

Kapitola o

Úvod do hlubokého učení

0.1 Umělá inteligence, strojové učení a hluboké učení

Umělá inteligence je široký obor zabývající se konstrukcí systémů, které vnímají, usuzují, učí se nebo jednají. **Strojové učení** je jeho část, v níž se chování systému neurčuje pouze ručně zapsanými pravidly, ale odvozuje se z dat. **Hluboké učení** je potom část strojového učení založená na neuronových sítích s více vrstvami naučených reprezentací.

Rozdíl mezi nimi je i praktický. Klasické programování začíná pravidly navrženými člověkem, zatímco neuronová síť dostává příklady a ztrátovou funkci. Trénovací algoritmus hledá parametry, s nimiž síť na příkladech dosahuje malého chybového signálu a současně dokáže zobecnit na dosud neviděná data.

0.2 Učení reprezentací

Vstupní data obvykle nejsou ve tvaru, v němž se úloha řeší snadno. Obraz je pole intenzit, věta posloupnost tokenů a zvuk časová řada. Model proto nejprve převádí vstup x na vnitřní reprezentaci

$$h = h_{\phi}(x)$$

a teprve z ní počítá predikci

$$\hat{y} = g_{\theta}(h), \quad f_{\theta, \phi} = g_{\theta} \circ h_{\phi}.$$

Parametry reprezentace ϕ i rozhodovací části θ lze učit společně minimalizací stejné ztrátové funkce. Tento princip se označuje jako **učení reprezentací** (*representation learning*).

Klasický systém často odděluje ručně navržené příznaky od následného klasifikátoru nebo regresoru. Takový postup může dobře využít oborové znalosti, ale jeho kvalita je omezena tím, co konstruktér dokázal předem vyjádřit. Hluboká síť se naproti tomu učí transformace přímo z dat. Jedna vlastnost nemusí odpovídat jedinému neuronu: informace bývá rozložena mezi více souřadnic a jedna souřadnice se současně podílí na několika vlastnostech. Hovoří se proto o **distribuované reprezentaci**.

Užitečná reprezentace zachovává informaci potřebnou pro cílovou úlohu a potlačuje změny, které pro ni podstatné nejsou. Klasifikátor objektů má být například málo citlivý na drobnou změnu osvětlení nebo posun objektu. Jindy se má reprezentace měnit předvídatelně: posun vstupního obrazu má posunout také mapu detekovaných objektů. První vlastnost se nazývá **invariance**, druhá **ekvivariance**. Architektura a data určují, které z nich se model může naučit. Konvoluce podporuje ekvivarianci vůči posunu, augmentace ukazuje žádoucí invariance a mechanismus attention umožňuje přímo modelovat vztahy mezi prvky vstupu.

Více vrstev dovoluje skládat jednodušší transformace do hierarchie. U obrazu mohou časné vrstvy reagovat na lokální kontrasty a textury, zatímco pozdější reprezentace nesou informaci o částech objektů a jejich vztazích. U jazyka se počáteční embedding tokenu mění v reprezentaci závislou na okolním kontextu. Tato interpretace je užitečná, není však doslovným popisem každé sítě: jednotlivé vrstvy nemají předem přidělený význam a naučená informace může být rozložena napříč celým modelem.

Společné učení celé kompozice se označuje jako **učení end-to-end**. Jeho výhodou je, že se reprezentace přizpůsobí přímo zadanému cíli. Stejná vlastnost je také zdrojem rizika. Ztrátová funkce neurčuje, *jakým* signálem má model úlohu řešit. Pokud trénovací data obsahují snadnou zkratku — například pozadí silně korelované s třídou — může ji síť zakódovat místo zamýšleného pojmu. Malá trénovací chyba proto sama nedokazuje, že reprezentace zachycuje správnou strukturu problému.

Kvalita reprezentace se posuzuje podle chování na nových datech, přenosu mezi úlohami, odolnosti vůči relevantním změnám a podle chyb v jednotlivých podskupinách. Reprezentace naučená na rozsáhlém zdrojovém souboru může sloužit jako výchozí bod pro menší cílovou úlohu; tomuto postupu se věnuje kapitola 8. Self-supervised cíle v kapitole 16 ukazují, že užitečnou reprezentaci lze učit také bez ručně zadaného labelu pro každý příklad.

Skládání diferencovatelných transformací je společným základem dopředných, konvolučních i rekurentních sítí a Transformerů. Parametry se učí gradientními metodami a algoritmem backpropagation; rozdíly mezi architekturami spočívají především v tom, jaké vazby a induktivní preference do reprezentace vnášejí.

0.3 Milníky oboru

Pokrok v hlubokém učení obvykle vznikl souběhem čtyř složek: vhodného modelu, učicího algoritmu, dostupných dat a výpočetního prostředí. Samotný architektonický nápad stačil jen zřídka. Výsledek navíc závisel na evaluační úloze; metoda úspěšná v jedné doméně nemusela být praktická v jiné.

McCulloch a Pitts formulovali v roce 1943 jednoduchý matematický model neuronu [McCULLOCH1943]. Rosenblattův perceptron spojil podobný výpočetní prvek s algoritmem učení z označených příkladů [ROSENBLATT1958]. Perceptron však vytváří pouze lineární rozhodovací hranici. Kritika jeho omezení [MINSKY1969] přispěla k útlumu zájmu o neuronové sítě, současně ale zpřesnila otázku: jak účinně učit vícevrstvé modely s nelineárními skrytými reprezentacemi?

