X \times Y$ pixelů zapisujeme jako tenzor $X \times Y \times 3$. Zploštěním na vektor délky $3XY$ bychom sice hodnoty zachovali, ale architektura by už neměla přímo k dispozici informaci o sousedství pixelů a společné příslušnosti barevných kanálů k jednomu pixelu.

Konvoluční sítě v kapitole 7 tuto prostorovou strukturu zachovávají pomocí lokálních spojení a sdílených vah.

Kategorická data kódujeme principem *one-hot*. Každé kategorii přiřadíme binární vektor s jednou jedničkou a ostatními nulami. Například atribut *barva* s hodnotami „červená“, „zelená“ a „modrá“ zakódujeme trojicí vstupů:

$$\begin{aligned} \text{červená} &\mapsto (1,0,0), \\ \text{zelená} &\mapsto (0,1,0), \\ \text{modrá} &\mapsto (0,0,1). \end{aligned}$$

Všimněme si, že bychom kategorie *neměli* kódovat jedním číselným vstupem (např. červená $\mapsto 1$, zelená $\mapsto 2$, modrá $\mapsto 3$): tím bychom do dat vnesli umělé uspořádání a vzdálenosti mezi kategoriemi, které v původních datech nejsou.

Pro malý počet kategorií je one-hot kód přehledný. Při tisících hodnot však vznikají dlouhé řídké vektory; tehdy se častěji učí kratší hustý *embedding*.

Příklad 3.3: Tři atributy, tři různá kódování

Atribut „barva“ s hodnotami červená, zelená a modrá zakódujeme one-hot vektorem, nikoli čísly $1, 2, 3$, která by vnucovala pořadí. Věk můžeme standardizovat jako $(x - \mu)/\sigma$. Identifikátor zákazníka naopak nesmíme bezmyšlenkovitě použít jako číselný atribut: síť by se mohla naučit konkrétní osoby a test po náhodném dělení by nadhodnotil generalizaci.

3.4 Výstupní vrstvy a ztrátové funkce

Na výstupu sítě je problém zakódování surových datových hodnot do skutečných hodnot $\boldsymbol{y}$ pro výstupní uzly velmi podobný problému kódování vstupů.

To by se týkalo cílových hodnot y. Ale u predikce $\hat{y}$ to nemůžeme zaručit. Ideálně by přesně odpovídala požadované hodnotě y, ztráta by byla nulová a měli bychom hotovo. Před začátkem učení se taková shoda objeví jen výjimečně.

Předchozí odvození gradientů používalo kvadratickou chybu.

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

To udržuje algebraické výpočty jednoduché, ale není to jediná možnost. Ve skutečnosti je pro většinu aplikací hlubokého učení běžnější

interpretovat výstupní hodnoty $\hat{y}$ jako pravděpodobnosti a jako ztrátovou funkci použít negativní logaritmickou pravděpodobnost (negative log likelihood, NLL).

Záporná logaritmická pravděpodobnost

Učení najde vektor vah $\boldsymbol{w}$, který maximalizuje pravděpodobnost pozorovaných dat. Protože je logaritmická funkce monotónní, je to ekvivalentní maximalizaci logaritmické pravděpodobnosti dat, což je zase ekvivalentní minimalizaci chybové funkce definované jako záporná logaritmická pravděpodobnost.

Pro jediný příklad chceme zvýšit pravděpodobnost správné třídy. Ztrátou pro j-tý příklad proto zvolíme záporný logaritmus pravděpodobnosti, kterou jí model přiřadil:

$$\mathcal{L} = -\log P_{\boldsymbol{w}}(\boldsymbol{t}_j \mid \boldsymbol{x}_j).$$

Čím blíže je tato pravděpodobnost jedničce, tím blíže je ztráta nule; naopak pravděpodobnost blízká nule vede k obrovské ztrátě. Pro celou trénovací množinu pak hledáme $\boldsymbol{w}^*$, které minimalizuje součet negativních logaritmů pravděpodobností N příkladů:

$$-\sum_{j=1}^{N} \log P_{\boldsymbol{w}}(\boldsymbol{t}_j \mid \boldsymbol{x}_j).$$

Příklad 3.4: Záporná logaritmická pravděpodobnost

Uvažujme model, který klasifikuje texty podle sentimentu do tří tříd: „negative“, „neutral“ a „positive“. Mějme tři příklady, jejichž skutečná třída je vždy „positive“, a model jim přiřadil tato rozdělení:

	$p($„negative“$)$	$p($„neutral“$)$	$p($„positive“$)$
příklad 1	0.1	0.2	0.7
příklad 2	0.05	0.15	0.8
příklad 3	0.2	0.3	0.5

Ztrátu spočítáme jako součet záporných logaritmů pravděpodobností přiřazených správné třídě „positive“:

$$\mathcal{L} = -\big[\log(0.7) + \log(0.8) + \log(0.5)\big] \approx 1.27.$$

Tuto hodnotu při trénování modelu minimalizujeme.

