BI-VZD přednáška 11

Daniel Vašata

FIT ČVUT

9. 12. 2021

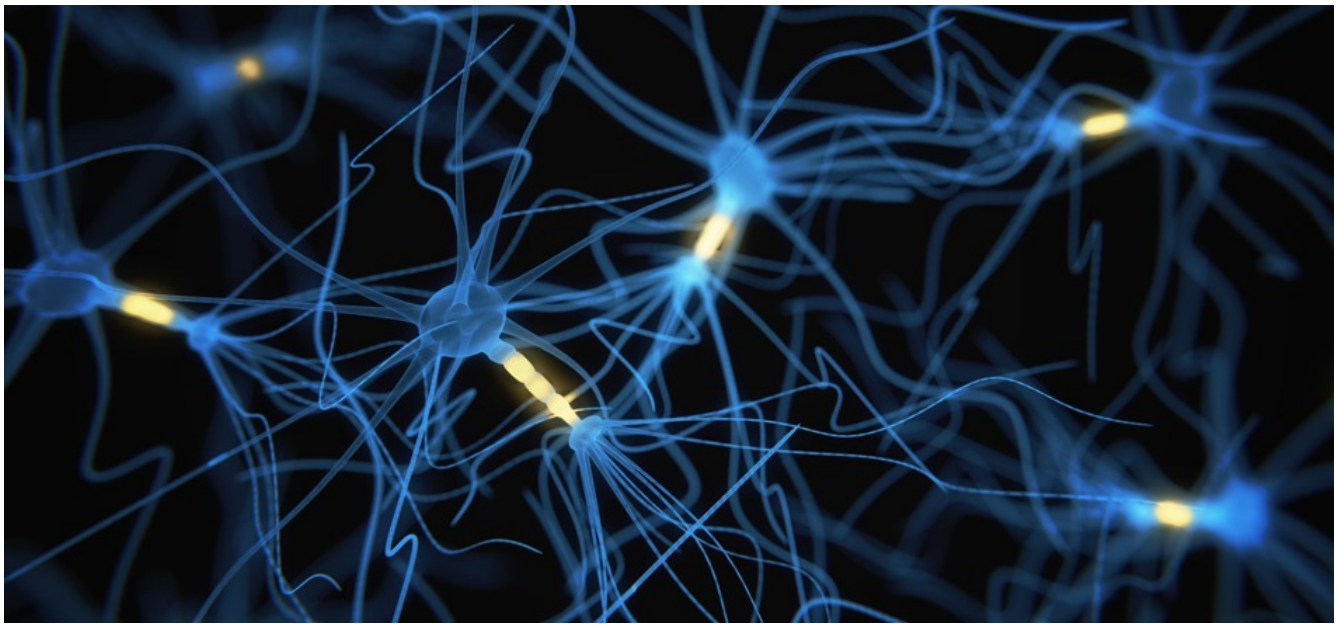
Autoři: Karel Klouda, Juan Pablo Maldonado Lopez, Daniel Vašata.
Problémy, návrhy apod. hlaste v GitLabu.
Verze souboru: 9. prosince 2021 12:56.

Co bude v dnešní přednášce

- Stručný úvod do neuronových sítí
- Učení neuronových sítí
- Speciální typy neuronových sítí:
  - autoenkodéry
  - konvoluční sítě
  - rekurentní sítě

Neuronové sítě

- Inspirace strukturou vzájemně projených neuronů v biologických systémech.

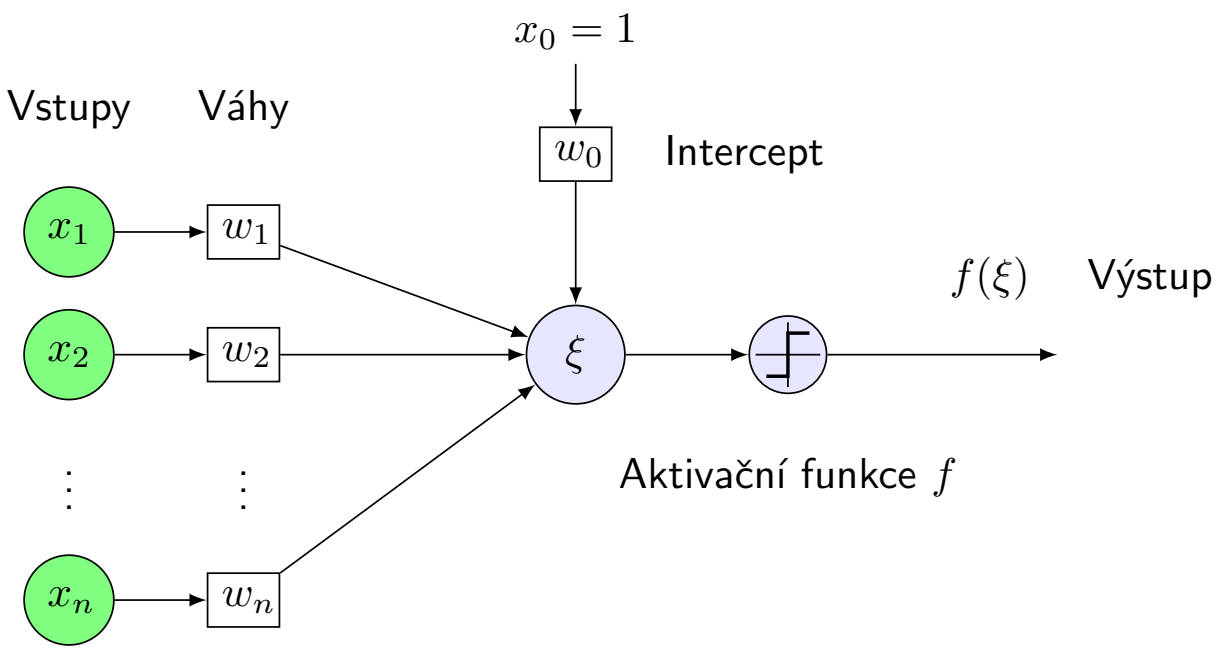


  - Neuron, jakožto základní stavební kámen, je jednoduchá výpočetní jednotka.
  - Přijímá několik reálných vstupů.
  - Produkuje jeden reálný výstup.