Řetězové pravidlo a zpětné šíření derivací byly známy v různých podobách již dříve, jejich použití pro praktické učení vícevrstevných sítí však výrazně zpopularizovala práce Rumelharta, Hintona a Williamse [RUMELHART1986]. [Backpropagation](#) je efektivní postup pro výpočet gradientu složené funkce, nikoli architektura ani optimalizační metoda; vlastní změnu parametrů provádí například gradientní sestup.

Konvoluční sítě využily lokální vazby a sdílení parametrů pro obrazová data. LeNet ukázal jejich praktické použití při rozpoznávání znaků [LECUN1998]. AlexNet pak v roce 2012 spojil hlubší konvoluční architekturu, rozsáhlý dataset ImageNet a výpočty na GPU [KRIZHEVSKY2012]. Jeho úspěch neznamenal vznik hlubokého učení v tomto roce; přesvědčivě však ukázal, že škálování dat a výpočtu může změnit dosažitelnou kvalitu modelu.

Sekvenční modely řešily odlišný problém: jak uchovat informaci napříč časem. Jednoduché rekurentní sítě trpí při učení dlouhých závislostí, což zmírnila buňka LSTM [HOCHREITER1997]. Modely encoder–decoder umožnily převádět jednu posloupnost na druhou [SUTSKEVER2014] a mechanismus attention odstranil nutnost stlačit celý vstup do jediného pevného vektoru [BAHDANAU2015].

Transformer nahradil sekvenční rekurenci bloky attention; původní architekturu popsali Vaswani a kol. [VASWANI2017]. Během trénování zpracovává pozice paralelně a každá pozice má přímou výpočetní

Vybrané milníky

- 1943 formální neuron
- 1958 perceptron
- 1986 backpropagation
- 1997–1998 LSTM, LeNet-5
- 2012 AlexNet
- 2014–2015 GAN, seq2seq, attention
- 2017 Transformer
- 2020 difuzní modely

Orientační výběr; milníky se překrývají a starší metody nezanikají.

vazbu ke všem ostatním. Tato architektura se stala základem kontextových jazykových reprezentací, velkých jazykových modelů i řady obrazových a multimodálních systémů. Její význam nelze oddělit od škálování předtrénovacích dat, výpočetního výkonu a následné adaptace modelů.

Souběžně se rozvíjely generativní modely. Autoencodery a variační autoencodery formulovaly učení latentního prostoru, GAN zavedly soupeření generátoru a diskriminátoru [GOODFELLOW2014] a difuzní modely postavily generování na postupném odšumování [HO2020]. Self-supervised učení ukázalo, že dohled lze vytvářet ze struktury samotných dat, například predikcí maskovaných částí nebo kontrastivním porovnáváním různých pohledů na tentýž příklad.

Milníky proto není vhodné číst jako seznam metod, které své předchůdce zcela nahradily. Konvoluce zůstávají účinnou induktivní preferencí, rekurentní modely mohou být vhodné pro proudová data a jednodušší model bývá při omezených datech lepší volbou než rozsáhlá architektura. Výkon modelu vzniká ze souhry reprezentace, cíle učení, dat, optimalizace a evaluace.

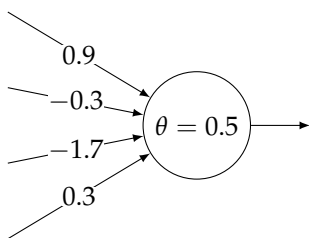
Kapitola 1

Perceptron

Perceptron je lineární binární klasifikátor a současně historický výchozí bod neuronových sítí. Jeho jednoduchost umožňuje přesně popsat rozhodovací hranici, algoritmus učení i podmínky konvergence.

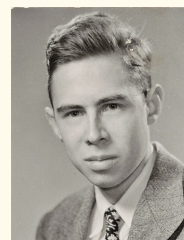
Perceptron je základní model neuronové sítě sloužící k binární klasifikaci. Má jeden nebo více vstupů, z nichž každý má přiřazeno reálné číslo – **váhu**. Dále má perceptron **práh aktivace** θ . Perceptron s n vstupy je formálně určen dvojicí tvořenou vektorem vah $w = \langle w_1, \dots, w_n \rangle$ a prahem excitace θ .

Perceptron bývá obvykle zakreslován jako uzel s n vstupními hranami s jejich váhami, prahem excitace uvnitř a výstupní hranou (obr. 1.1).



Obrázek 1.1: Grafická reprezentace perceptronu se čtyřmi vstupy, vahami a prahem excitace.

Perceptron s n vstupy představuje implicitně zadanou funkci $f : \mathbb{R}^n \rightarrow \{0, 1\}$, která vrací 1, pokud vážený součet vstupních hodnot



Frank Rosenblatt (1928–1971)
Americký psycholog a neurovědec formuloval perceptron a v Cornell Aeronautical Laboratory vedl konstrukci jeho rané hardwarové realizace. Jeho práce spojila biologickou inspiraci s učením lineárního klasifikátoru.

dosáhne prahu excitace θ , a 0 v opačném případě, tj.