Křížová entropie

Další možností je minimalizace křížové entropie *cross-entropy*. Křížová entropie značená $H(P, Q)$, je druhem míry odlišnosti mezi

dvěma rozděleními pravděpodobnosti P a Q:

$$\begin{aligned} H(P,Q) &= -\mathbb{E}_{\boldsymbol{z}\sim P(\boldsymbol{z})}\left[\log Q(\boldsymbol{z})\right] \\ &= -\int P(\boldsymbol{z}) \log Q(\boldsymbol{z})\, d\boldsymbol{z} \quad \text{pro spojité } \boldsymbol{z} \\ &= -\sum_{\boldsymbol{z}} P(\boldsymbol{z}) \log Q(\boldsymbol{z}) \quad \text{pro diskrétní } \boldsymbol{z} \end{aligned}$$

Poznámka: Cross-entropy není vzdálenost, protože $H(P,P) = H(P) \neq 0$.

Dá se ukázat, že:

$$H(P,Q) = H(P) + D_{\mathrm{KL}}(P \parallel Q),$$

kde $H(P)$ je entropie

$$\begin{aligned} H(P) &= -\int P(\boldsymbol{z}) \log P(\boldsymbol{z})\, d\boldsymbol{z} \quad \text{pro spojité } \boldsymbol{z}, \\ &= -\sum_{\boldsymbol{z}} P(\boldsymbol{z}) \log P(\boldsymbol{z}) \quad \text{pro diskrétní } \boldsymbol{z}, \end{aligned}$$

a $D_{\mathrm{KL}}(P \parallel Q)$ je Kullback-Leiblerova divergence

$$\begin{aligned} D_{\mathrm{KL}}(P \parallel Q) &= \int P(\boldsymbol{z}) \log \frac{P(\boldsymbol{z})}{Q(\boldsymbol{z})}\, d\boldsymbol{z} \quad \text{pro spojité } \boldsymbol{z}, \\ &= \sum_{\boldsymbol{z}} P(\boldsymbol{z}) \log \frac{P(\boldsymbol{z})}{Q(\boldsymbol{z})} \quad \text{pro diskrétní } \boldsymbol{z}. \end{aligned}$$

Studenti toto znají z kurzu Teorie informace KMI/TINF.

Takže pro pevné P je minimalizace Q vzhledem k $H(P,Q)$ také minimalizací $D_{\mathrm{KL}}(P \parallel Q)$.

V machine learningu se používá tato definice, kde P je skutečná distribuce $P(\boldsymbol{x},\boldsymbol{t})$ nad trénovací množinou a Q je prediktivní hypotéza $P_{\boldsymbol{w}}(\boldsymbol{t} \mid \boldsymbol{x})$.

Minimalizace $H(P^*(\boldsymbol{x},\boldsymbol{t}), P_{\boldsymbol{w}}(\boldsymbol{t} \mid \boldsymbol{x}))$ úpravou vah $\boldsymbol{w}$ přiblíží model skutečné distribuci.

Ve skutečnosti nemůžeme tuto funkci přímo minimalizovat, protože neznáme $P^*(\boldsymbol{x},\boldsymbol{t})$, ale máme k dispozici vzorky z $P^*(\boldsymbol{x},\boldsymbol{t})$. Proto součet přes skutečná data v trénovací množině aproximuje očekávání na množině vzorků.

Příklad 3.5: Křížová entropie

Mějme model, který predikuje rozdělení pravděpodobností pro tři třídy „negative“, „neutral“ a „positive“. Skutečné rozdělení je

$$P(\text{„negative“}) = 0.0, \quad P(\text{„neutral“}) = 0.0,$$
$$P(\text{„positive“}) = 1.0.$$

model však predikuje

$$Q(\text{„negative“}) = 0.1, \quad Q(\text{„neutral“}) = 0.2,$$
$$Q(\text{„positive“}) = 0.7.$$

Křížová entropie těchto dvou rozdělení je

$$\begin{aligned} H(P,Q) &= -\left[1.0 \cdot \log(0.7) + 0.0 \cdot \log(0.2) + 0.0 \cdot \log(0.1)\right] \\ &= -\log(0.7). \end{aligned}$$

Je-li cílové rozdělení P typu one-hot, zbude v křížové entropii jediný člen: záporný logaritmus pravděpodobnosti správné třídy. Křížová entropie je tedy zobecněním záporné logaritmické věrohodnosti na obecné cílové rozdělení.