- Prvotní model neuronu představili již v roce 1943 McCulloch a Pitts.

Perceptron (1/3)

Nejjednodušším modelem neuronové sítě určeným ke klasifikaci[1] je tzv. **jednovrstvý perceptron** (angl. **single-layer perceptron**), který se skládá z jediného umělého neuronu (angl. artificial neuron):



Výstup neuronu se získá aplikací nelineární **aktivační funkce** f na hodnotu **vnitřního potenciálu** ξ daného součtem vstupů $x_1, \ldots, x_n$ pronásobených příslušnými vahami $w_1, \ldots, w_n$ a interceptu w_0 (angl. **bias**).

[1] F. Rosenblatt 1957

Perceptron (2/3)

- Vnitřní potenciál tedy spočteme vztahem

$$\xi = w_0 + \sum_{i=1}^{n} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} + w_0,$$

kde značíme $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ a $\boldsymbol{w} = (w_1, \ldots, w_n)^T$.

- Výstup perceptronu je pak určen jako

$$\hat{Y} = f(\xi) = f\big(\boldsymbol{w}^T \boldsymbol{x} + w_0\big),$$

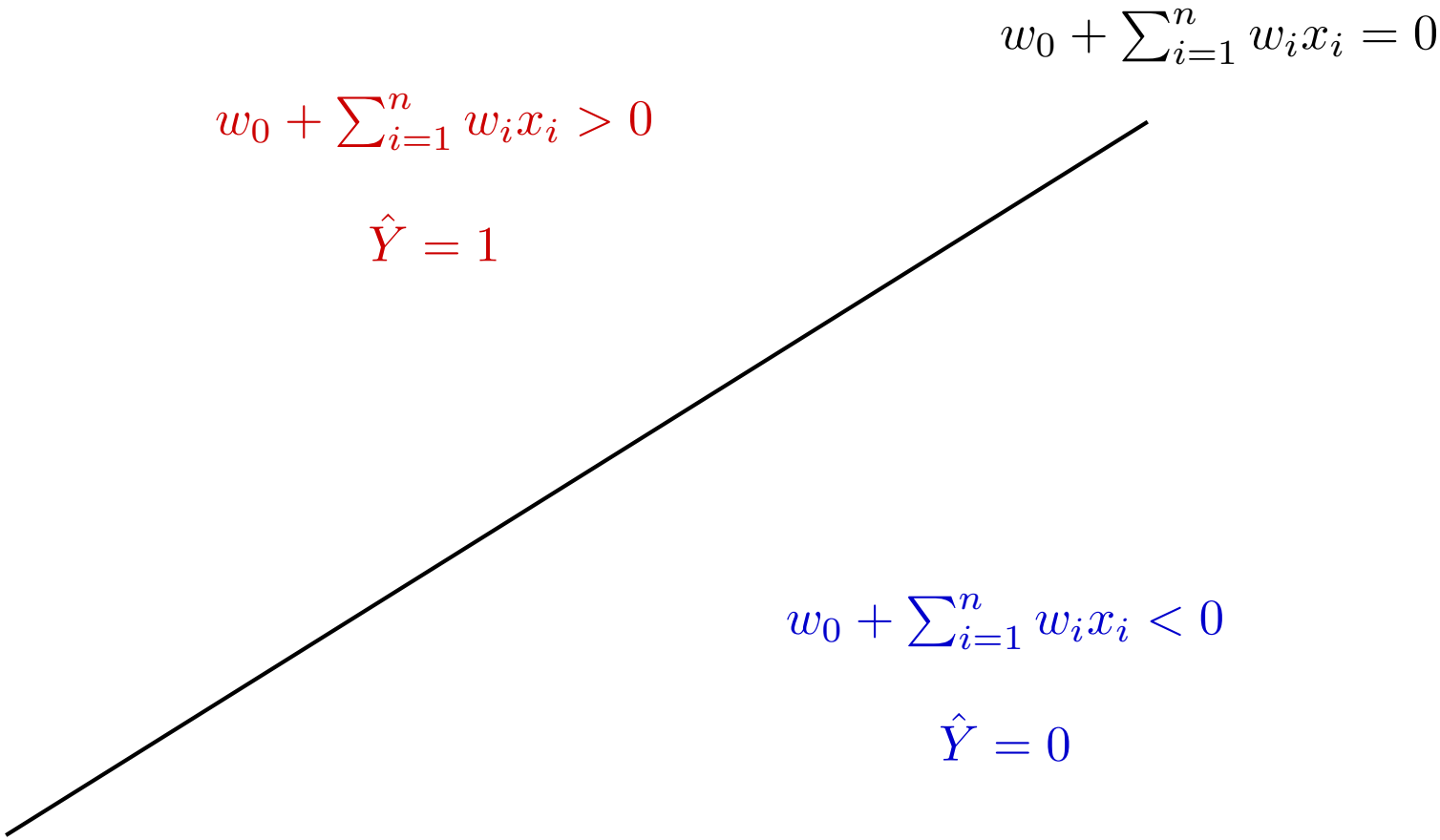
kde funkce f je tzv. **aktivační funkce** (angl. **activation function**), která je v případě perceptronu dána jako skoková funkce

$$f(\xi) = \begin{cases} 1 & \text{když } \xi \geq 0, \\ 0 & \text{když } \xi < 0. \end{cases}$$

- Neuron je tedy aktivován, $f(\xi) = 1$, pokud je $\sum_{i=1}^{n} w_i x_i \geq -w_0$. Proto se někdy hodnota $-w_0$ nazývá **prahová**.