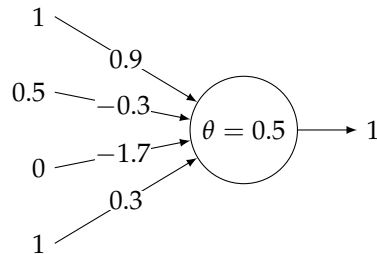
$$f(x) = \begin{cases} 1 & \text{pokud } w^T \cdot x \geq \theta, \\ 0 & \text{jinak.} \end{cases} \quad (1.1)$$

Příklad 1.1

Pro neuron na obr. 1.2 a vstupní vektor $x = \langle 1, 0.5, 0, 1 \rangle$, vychází

$$w \cdot x = 0.9 \cdot 1 + (-0.3) \cdot 0.5 + (-1.7) \cdot 0 + 0.3 \cdot 1 = 1.05.$$

To je více než θ a tedy $f(x) = 1$.



Obrázek 1.2: Výpočet výstupu perceptronu z příkladu 1.1.

Funkce perceptronu je vlastně složení dvou funkcí:

- **Agregační funkce** $\mathbb{R}^n \rightarrow \mathbb{R}$ – v našem případě vážený součet vstupních hodnot reprezentovaný skalárním součinem $w^T \cdot x$,
- **Aktivační funkce** $\mathbb{R} \rightarrow \{0, 1\}$ – našem případě schodková funkce

$$s_\theta(x) = \begin{cases} 1 & \text{pokud } x \geq \theta, \\ 0 & \text{jinak,} \end{cases} \quad (1.2)$$

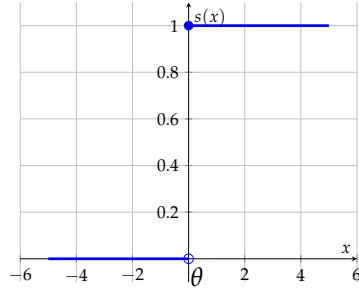
jejíž graf vidíme na obr. 1.3 (pro $\theta = 0$).

Práh excitace lze zapsat jako další váhu. Algoritmus učení pak pracuje s jediným typem parametru a aktivační funkce nemusí mít samostatný práh.

Uvažme, že v perceptronu nahradíme aktivační funkci s_θ (1.2) funkcí s_0

$$s_0(x) = \begin{cases} 1 & \text{pokud } x \geq 0, \\ 0 & \text{jinak,} \end{cases}$$

přidáme jednu vstupní hranu a její váhu nastavíme na $w_0 = -\theta$. Vstupní hodnotou pro tuto hranu bude konstanta 1. Před touto



Obrázek 1.3: Prahová aktivační funkce pro $\theta = 0$.

změnou perceptron vracel 1 právě pro všechny vstupy, pro které platilo

$$\sum_{i=1}^n w_i x_i \geq \theta$$

neboli

$$\sum_{i=1}^n w_i x_i - \theta \geq 0. \quad (1.3)$$

Po změně vrací 1 právě pro všechny vstupy, pro které platí

$$\sum_{i=1}^n w_i x_i + w_0 \cdot 1 \geq 0, \quad (1.4)$$

což je totéž jako (1.3), protože $w_0 = -\theta$. Oba zápisy porovnává obr. 1.4. Ekvivalentně lze rozšířit vstupní prostor. Původní vstup z $\mathbb{R}^n$ se nahradí prvkem $\mathbb{R}^{n+1}$, přesněji $\{1\} \times \mathbb{R}^n$. Každý vektor $\mathbf{x} = \langle x_1, \dots, x_n \rangle$ se rozšíří na $\bar{\mathbf{x}} = \langle x_0, x_1, \dots, x_n \rangle$ (striktněji $\bar{\mathbf{x}} = \langle 1, x_1, \dots, x_n \rangle$). Nerovnice (1.3) a (1.4) jsou pak ekvivalentní s

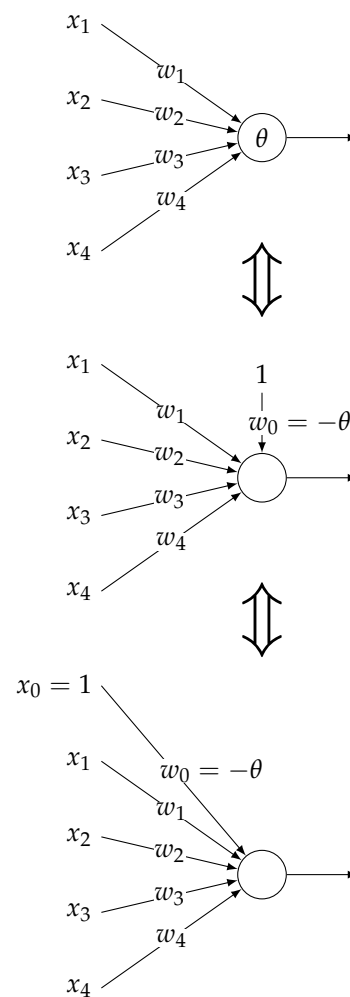
$$\sum_{i=0}^n w_i x_i \geq 0.$$

1.1 Geometrická interpretace a lineární separabilita

Uvažujme perceptron se dvěma vstupy, vektorem vah $\mathbf{w} = \langle a, b \rangle$ a prahem excitace $\theta = -c$. Funkce tohoto perceptronu pak pro vstupní vektor $\mathbf{x} = \langle x, y \rangle$ vrací

$$f(\langle x, y \rangle) = \begin{cases} 1 & \text{pokud } ax + by + c \geq 0, \\ 0 & \text{jinak.} \end{cases}$$

Rovnice $ax + by + c = 0$ je obecná rovnice přímky. Perceptron tedy rozděluje body $\langle x, y \rangle$ vstupního prostoru $\mathbb{R}^2$ podle toho, na které



Obrázek 1.4: Ekvivalentní zápis prahu excitace pomocí biasu a konstantního vstupu.

straně této přímky leží – určuje polorovinu ve $\mathbb{R}^2$. Analogicky perceptron se třemi vstupy rozděluje $\mathbb{R}^3$ rovinou, perceptron se čtyřmi vstupy rozděluje $\mathbb{R}^4$ prostorem, atd. Tento vzhled dává představu, jaké funkce lze perceptronem reprezentovat.