Multiclass

Pro pravděpodobnostní výstup s K možnými třídami je potřeba K výstupních jednotek s nezápornými hodnotami o součtu 1.

Abychom toho dosáhli, používáme softmax vrstvu, která z vektoru K logitů $\boldsymbol{z} = (z_1, \ldots, z_K)$ vytvoří pravděpodobnostní vektor $\boldsymbol{p} = (p_1, \ldots, p_K)$:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

Výsledné hodnoty jsou kladné a jejich součet je 1, takže p_i můžeme číst jako modelovou pravděpodobnost třídy i.

Softmax je navíc hladký, takže se přes něj dá bez potíží derivovat. Ve dvojici s kategorickou křížovou entropií tvoří standardní zakončení sítě pro vícetřídní klasifikaci; při odvození gradientu se navíc obě funkce podstatně zjednoduší.

Softmax jako zobecnění sigmoidy

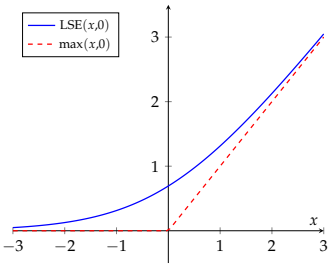
U binární klasifikace lze $\sigma(x)$ interpretovat jako pravděpodobnost první třídy a $1 - \sigma(x)$ jako pravděpodobnost druhé. Softmax tuto konstrukci zobecňuje na více tříd. Pro $K = 2$ se po vydělení čitatele i jmenovatele faktorem e^{z_1} první složka upraví na

$$p_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2}} = \frac{1}{1 + e^{z_2 - z_1}} = \sigma(z_1 - z_2),$$

a symetricky $p_2 = \sigma(z_2 - z_1) = 1 - p_1$. Pro dvě třídy se tedy softmax přesně redukuje na sigmoidu aplikovanou na rozdíl obou logitů. Intuitivně je softmax „vícerozměrná sigmoida“, která rozděluje jistotu modelu mezi více tříd.

Softmax jako vyhlazená funkce max

Rozdíl mezi maximem a jeho hladkou aproximací ukazuje obr. 3.2.



Obrázek 3.2: Log-sum-exp jako hladká náhrada maxima pro dva vstupy.

Odkud se vzalo jméno *softmax*? Při samotném rozhodování by nás často zajímal jen nejvyšší logit, tedy hodnota $\max_i x_i$ (respektive index třídy, která jej dosahuje). Funkce max je však pro trénování nepoužitelná: není hladká a její gradient je téměř všude nulový vzhledem ke všem složkám kromě té vítězné, takže by zpětné šíření nemělo co propagovat.

Hladkou náhradou maxima je funkce log-sum-exp:

$$\mathrm{LSE}(\boldsymbol{x}) = \log \sum_{j=1}^{d} e^{x_j}.$$

Platí totiž odhad

$$\max_j x_j \;\leq\; \mathrm{LSE}(\boldsymbol{x}) \;\leq\; \max_j x_j + \log d,$$

takže log-sum-exp se od skutečného maxima liší nejvýše o konstantu $\log d$ a aproximuje jej tím lépe, čím jsou hodnoty x_j od sebe dále. Na rozdíl od maxima je ale spojitá a diferencovatelná, a tedy vhodná pro gradientní metody.

Se softmaxem souvisí log-sum-exp derivováním: parciální derivace log-sum-exp podle x_i je právě i-tá složka softmaxu:

$$\frac{\partial}{\partial x_i}\mathrm{LSE}(\boldsymbol{x}) = \frac{e^{x_i}}{\sum_{j=1}^{d} e^{x_j}} = \mathrm{softmax}(\boldsymbol{x})_i.$$

Softmax tedy říká, jak silně jednotlivé složky x_i přispívají k hladkému „měkkému maximu“ $\mathrm{LSE}(\boldsymbol{x})$. V tomto smyslu je softmax vyhlazenou verzí funkce max — přesněji řečeno vyhlazenou verzí funkce *argmax*, protože jeho výstupem není největší hodnota, nýbrž (měkký) ukazatel na to, *která* složka je největší.