Perceptron (3/3)

Pro lepší pochopení reakce perceptronu na vstupy si ukažme jednoduchý obrázek.



Prostor $\mathbb{R}^n$ vstupů $(x_1, \ldots, x_n)^T$ je tedy rozdělen na dva lineárně separované poloprostory, kdy v jednom z nich je neuron aktivní a v druhém ne. Váhy $w_0, w_1, \ldots, w_n$ pak určují nadrovinu, která tyto dva poloprostory separuje.

Dopředný a zpětný chod

- Pro zadané váhy $w_0, w_1, \ldots w_n$ můžeme na základě hodnot vstupu $\boldsymbol{x}$ spočítat výstup.
- Tento výpočet se nazývá **dopředný chod** (angl. **forward pass**).
- Jak ale tyto váhy získáme?
- V procesu učení potřebujeme, aby se shodoval výstup neuronu se skutečností.
- Na základě hodnot získaných v dopředném chodu určíme chybu predikce a na jejím základě provedeme inkrementální update vah:

$$\begin{aligned} \text{error} &= Y - \hat{Y} \\ w_i &\leftarrow w_i + \text{error} \cdot x_i, \quad i = 1, \ldots, n \\ w_0 &\leftarrow w_0 + \text{error} \end{aligned}$$

  kde $\hat{Y} = f\left(\boldsymbol{w}^T \boldsymbol{x} + w_0\right)$ je predikce v bodě $\boldsymbol{x}$ trénovací množiny a Y je skutečná hodnota.
- Tento update se nazývá **zpětný chod** (angl. backward pass).
- Stejná myšlenka se používá i v případě komplexních sítí (není vázána na jednovrstvý perceptron).

Jak moc jsou perceptrony užitečné?

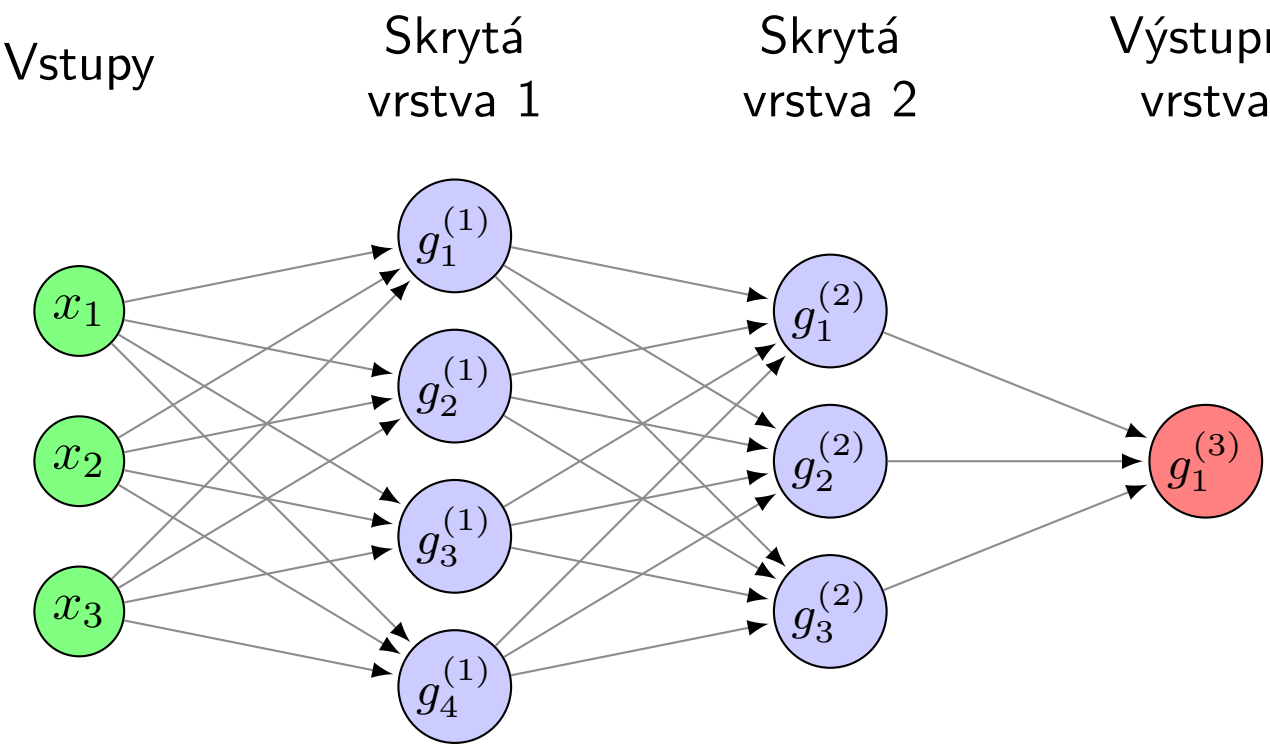
- Moc ne! Dokážou reprezentovat pouze lineární funkce.
- Jednoduchý příklad, který perceptron nedokáže vyřešit: funkce XOR.

id	X_1	X_2	Y
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

- Nalezení vah optimálního perceptronu odpovídá nalezení přímky (nadroviny), která odseparuje body různých tříd na různé strany. To ale nejde!
- Tyto body je možné zanořit v $\mathbb{R}^3$ takovým způsobem, že jsou lineárně separabilní! (Přidáme např. $X_3 = X_1$ AND X_2.)
- To nám napovídá, že s využitím skládání perceptronů by se daly získat mnohem lepší modely.