Příklad 1.2: Rozhodovací přímka a dvě klasifikace

Perceptron s $w = (2, -1)$ a prahem $\theta = 1$ má hranici $2x - y - 1 = 0$. Pro bod $(2, 1)$ je vážený součet $3 \geq 1$, a výstup je tedy 1. Pro bod $(0, 1)$ dostaneme $-1 < 1$, a výstup je 0. Změna vah otáčí hranici, změna prahu ji posouvá rovnoběžně.

1.1.1 XOR problém

Které logické funkce lze perceptronem implementovat? Začneme funkcemi dvou proměnných. Obr. 1.5 shrnuje všech 16 Booleovských funkcí dvou proměnných f_0 až f_{15} ; sloupec f_i udává hodnotu funkce pro každou kombinaci proměnných x_1 a x_2 . Například f_0 je nulová funkce a f_{14} je OR. Perceptronem lze spočítat právě ty funkce, u

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

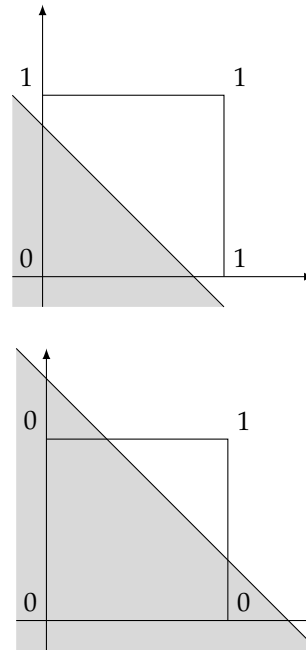
Obrázek 1.5: Pravdivostní tabulka všech šestnácti booleovských funkcí dvou proměnných.

nichž lze body s funkční hodnotou 0 oddělit přímkou od bodů s funkční hodnotou 1. Obr. 1.6 ukazuje takové separace pro funkce OR a AND.

Dvě funkce z tabulky takto vypočítat nelze: XOR a identita (f_6 a f_9). Intuitivně žádná přímka potřebné oddělení nezajistí; dokažme to i analyticky.

Nechť w_1 a w_2 jsou váhy perceptronu se dvěma vstupy a θ jeho práh. Pokud perceptron počítá funkci XOR, musí být splněny následující čtyři nerovnosti:

$$\begin{array}{llll}
 x_1 = 0, x_2 = 0 & w_1 x_1 + w_2 x_2 = 0 & \Rightarrow & 0 < \theta \\
 x_1 = 1, x_2 = 0 & w_1 x_1 + w_2 x_2 = w_1 & \Rightarrow & w_1 \geq \theta \\
 x_1 = 0, x_2 = 1 & w_1 x_1 + w_2 x_2 = w_2 & \Rightarrow & w_2 \geq \theta \\
 x_1 = 1, x_2 = 1 & w_1 x_1 + w_2 x_2 = w_1 + w_2 & \Rightarrow & w_1 + w_2 < \theta
 \end{array}$$



Obrázek 1.6: Lineární separace vstupního prostoru pro funkce OR a AND.

Podle první nerovnosti je θ kladné, podle druhé a třetí jsou tedy kladné i w_1 a w_2 . Nerovnost $w_1 + w_2 < \theta$ pak nemůže platit. Tento spor ukazuje, že žádný perceptron funkci XOR nepočítá. Obdobný důkaz platí i pro funkci f_9 .

Podobně jako XOR nelze prahovým prvkem vypočítat mnoho dalších logických funkcí více proměnných. Souvisí to s geometrií n -dimenzionální hyperkrychle, jejíž vrcholy reprezentují kombinace logických hodnot argumentů: logická funkce rozděluje vrcholy do dvou tříd, a pokud body s funkční hodnotou 1 nelze oddělit lineárním řezem od bodů s hodnotou 0, funkce není perceptronem vypočitatelná. Následující definice dává problému obecnější rámeček.

Definice 1.1

Dvě množiny bodů A a B v n -rozměrném prostoru se nazývají lineárně separovatelné, pokud existuje $n + 1$ reálných čísel $w_1, \dots, w_{n+1}$, takových, že každý bod $(x_1, x_2, \dots, x_n)$ v A splňuje $\sum_{i=1}^n w_i x_i \geq w_{n+1}$ a každý bod $(x_1, x_2, \dots, x_n)$ v B splňuje $\sum_{i=1}^n w_i x_i < w_{n+1}$.

Formule vyjadřující počet lineárně separovatelných funkcí jako funkci n není přes rozsáhlý výzkum dosud známa.

Kolik lineárně separovatelných funkcí binárních argumentů existuje? Pro $n = 2$ je to 14 ze 16 Booleovských funkcí, pro $n = 3$ 104 ze 256 a pro $n = 4$ 1882 z 65536.