3.5 Skryté vrstvy

Skryté vrstvy v neuronových sítích obvykle používají méně různorodých aktivačních funkcí ve srovnání s výstupními vrstvami.

1985–2010: sigmoida a tanh

2010–nyní: ReLU, softplus

Částečně proto, že se věří, že pomáhají vyřešit *problém mizejících gradientů*:

Je-li derivace malá nebo nulová, změna vstupních vah téměř neovlivní výstup neuronu. Při zpětném průchodu se malé derivace násobí, takže gradient ve vzdálenějších vrstvách může téměř zmizet a jejich váhy se přestanou učit.

Gradient vzdálené vrstvy obsahuje součin mnoha derivací. Derivace sigmoidy a tanh mají absolutní hodnotu menší než 1, takže součin může s hloubkou exponenciálně klesat. Vrstvy blízko vstupu pak dostávají zanedbatelný učicí signál.

Problém explodujících gradientů

Při *explozi gradientů* naopak součin rychle roste. Častá je u rekurentních sítí nad dlouhou sekvencí, protože se v součinu opakuje tatáž matice vah.

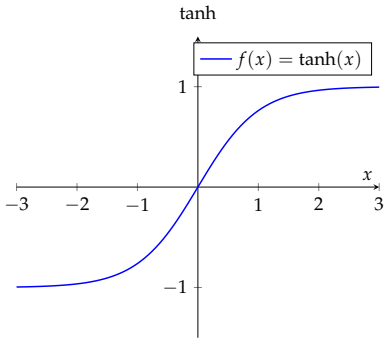
Velký gradient vyvolá velkou změnu parametrů a trénování může oscilovat nebo divergovat. *Ořezávání gradientu* jej při překročení zvolené normy přeškáluje na mezní velikost.

Oba jevy vznikají násobením mnoha členů: činitele menší než 1 součin tlumí, větší než 1 jej zvětšují. Explozi lze omezit ořezáváním gradientů; na mizení mají vliv také aktivační funkce.

Hyperbolický tangens

Hyperbolický tangens (tanh) je aktivační funkce běžně používaná v neuronových sítích, zejména v dřívějších typech rekurentních neuronových sítí. Její matematická definice je:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Funkce $\tanh(x)$ převádí vstup na výstup v rozmezí $[-1, 1]$; sigmoida oproti tomu vrací hodnoty mezi 0 a 1. Rozdíl je v tom, že tanh je souměrný kolem nuly, takže záporný vstup dá záporný výstup. Aktivace jedné vrstvy pak nemají systematicky kladný průměr, což usnadňuje práci vrstvě následující.

Rectified linear unit

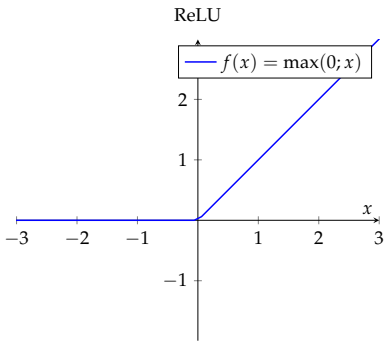
[Nair2010] V. Nair, G. E. Hinton, Rectified linear units improve restricted Boltzmann machines. ICML 2010

Zhruba od roku 2010 se používá jako aktivační funkce zejména Rectified linear unit (ReLU) [Nair2010]:

$$\begin{aligned}\phi(x) &= \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \\ &= \max(0, x).\end{aligned}$$

Funkce ReLU zjevně není hladká kvůli onomu zlomu v bodě 0. To se řeší tak, že stanovíme, že derivace v tom bodě je 0 a tedy máme:

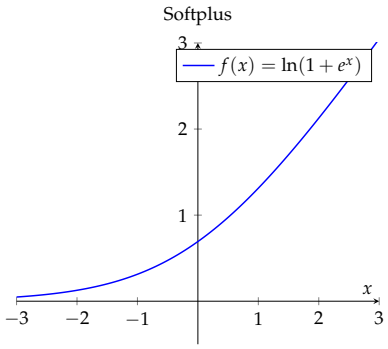
$$\phi'(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$



Softplus

Softplus [Dugas2001], $\text{softplus}(x) = \ln(1 + e^x)$, je hladká aproximace ReLU. Její derivace existuje všude a není přesně nulová. Výpočet je dražší než u ReLU; funkce se používá také k parametrizaci kladné veličiny, například rozptylu.