Vícevrstvá neuronová síť

- Základním rozšířením jednovrstvého perceptronu je takzvaný **vícevrstvý perceptron** (angl. **multilayer perceptron** aka **MLP**, **feedforward neural network**).
- V tomto modelu se neuronová síť skládá z vrstev, které jsou propojené tak, že výstupy neuronů z jedné vrstvy tvoří vstupy neuronů do další vrstvy.
- Všechny vrstvy, kromě té výstupní, se nazývají **skryté** (angl. **hidden layers**).



Matematický model vícevrstvé neuronové sítě

- Uvažujme l vrstvou neuronovou síť a označme $n_1, \ldots, n_l$ počty neuronů v jednotlivých vrstvách. Dále označme počet vstupních proměnných jako n_0.
- Uvažujme $i$tou vrstvu této sítě. Výstup j-tého neuronu může reprezentovat funkcí $g_j^{(i)} : \mathbb{R}^{n_{i-1}} \to \mathbb{R}$, která má na vstupu výstupy n_{i-1} neuronů z předchozí vrstvy.
- Interně se $g_j^{(i)}(\boldsymbol{x})$ opět počítá jako $f(\boldsymbol{w}^T \boldsymbol{x} + w_0)$, kde f je aktivační funkce daného neuronu a $w_0, w_1 \ldots, w_n$ jsou jeho váhy.
- i-tou vrstvu této neuronové sítě jako celek pak můžeme chápat jako vícehodnotovou funkci $\boldsymbol{g}^{(i)} : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, kde $\boldsymbol{g}^{(i)} = \left(g_1^{(i)}, \ldots, g_{n_i}^{(i)}\right)^T$.
- Celá neuronová síť při dopředném chodu je tedy reprezentována funkcí $\boldsymbol{g} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$, která vznikne složením jednotlivých vrstev

$$\boldsymbol{g} = \boldsymbol{g}^{(l)} \circ \boldsymbol{g}^{(l-1)} \circ \ldots \circ \boldsymbol{g}^{(2)} \circ \boldsymbol{g}^{(1)}.$$

- Např. pro $l = 3$ tak máme

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{g}^{(3)}\left(\boldsymbol{g}^{(2)}\left(\boldsymbol{g}^{(1)}(\boldsymbol{x})\right)\right).$$

Role skrytých vrstev

- Skryté vrstvy plní roli příznaků pro další vrstvy. V problému XOR dokáže skrytá vrstva zajistit transformaci do souřadného systému, kde jsou již body separabilní.
- Matematicky lze ukázat (věta o univerzální aproximaci, 1989), že síť s jednou skrytou vrstvou dokáže na výstupu s libovolnou přesností aproximovat jakoukoliv spojitou funkci s kompaktním nosičem v $\mathbb{R}^n$.
- V praxi to nefunguje úplně optimálně: pro získání dobré aproximace je zapotřebí obrovské množství neuronů. Přitom přidání dalších neuronů snižuje výkonnost.
- Ve skutečnosti tedy preferujeme spíš hlubší sítě (více skrytých vrstev). Intuitivně můžeme říci, že hlubší sítě vytvářejí sofistikovanější a více zajímavé příznaky.
- **Hluboké učení** (angl. **deep learning**) pracuje s neuronovými sítěmi (deep feedforward networks) s více než třemi až pěti skrytými vrstvami (neexistuje žádný konsenzus).

Učení vícevrstvých sítí

- Zvýšení počtu vrstev zvyšuje flexibilitu sítě. Problémem je ale učení!
- Pokud bychom vrstvy poskládali z perceptronů, který jsme si doteď ukazovali, nedokázali bychom jednoduchým způsobem provádět update vah (vztah pro jeden perceptron se nedá snadno rozšířit na více).
- Je možné používat black-box optimalizační metody, jako jsou například genetické algoritmy.
- Tento přístup však pro velké množství parametrů a hluboké sítě není efektivní, což v 60tých letech vedlo ke zklamání a vlně skepse, která rozvoj neuronových sítí utlumila.
- Klíčová změna pohledu, která vedla k obnovení zájmu o neuronové sítě v 80tých letech, spočívala ve vyvinutí učícího algoritmu **zpětného šíření** chyby (angl. **back-propagation**).
- Pro tento přístup požadujme, aby byla neuronová síť jako funkce parametrů skoro všude diferencovatelná, a použijme gradientní sestup!

Aktivační funkce pro skryté vrstvy

Diferencovatelnosti docílíme volbou vhodných aktivačních funkcí namísto předchozí skokové.