1.2 Algoritmus učení perceptronu

Tréninková sada se skládá ze dvou množin P a N v n -rozměrném rozšířeném vstupním prostoru. Hledáme vektor w , který obě množiny absolutně oddělí: všechny vektory z P patří do otevřeného kladného poloprostoru a všechny vektory z N do otevřeného záporného poloprostoru lineární separace.

```
 $P \leftarrow$  Inputs with label 1;  
 $N \leftarrow$  Inputs with label 0;  
Initialize  $w$  randomly;  
while not convergence do  
  Pick random  $x \in P \cup N$ ;  
  if  $x \in P$  and  $w^T \cdot x \leq 0$  then  
     $w \leftarrow w + x$   
  end  
  if  $x \in N$  and  $w^T \cdot x \geq 0$  then  
     $w \leftarrow w - x$   
  end  
end
```

Algoritmus 1: Algoritmus učení perceptronu

Tento algoritmus [MINSKY1969] opraví vektor vah vždy, když vybraný vektor z P nebo N není správně klasifikován. Věta o konvergenci perceptronu zaručuje, že pro lineárně separovatelné množiny P a N je počet aktualizací vektoru w konečný. Proceduru lze zastavit, jakmile jsou všechny vektory klasifikovány správně; o příslušný test je třeba pseudokód doplnit.

Nerovnosti v obou podmínkách jsou *neostré*. Není to nedbalost: nulový skalární součin znamená, že vzor leží přesně na rozhodovací hranici, a to za správné zařazení nepovažujeme — zadání úlohy žádá ostré oddělení. Kdyby podmínka pro P zněla $w \cdot x < 0$, algoritmus by z nulového váhového vektoru vůbec nevyjel; v běhu na následujících stranách je to vidět hned v prvním kroku.

Příklad 1.3: Jeden opravný krok perceptronu

Nechť rozšířený kladný vzor je $x = (1, 2, -1)$ a aktuální váhy $w = (0, -1, 0)$. Protože $w^T x = -2 < 0$, je vzor chybně zařazen. Aktualizace dává

$$w \leftarrow w + x = (1, 1, -1).$$

Pro tentýž vzor je nový skalární součin $1 + 2 + 1 = 4 > 0$. Krok opravil tento vzor, ale mohl změnit klasifikaci jiných bodů; proto algoritmus data opakovaně prochází.

[MINSKY1969] M. Minsky, S. Papert:
Perceptrons: An Introduction to Computational Geometry.
MIT Press, Cambridge, MA (1969)

1.2.1 Ukázka celého běhu: funkce AND

Jeden opravný krok ukazuje mechaniku, ne však průběh učení. Projděme proto celý běh na funkci AND. Použijeme bipolární kódování, $x_i \in \{-1, +1\}$, a rozšířený vstup $\bar{x} = \langle 1, x_1, x_2 \rangle$; množiny vzorů pak jsou

$$P = \{(1, 1, 1)\}, \quad N = \{(1, 1, -1), (1, -1, 1), (1, -1, -1)\}.$$

Začneme s $w_0 = (0, 0, 0)$ a vzory procházíme cyklicky v uvedeném pořadí.

krok	$\bar{x}$	třída	$w \cdot \bar{x}$	akce	nové w
1	(1, 1, 1)	P	0	$w + \bar{x}$	(1, 1, 1)
2	(1, 1, -1)	N	1	$w - \bar{x}$	(0, 0, 2)
3	(1, -1, 1)	N	2	$w - \bar{x}$	(-1, 1, 1)
4	(1, -1, -1)	N	-3	—	(-1, 1, 1)
5	(1, 1, 1)	P	1	—	(-1, 1, 1)
6	(1, 1, -1)	N	-1	—	(-1, 1, 1)
7	(1, -1, 1)	N	-1	—	(-1, 1, 1)

Proběhly tři opravy a po nich celý průchod bez jediné změny vah; to je zastavovací podmínka.

V prvním kroku stojí za povšimnutí, že $w \cdot \bar{x} = 0$: vzor leží na hranici, kterou nulový váhový vektor určuje jen formálně, a opravuje se právě proto, že je podmínka neostrá.

Než se na běh podíváme geometricky, je třeba rozlišit, co je na obrázku vidět a co ne. Váhový vektor má tři složky, $w = (w_0, w_1, w_2)$, a rozhodovací hranice je množina bodů, kde

$$w \cdot \bar{x} = w_0 + w_1 x_1 + w_2 x_2 = 0.$$

Dvojice (w_1, w_2) je normálou této přímky: určuje její směr a to, na které straně leží kladný výstup. Složka w_0 přímkou pouze posouvá, aniž by změnila její směr. Dva stavy učení se proto mohou lišit, přestože mají tutéž normálu — což je přesně případ prvního a třetího panelu níže.

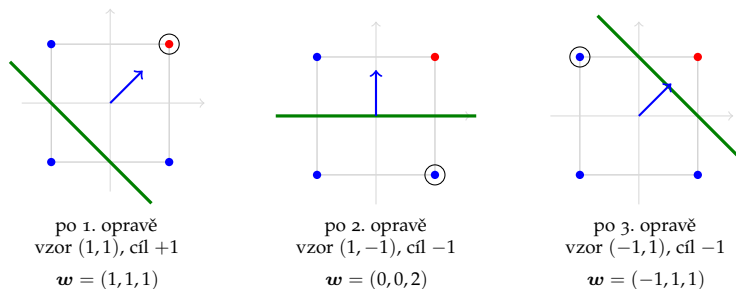
Obratem *oprava vzorem $\bar{x}$* rozumíme ten krok algoritmu, ve kterém byl k úpravě vah použit právě vzor $\bar{x}$. Protože se vzor přičte, respektive odečte, po složkách,

$$w \pm \bar{x} = (w_0 \pm 1, w_1 \pm x_1, w_2 \pm x_2),$$

mění jediná oprava obojí naráz: první složka posune hranici, druhá a třetí otočí normálu. V popiskách obrázku uvádíme u vzoru jen souřadnice (x_1, x_2) ; první složka rozšířeného vstupu je vždy 1.