[Dugas2001] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, R. Garcia.
Incorporating second-order functional knowledge for better option pricing.
In Advances in neural information processing systems, pp. 472-478, (2001)



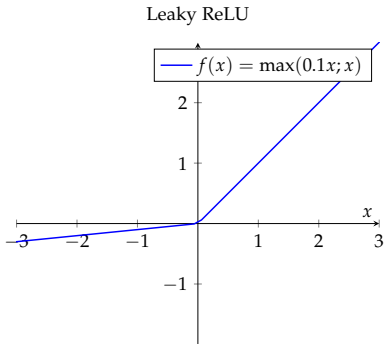
Leaky ReLU

Leaky ReLU ponechává malý sklon i pro záporné vstupy:

$$\text{LeakyReLU}(x) = \max(\alpha x, x),$$

kde bývá například $\alpha = 0{,}01$. Záporná větev má derivaci α, takže jednotka neztratí gradient jen proto, že jsou její preaktivace záporné.

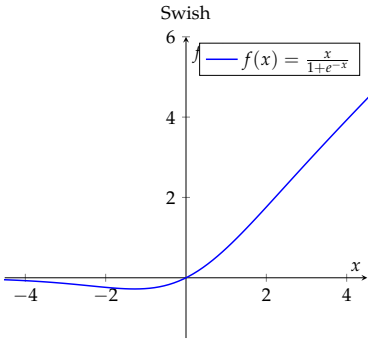
[Maas2013] A.L. Maas, A.Y. Hannun, A.Y. Ng
Rectifier Nonlinearities Improve Neural Network Acoustic Models
In Proceedings of the 30th International Conference on Machine Learning (ICML), Atlanta, Georgia, USA (2013)



Swish

Swish [Ramachandran2017] má tvar $\text{swish}(x) = x\sigma(x)$. Pro velká kladná x se blíží identitě, kolem nuly je hladká a mírně záporné vstupy nepotlačí na nulu. Funkce vzešla z automatizovaného hledání; příbuzná GELU se používá v Transformerech.

[Ramachandran2017]
P. Ramachandran, B. Zoph, Q.V. Le
Searching for Activation Functions.
arXiv:1710.05941 (2017)



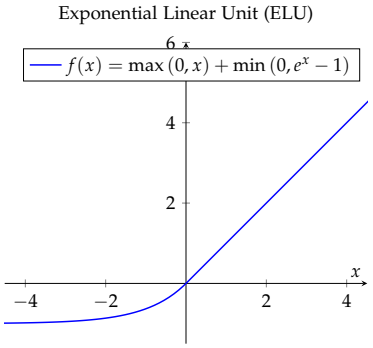
[Clevert2016] Djork-Arne Clevert, Thomas Unterthiner & Sepp Hochreiter
Fast and accurate deep network learning by exponential linear units (ELUs)
ICLR 2016

Exponential Linear Unit

Aktivační funkce Exponential Linear Unit (ELU) [Clevert2016] je nelineární funkce navržená tak, aby překonala některé problémy klasické ReLU, zejména problém nulových gradientů pro záporné hodnoty. ELU je definována jako:

$$f(x) = \begin{cases} x & \text{pro } x > 0 \\ \alpha(e^x - 1) & \text{pro } x \leq 0 \end{cases}$$

kde α je hyperparametr, obvykle rovný 1. Pro záporné vstupy tedy ELU neklesne pod $-\alpha$, ale ani nezůstane na nule: jednotka si zachová nenulový gradient a nemůže trvale odumřít. Záporná větev navíc stahuje průměrnou aktivaci vrstvy blíž k nule, což má podobný efekt jako normalizace.



Počet a velikost skrytých vrstev

Hloubka dovoluje opakovaně skládat jednodušší transformace. U obrazu mohou časné vrstvy reagovat na hrany a další vrstvy jejich výstupy kombinovat do složitějších tvarů. Není však obecně pravda, že při stejném počtu parametrů je hlubší síť vždy lepší; záleží na úloze, architektuře a optimalizaci.

Hloubka a šířka nejsou jediné volby a u většiny z nich neumíme spolehlivě vysvětlit, proč fungují.

Teorie, která by pro daný problém dopředu určila vhodnou architekturu, neexistuje. K dispozici je jen nasbíraná zkušenost, co se na jakých úlohách osvědčilo.

Návrh sítě je proto z velké části empirický. Vyplatí se umět rychle vyzkoušet několik variant a poctivě je porovnat, spíše než dlouho hledat tu jedinou správnou.