Příklady využívaných aktivačních funkcí pro skryté vrstvy jsou:

- **Oříznutá lineární funkce** (angl. rectified linear unit) – **RELU**

$$f(\xi) = \max(0, \xi) = \begin{cases} \xi & \text{pro } \xi \geq 0, \\ 0 & \text{pro } \xi < 0. \end{cases}$$

  Tato funkce sice není diferencovatelná v 0, ale důležité je, že má v kladném oboru nenulovou derivaci.

- **Hyperbolický tangens**

$$f(\xi) = \tanh(\xi) = \frac{\mathrm{e}^{\xi} - \mathrm{e}^{-\xi}}{\mathrm{e}^{\xi} + \mathrm{e}^{-\xi}}.$$

Aktivační funkce pro výstupní vrstvy

Ve výstupní vrstvě je naším cílem převést hodnoty napočítané v předchozích vrstvách na hodnoty použitelné k predikci. Nejtypičtější jsou následující 3 scénáře:

- Regresní úloha - predikce spojité proměnné. Ve výstupní vrstvě bereme jeden neuron bez aktivační funkce (resp. s aktivační funkcí $f(\xi) = \xi$).
- Binární klasifikace - predikce hodnoty 0 nebo 1. Ve výstupní vrstvě máme jeden neuron jehož aktivační funkce je **logistická funkce**, **sigmoida**,

$$f(\xi) = \frac{1}{1 + \mathrm{e}^{-\xi}} = \frac{\mathrm{e}^{\xi}}{1 + \mathrm{e}^{\xi}},$$

  jejíž hodnotu interpretujeme jako pravděpodobnost příslušnosti ke třídě 1, tj. jako $\hat{\mathrm{P}}(Y = 1|\boldsymbol{X} = \boldsymbol{x})$.

  Predikce v bodě $\boldsymbol{x}$ je potom $\hat{Y} = 1$ když $\hat{\mathrm{P}}(Y = 1|\boldsymbol{X} = \boldsymbol{x}) > 0.5$ jinak $\hat{Y} = 0$.

- Klasifikace do c tříd - predikce hodnot $1, \ldots, c$. Ve výstupní vrstvě máme c neuronů s normovanou aktivační funkcí **softmax**,

$$f_i(\boldsymbol{\xi}) = \frac{\mathrm{e}^{\xi_i}}{\mathrm{e}^{\xi_1} + \ldots + \mathrm{e}^{\xi_c}},$$

  kde $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_c)^T$ je vektor vnitřních potenciálů c neuronů a $f_i(\boldsymbol{\xi})$ je aktivační funkce i-tého neuronu jejíž výstup interpretujeme jako pravděpodobnost příslušnosti ke třídě i, tj. jako $\hat{\mathrm{P}}(Y = i|\boldsymbol{X} = \boldsymbol{x})$.

  Predikce v bodě $\boldsymbol{x}$ je potom $\hat{Y} = \arg\max_{i \in 1,\ldots,c} \hat{\mathrm{P}}(Y = i|\boldsymbol{X} = \boldsymbol{x})$.

Ztrátová funkce

Při učení se snažíme minimalizovat chybu predikce měřenou pomocí průměrné hodnoty **ztrátové funkce** L na trénovací množině.

Ztrátová funkce vhodným způsobem měří, jak dobře daný model predikuje konkrétní hodnotu z trénovací množiny.

Nejpoužívanější ztrátové funkce pro běžné typy úloh:

- Regresní úloha - kvadratická ztrátová funkce (angl. **squared error**)

$$L(Y, \hat{Y}) = (Y - \hat{Y})^2.$$

- Binární klasifikace - binární relativní entropie (angl. **binary cross-entropy**). Označme $\hat{p} = \hat{\mathrm{P}}(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$.

$$L(Y, \hat{p}) = -Y \log \hat{p} - (1 - Y) \log(1 - \hat{p}).$$

- Klasifikace do c tříd - kategorická relativní entropie (angl. **categorical cross-entropy**) označme $\hat{p}_i = \hat{\mathrm{P}}(Y = i | \boldsymbol{X} = \boldsymbol{x})$ a $\hat{\boldsymbol{p}} = (\hat{p}_1, \ldots, \hat{p}_c)^T$:

$$L(Y, \hat{\boldsymbol{p}}) = -\sum_{j=1}^{c} \mathbb{1}_{Y=j} \log \hat{p}_j = -\log \hat{p}_Y,$$

kde $\mathbb{1}_{Y=j} = 1$ když $Y = j$ a $\mathbb{1}_{Y=j} = 0$ jinak.

Učení gradientním sestupem - formulace

- Při učení se snažíme minimalizovat chybu predikce měřenou pomocí průměrné hodnoty ztrátové funkce L na trénovací množině.
- Minimalizujeme tedy

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L\left(Y_i, g(\boldsymbol{x}_i; \boldsymbol{w})\right)$$

  vzhledem k parametrům sítě $\boldsymbol{w}$, které se vyskytují ve funkci g.
- Minimalizaci provádíme gradientním sestupem, resp. nějakou jeho vylepšenou variantou.
- Při výpočtu gradientu se využívá pravidla pro výpočet derivace složené vícehodnotové funkce, který je přímým zobecněním vztahu pro výpočet derivace složené funkce.
  Pro $f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{g} : \mathbb{R} \to \mathbb{R}^n$, kde $\boldsymbol{g} = (g_1, \ldots, g_n)^T$ tak platí

$$\frac{\partial f \circ \boldsymbol{g}}{\partial x}(x) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\boldsymbol{g}(x)) \frac{\partial g_i}{\partial x}(x).$$

- Při výpočtu tedy dochází k postupnému pronásobování a sčítání parciálních derivací vrstev ve směru od výstupní vrstvy směrem k vstupní - tedy k tzv. **zpětnému šíření** (angl. **back-propagation**).