Obrázek 1.7 zachycuje týž běh geometricky. Zelená přímka je aktuální rozhodovací hranice, modrá šipka normála (w_1, w_2) . Červený

bod má cílovou hodnotu +1, modré body -1; kroužek označuje vzor, kterým se v daném kroku opravovalo. Po první opravě leží na kladné straně ještě tři body, proto opravy pokračují; po třetí opravě už je červený bod na kladné straně sám a celý další průchod daty proběhne beze změny vah.



Obrázek 1.7: Průběh učení perceptronu na funkci AND. Zobrazeny jsou stavy po jednotlivých opravách vah.

Naučené váhy $w = (-1, 1, 1)$ odpovídají prahu $\theta = 1$ a rozhodovací funkci

$$y = \text{sign}(x_1 + x_2 - 1),$$

tedy hranici $x_1 + x_2 = 1$. Kontrola na všech čtyřech vzorech dává net = 1, -1, -1, -3, a tedy výstupy +1, -1, -1, -1; to je přesně funkce AND v bipolárním kódování.

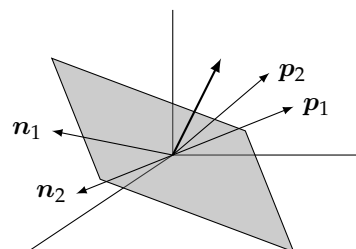
Na čem počet kroků závisí

Na pořadí vzorů, na inicializaci a na měřítku vstupů. Táž úloha zapsaná v binárním kódování, $x_i \in \{0, 1\}$, potřebuje při stejném pořadí dvacet dva oprav místo tří; napříč všemi 24 pořadími vzorů jich je 18 až 22, zatímco v bipolárním kódování nejvýše tři. Důvod je ten, že vzor (0,0) nenese v prvních dvou souřadnicích žádnou informaci, takže z něj oprava mění pouze práh.

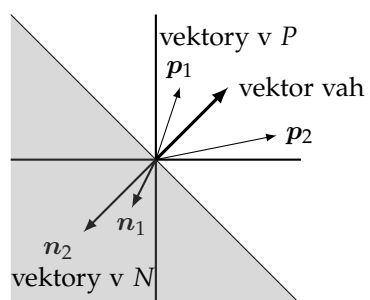
1.2.2 Geometrická vizualizace

Učení perceptronu lze zobrazit přímo v prostoru vstupů nebo v rozšířeném prostoru, kde se hledá nadrovina procházející počátkem, $w_1x_1 + w_2x_2 + w_3 = 0$. Jejím normálovým vektorem je vektor vah $w = (w_1, w_2, w_3)$. Oba pohledy ilustrují obr. 1.8 a 1.9.

Hledáme tedy váhový vektor w s kladným skalárním součinem se všemi rozšířenými vektory z P a záporným se všemi z N . Obecněji: pro množiny P a N n -dimenzionálních vektorů hledáme váhový



Obrázek 1.8: Separace kladných a záporných vstupních vektorů váhovým vektorem.



Obrázek 1.9: Rozhodovací přímka a odpovídající oblasti tříd ve vstupním prostoru.

vektor w takový, že $w \cdot x > 0$ pro všechna $x \in P$ a $w \cdot x < 0$ pro všechna $x \in N$.

Algoritmus začíná s náhodně zvoleným váhovým vektorem w_0 . Pokud pro vektor x z množiny P platí $w \cdot x < 0$, svírají oba vektory úhel větší než 90 stupňů a váhový vektor je třeba otočit směrem k x , aby se x přesunul do kladné poloviny prostoru definovaného vektorem w . Toho algoritmus dosahuje přičtením x k w . Pokud x patří do N a $w \cdot x > 0$, je úhel menší než 90 stupňů a váhový vektor je třeba otočit od x , tedy x odečíst. Vektory z P tak otáčejí váhový vektor jedním směrem, vektory z N opačným. Pokud řešení existuje, algoritmus ho najde po konečném počtu kroků. Dobrou počáteční heuristikou je průměr kladných vstupních vektorů mínus průměr záporných – často leží blízko oblasti řešení.

Nepracujeme nutně s normalizovanými vektory, takže každá aktualizace $w + x$ otáčí váhový vektor o jiný úhel. Pokud x patří do P a $\|x\|$ je mnohem větší než $\|w\|$, je nový váhový vektor $w + x$ téměř roven x . Průběh učení ukazuje obr. 1.10: počáteční váhový vektor je aktualizován přičtením x_1 , x_3 a znovu x_1 . Vektor w se v tomto příkladu po každé korekci zvětšuje, takže každá další korekce jej otáčí o menší úhel a algoritmus pozici váhového vektoru „ladí“. Rychlost učení, tj. míra změny vektoru w , se tak časem zmenšuje a při pokračujícím trénování se i po správném oddělení vektorů blíží k nule; intuitivně naučené vektory zvyšují „inerci“ váhového vektoru. Vektory ležící těsně mimo kladnou oblast jsou do ní přivedeny otočením váhového vektoru právě jen o tolik, kolik je třeba k opravě chyby.

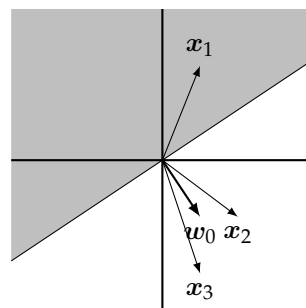
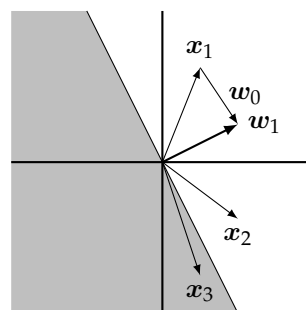
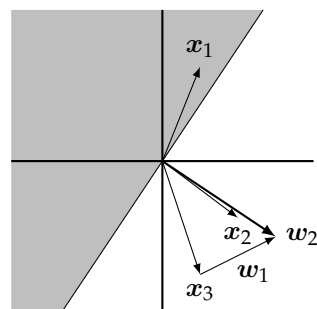
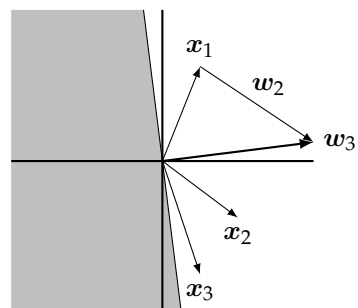
To je typická vlastnost mnoha algoritmů učení neuronových sítí: využívají konstantu učení, která je během učení snižována k nule, aby se naučené konsolidovalo. Algoritmus učení perceptronu poskytuje jakousi automatickou konstantu učení, jež určuje stupeň adaptability vah (tzv. plasticitu sítě).

1.2.3 Konvergence algoritmu

Důkaz konvergence se snáze sleduje s vizualizací z předchozí sekce na paměti.

Věta 1.4

Pokud jsou množiny P a N konečné a lineárně separovatelné, a pokud jsou vektory v nich testovány cyklicky jeden po druhém, váhový vektor w_t je nalezen po konečném počtu kroků t , který může tyto dvě množiny oddělit.



Obrázek 1.10: Geometrický průběh aktualizací během učení perceptronu.

Důkaz

Můžeme provést tři zjednodušení bez ztráty obecnosti:

- Množiny P a N lze spojit do jedné množiny P' , kde množina N^- se skládá z negovaných prvků N .
- Vektory v P' mohou být normalizovány, protože pokud je nalezen váhový vektor w , pro který platí $w \cdot x > 0$, pak to platí i pro jakýkoli jiný vektor ηx , kde η je konstanta.
- Váhový vektor může být také normalizován. Vzhledem k tomu, že předpokládáme, že řešení problému lineární separace existuje, nazveme w^* normalizovaným řešením vektoru.

Předpokládejme, že po $t + 1$ krocích byl vypočítán váhový vektor w_{t+1} . To znamená, že v čase t byl vektor p_i nesprávně klasifikován váhovým vektorem w_t , a proto $w_{t+1} = w_t + p_i$. Kosinus úhlu ρ mezi w_{t+1} a w^* je:

$$\cos \rho = \frac{w^* \cdot w_{t+1}}{\|w_{t+1}\|} \quad (1.5)$$

Pro výraz v čitateli víme, že

$$w^* \cdot w_{t+1} = w^* \cdot (w_t + p_i) = w^* \cdot w_t + w^* \cdot p_i \geq w^* \cdot w_t + \delta$$

kde $\delta = \min\{w^* \cdot p \mid p \in P'\}$. Vzhledem k tomu, že váhový vektor w^* definuje absolutní lineární separaci množin P a N , víme, že $\delta > 0$.

Veličina δ má jasný geometrický význam: protože je w^* normalizovaný, je to nejmenší vzdálenost vzoru od oddělovací nadroviny, tedy míra toho, jak *těsně* řešení data odděluje. Říká se jí *odstup* (anglicky *margin*). Čím je větší, tím rychleji roste pravá strana odhadu a tím dříve musí učení skončit; data oddělitelná jen velmi těsně naopak mohou vyžadovat velmi mnoho kroků. Věta tedy tvrdí, že aktualizací je konečně mnoho, nikoli že je jich málo. Indukcí získáváme

$$w^* \cdot w_{t+1} \geq w^* \cdot w_0 + (t + 1)\delta. \quad (1.6)$$

Na druhou stranu pro výraz ve jmenovateli (1.5) víme, že

$$\|w_{t+1}\|^2 = (w_t + p_i) \cdot (w_t + p_i) = \|w_t\|^2 + 2w_t \cdot p_i + \|p_i\|^2.$$

Protože $w_t \cdot p_i \leq 0$ – jinak by aktualizace pomocí p_i nenastala – platí

$$\|w_{t+1}\|^2 \leq \|w_t\|^2 + \|p_i\|^2 \leq \|w_t\|^2 + 1.$$

Protože všechny vektory v P byly normalizovány, indukci dostáváme

$$\|w_{t+1}\|^2 \leq \|w_0\|^2 + (t + 1). \quad (1.7)$$

Biologický neuron:

Biologická inspirace McCullochova–Pittsova neuronu je pro matematický model perceptronu pouze volná. Podrobnější historický kontext uvádí e-book *Neural Networks: A Systematic Introduction* [ROJAS1996].