Zpětné šíření chyby - příklad

Uvažujme regresní úlohu, ve které máme na vstupu dva příznaky X_1, X_2, které načítají 2 neurony v první vrstvě s aktivační funkcí f a hodnoty těchto neuronů jsou dále vstupem do jednoho neuronu bez aktivační funkce.
Označíme-li váhy i-tého neuronu v první vrstvě jako $\boldsymbol{w}_i^{(1)} = \left(w_{i;1}^{(1)}, w_{i;2}^{(1)}\right)^T$, intercept jako $w_{i;0}^{(1)}$ a váhy neuronu v druhé vrstvě jako $w_1^{(2)}, w_2^{(2)}$, intercept jako $w_0^{(2)}$, platí

$$g(\boldsymbol{x}; \boldsymbol{w}) = w_1^{(2)} \cdot f\left(\boldsymbol{x}^T \boldsymbol{w}_1^{(1)} + w_{1;0}^{(1)}\right) + w_2^{(2)} \cdot f\left(\boldsymbol{x}^T \boldsymbol{w}_2^{(1)} + w_{2;0}^{(1)}\right) + w_0^{(2)}.$$

Při kvadratické ztrátové funkci tedy minimalizujeme

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \left(Y_i - g(\boldsymbol{x}_i; \boldsymbol{w})\right)^2.$$

Vybrané složky gradientu jsou:

$$\frac{\partial J}{\partial w_1^{(2)}} = \frac{1}{N} \sum_{i=1}^{N} 2(Y_i - g(\boldsymbol{x}_i; \boldsymbol{w})) \cdot (-1) \cdot f\left(\boldsymbol{x}^T \boldsymbol{w}_1^{(1)} + w_{1;0}^{(1)}\right),$$

$$\frac{\partial J}{\partial w_{1;1}^{(1)}} = \frac{1}{N} \sum_{i=1}^{N} 2(Y_i - g(\boldsymbol{x}_i; \boldsymbol{w})) \cdot (-1) \cdot w_1^{(2)} \cdot f'\left(\boldsymbol{x}_i^T \boldsymbol{w}_1^{(1)} + w_{1;0}^{(1)}\right) \cdot x_{i;1}.$$

Učení gradientním sestupem - dávkové učení

- Standardní gradientní sestup známý jako **dávkové učení** (angl. **batch training**) počítá chybu přes celou trénovací množinu a pak teprve provede krok proti směru gradientu.
- To v případě velkých datasetů a velkých množstvích parametrů představuje problém z pohledu paměti.
- Využívají se tedy různá speciální trénovací schémata.
- Například se trénuje v takzvaných **minibatches** (výběr z několika bodů a pak update vah) nebo se v extrémním případě dokonce update provádí pro každý bod zvlášť, tzv. **online training**.

Dávkové učení neuronové sítě

- Máme neuronovou síť s parametry $\boldsymbol{w} = (w_1, \ldots, w_m)^T$ a trénovací data $(Y_1, \boldsymbol{x}_1), \ldots, (Y_N, \boldsymbol{x}_N)$.
- Inicializujeme všechny váhy $\boldsymbol{w}$ jako malá náhodná čísla.
- Opakujeme dokud nejsou splněna kritéria zastavení:
  - Pro trénovací množinu spočteme průměrnou chybu predikce

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L\left(Y_i, g(\boldsymbol{x}_i; \boldsymbol{w})\right)$$

  - Spočteme gradient

$$\nabla_{\boldsymbol{w}} J = \left(\frac{\partial J}{\partial w_1}, \ldots \frac{\partial J}{\partial w_m} \right)^T.$$

  - Provedeme přepočet vah

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} J.$$

Konvergence

- Gradientní sestup míří v prostoru parametrů do míst, kde je gradient nulový.
- Pro jednovrstvou síť se jedná o globální minimum. Pro vícevrstvé sítě můžeme uváznout v lokálním minimu a nebo dokonce v sedlovém bodě (to je horší).
- Pro sítě s velkým množstvím neuronů (a tedy parametrů) se naštěstí ukazuje, že výše uvedené situace nejsou moc velký problém, protože jsou lokální minima a sedlové body v prostoru parametrů velmi řídce rozmístěné a především je v nich zkoumaná funkční hodnota často blízká globálnímu minimu.
- Existuje mnoho konkrétních pravidel pro update vah založených na gradientním sestupu, které jsou v různých situacích lepší než ostatní (RMSprop, Adam, Adagrad). Žádná z metod ale není obecně nejlepší.
- V případě více vrstev se využívají speciální datové struktury na výpočty gradientu (computational graphs).

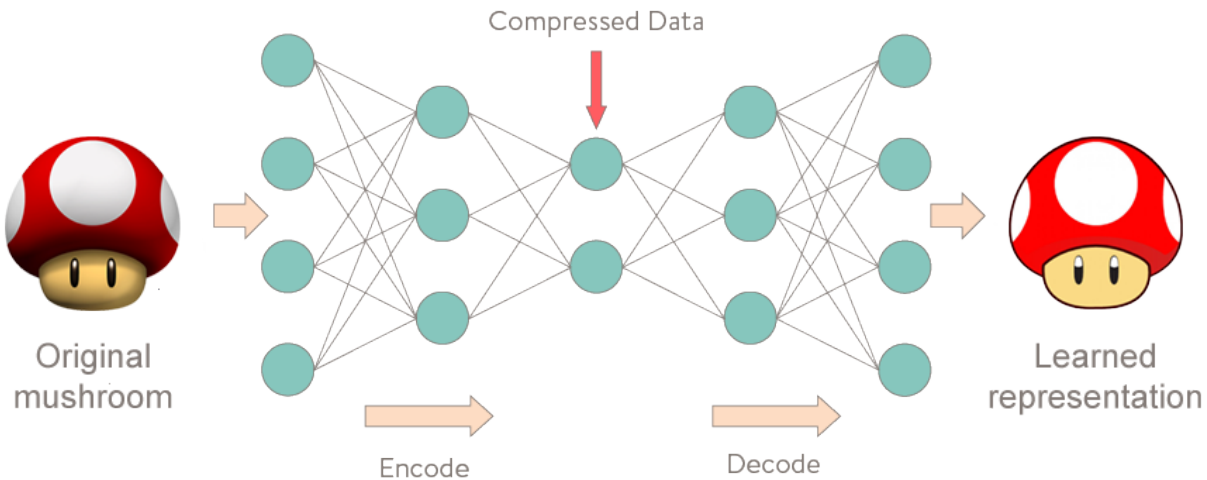
Problémy v hlubokých neuronových sítích

- Přidávání dalších vrstev může vést k přeučování (zvyšuje se počet parametrů).
- Často se tedy v jednotlivých vrstvách používají různé regularizace. Například se k účelové funkci přidává člen penalizující velikosti vah všech neuronů v dané vrstvě.
- Další používánou formou regularizace je dropout (náhodné vynulování některých neuronů), který může velmi pomoci.
- Obvykle je pro učení rozsáhlé sítě potřeba velké množství dat!
- Obvyklým přístupem k nedostatku dat je využít tzv data augmentation: vytvořit více trénovacích dat nějakými poruchami existujících. V kontextu počítačového vidění to znamená např. otáčením, posunutím, přeškálováním obrázků.
- U některých speciálních typů úloh pak existují předtrénované sítě (počítačové vidění, zpracování textu), které je možné využít.

Autoenkodéry

- **Autoenkodér** se skládá ze dvou částí: enkodéru a dekodéru.
- Cílem je vytvořit model, který bude v části enkodéru vytvářet abstraktní příznaky (tzv. kód), které pak v dekodéru dokáže vrátit zpět do původního prostoru příznaků a replikovat co nejvěrněji vstupní data.
- Toto může být velmi užitečné např. pro kompresi dat a tedy pro redukci dimenzionality.
- Dalším využitím je detekce odlehlých hodnot: Bod považujeme za odlehlou hodnotu, pokud je jeho rekonstrukční chyba velká. To funguje, protože se autoenkodér naučí především vlastnosti těch obvyklých bodů. Odlehlé body se pak vymykají jak v enkódovací, tak v dekódovací části.