Z nerovnic (1.6), (1.7) a rovnice (1.5) dostáváme nerovnost

$$\cos \rho \geq \frac{\mathbf{w}^* \cdot \mathbf{w}_0 + (t+1)\delta}{\sqrt{\|\mathbf{w}_0\|^2 + (t+1)}}$$

Pravá strana roste proporcionálně k $\sqrt{t}$ a jelikož δ je kladná, může nabývat libovolně velkých hodnot. Ale, vzhledem k tomu, že $\cos \rho \leq 1$, musí být omezena maximální hodnotou. Proto musí být počet aktualizací váhového vektoru konečný.

Důkaz ukazuje, že učení perceptronu funguje tak, že přiblíží počáteční vektor $\mathbf{w}_0$ dostatečně blízko k $\mathbf{w}^*$ (protože $\cos \rho$ se zvětšuje a ρ se proporcionálně zmenšuje).

[ROJAS1996] Raúl Rojas
Neural Networks: A Systematic Introduction
Springer Berlin, 1996

1.3 Historie perceptronu

Historie perceptronu vychází z neurologických studií, které vedli především Santiago Ramón y Cajal a Sir Charles Scott Sherrington. Ramón y Cajal při zkoumání struktury nervové tkáně ukázal, že neurony jsou fyzicky oddělené, přestože spolu komunikují. To vedlo k pokusům popsat, jak mohou jednotlivé neurony dávat vzniknout pozorovatelnému chování.

[McCULLOCH1943] W.S. McCulloch a W. Pitts.
A logical calculus of the ideas immanent in nervous activity.
The bulletin of mathematical biophysics 5 (1943): 115-133.

V roce 1943 publikovali Warren McCulloch a Walter Pitts práci *A Logical Calculus of Ideas Immanent in Nervous Activity* [McCULLOCH1943], podle níž jsou neurony s binární prahovou aktivační funkcí analogické větám logiky prvního řádu. Neuron McCulloch-Pitts pracoval s binárními vstupy (1 pravda, 0 nepravda) a prahovou hodnotou určující výstup.

[HEBB1949] D.O. Hebb.
The organization of behavior.
New York (1949).

V roce 1949 formuloval Donald Hebb v knize *The Organization of Behavior* [HEBB1949] Hebbovo pravidlo: je-li axon buňky A dostatečně blízko, aby excitoval buňku B , a opakovaně ji aktivuje, dojde v jedné nebo obou buňkách k růstu či metabolické změně, která zvýší účinnost A při aktivaci B . Z pravidla plynulo, že McCulloch-Pittsův neuron je třeba upravit tak, aby umožnil vážené vstupy.

[ROSENBLATT1962] F. Rosenblatt
Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. Vol. 55.
Washington, DC: Spartan books, 1962.

Na McCulloch-Pittsův neuron a Hebbovy poznatky navázal Frank Rosenblatt vývojem prvního perceptronu, který se dokázal učit vážením vstupů; první popis publikoval v roce 1958 [ROSENBLATT1958] a později jej rozpracoval v knize *Principles of Neurodynamics* [ROSENBLATT1962]. Základní perceptron vstupy vážil a sumoval v uzlu a při dosažení prahové hodnoty generoval výstup; váhy jednotlivých vstupů přitom nemusely být stejné.

[MINSKY1969] M. Minsky, S. Papert:
Perceptrons: An Introduction to Computational Geometry.
 MIT Press, Cambridge, MA (1969).

Omezení perceptronu kritizovali Marvin Minsky a Seymour Papert v knize *An Introduction to Computational Geometry* [MINSKY1969]: ukázali, že perceptron řeší pouze lineárně separabilní funkce (nikoli např. XOR). Výzkum perceptronů byl proto na léta přerušen a obnoven až v 80. letech s rozvojem komplexnějších neuronových sítí.

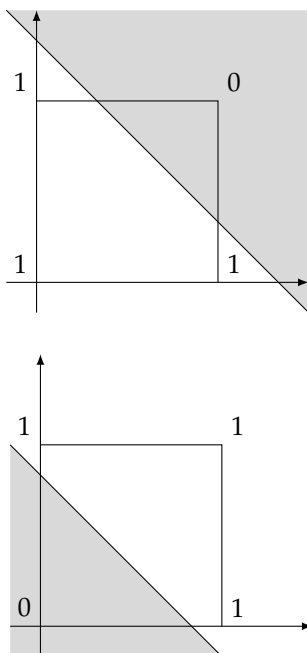
1.3.1 Řešení XOR problému

Jediný perceptron funkci XOR nerepresentuje, síť více perceptronů však ano. Funkce AND, NAND a OR (v tabulce f_8 , f_7 a f_{14}) jsou lineárně separabilní a pro každé $a, b \in \{0, 1\}$ platí

$$a \text{ XOR } b = (a \text{ NAND } b) \text{ AND } (a \text{ OR } b).$$

(XOR by šel poskládat i jen z funkcí NAND, tato varianta je ale přehlednější.)

Dva perceptrony spočítají NAND a OR, jejich výstupy zpracuje třetí perceptron AND, na jehož výstupu obdržíme XOR původních vstupů. Lineární separace prvních dvou funkcí jsou na obr. 1.11.



Obrázek 1.11: Lineární separace vstupního prostoru pro funkce NAND a OR.