Autoenkodéry

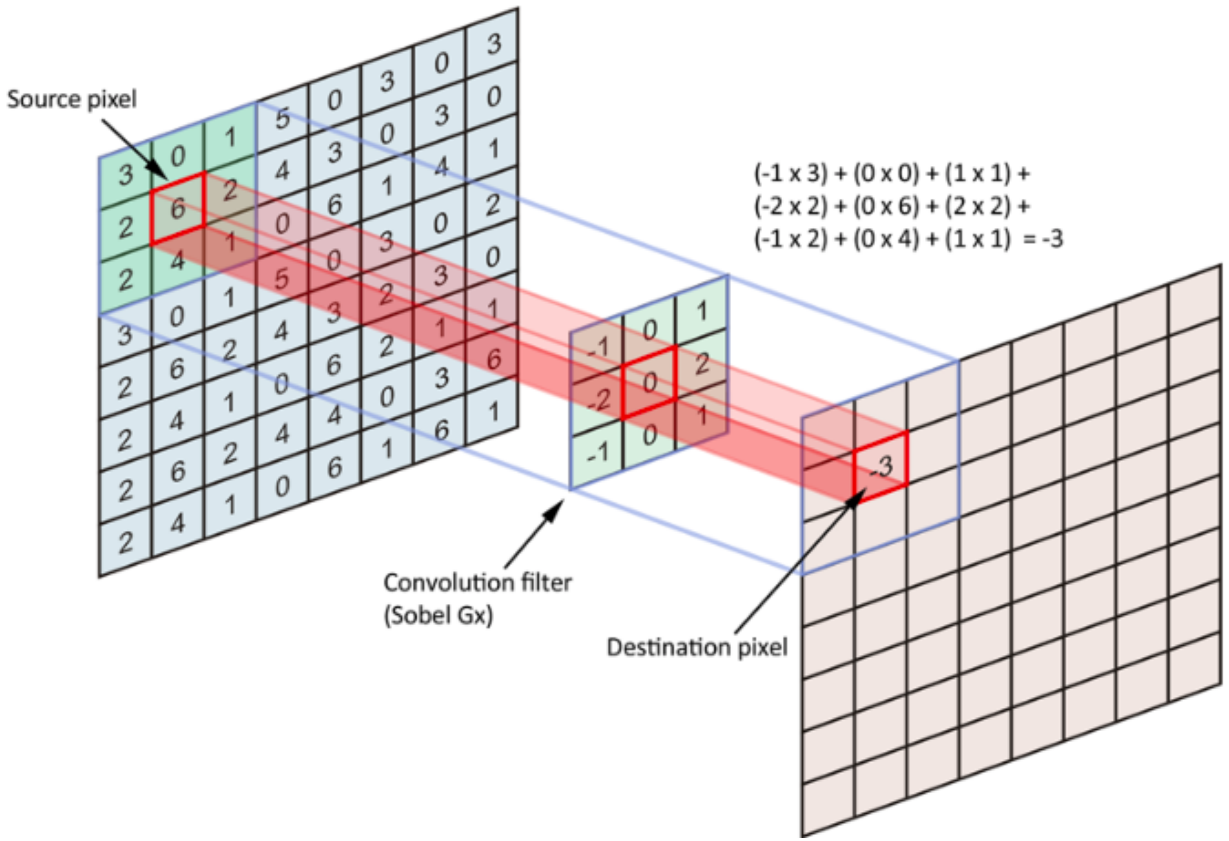


Vizualizace autoenkodéru.

Konvoluční neuronové sítě

- Ačkoliv už dnes počítač umí porážet šachové velmistry, do nedávné doby neuměl na obrázku rozpoznat například štěňátko.
- Proč to tak je? Zdá se, že vnímání se děje mimo senzorické rozpoznávání: tj. v našem mozku jsou velmi abstraktní neurony (příznaky), které jsou vysoko nad senzorickým vnímáním.
- Z experimentů na kočkách v 50tých letech (Hubel a Wiesel, oba obdrželi Nobelovu cenu v medicíně) víme, že neurony ve vizuální oblasti spolupracují pouze lokálně. Teprve z těchto lokálních oblastí se skládá celkový obrazový vjem.
- Hluboké učení nabízí jako možné východisko z tohoto problému **konvoluční neuronové sítě**. V zásadě jde o sady vrstev, které počítají vhodné **diskrétní konvoluce** vstupů. V trénovací fázi se navíc učí, jaké jsou ty vhodné.

Konvoluční neuronové sítě

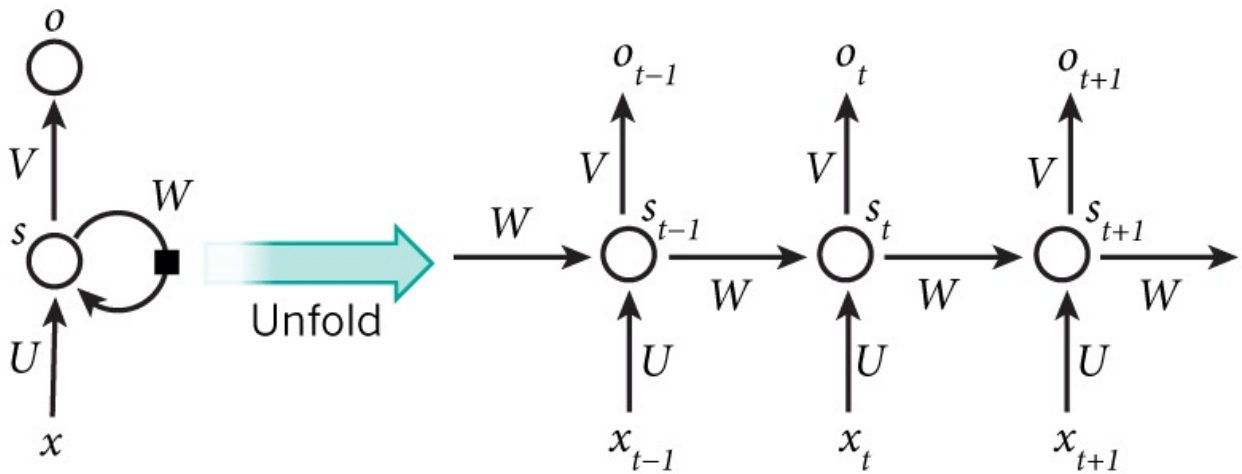


Příklad výpočtu konvoluce mezi sousedními vrstvami CNN.

Rekurentní neuronové sítě

- Používají se na zpracování posloupností dat: časové řady, analýza textu, skládání hudby atd., často různých délek.
- Při výpočtu výstupu určitých neuronů se nevyužívají pouze vstupy z daného časového kroku, ale také výstupy předchozího časového kroku.
- Není v nich tedy jednoduchý jednosměrný tok informací: vyskytují se zde skryté stavy.
- To umožňuje síti podchytit dlouhé časové závislosti.

Rekurentní neuronové sítě



Jednoduchý příklad RNN a "rozbalení"výpočetního grafu.

Závěrečné poznámky

- Téma neuronových sítí je obrovské a v budoucích kurzech se na něj určitě narazíte. Některé se mu dokonce budou věnovat speciálně (např. MI-TNN).

- Na zmínky o neuronových sítích v dnešní době narazíte velmi často. Jsou-li správně využity, tvoří třídu modelů, které jsou vhodné na široké spektrum problémů a dosahují zde excelentních výsledků.

- Existuje několik knihoven specializovaných na práci s neuronovými sítěmi. Asi nejznámější z nich jsou TensorFlow (High level API Keras) a PyTorch.

Další informace naleznete například zde:

- I. Goodfellow, Y. Bengio, A. Courville: Deep Learning, 2016, MIT Press (www.deeplearningbook.org)
- A. Geron, *Hand-On Machine Learning with Scikit-Learn and Tensorflow*, Chapter 10, 13-15, 2017.
- Karpathy, A., The Unreasonable Effectiveness of Recurrent Neural Networks, 2016 (blog post).