

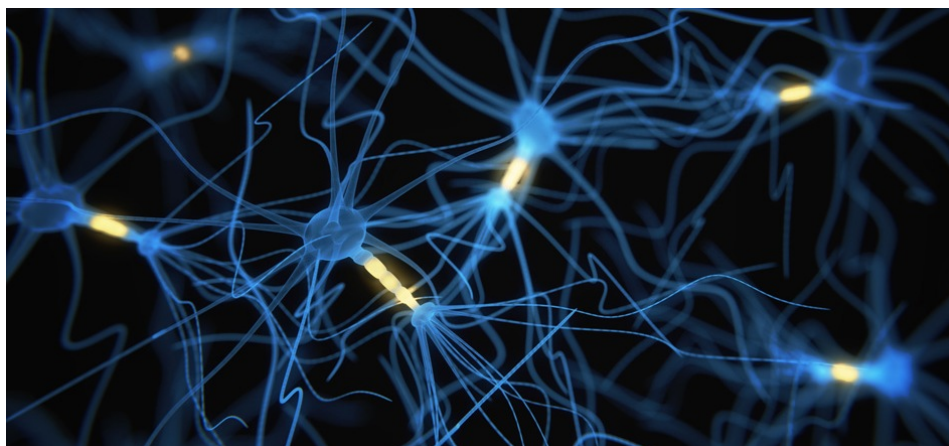
Co bude v dnešní přednášce

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

42 Úvod

Neuronové sítě

- Inspirace strukturou vzájemně projených neuronů v biologických systémech.

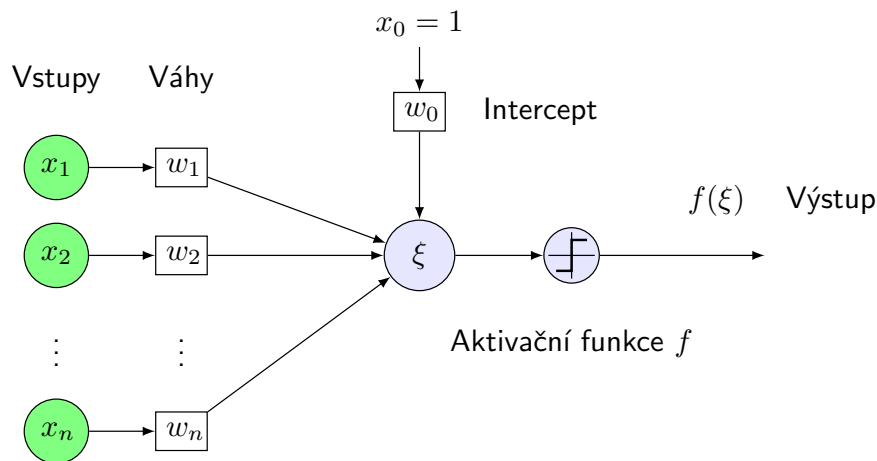


- Neuron, jakožto základní stavební kámen, je jednoduchá výpočetní jednotka.
 - Přijímá několik reálných vstupů.
 - Produkuje jeden reálný výstup.
- Prvotní model neuronu představili již v roce 1943 McCulloch a Pitts.

Perceptron (1/3)

Nejjednodušším modelem neuronové sítě určeným ke klasifikaci¹ je tzv. *jednovrstvý perceptron* (angl. *single-layer perceptron*), který se skládá z jediného **umělého neuronu** (angl. **artificial neuron**):

¹F. Rosenblatt 1957



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

Perceptron (2/3)

- Vnitřní potenciál tedy spočteme vztahem

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

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

- Výstup perceptronu je pak určen jako

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

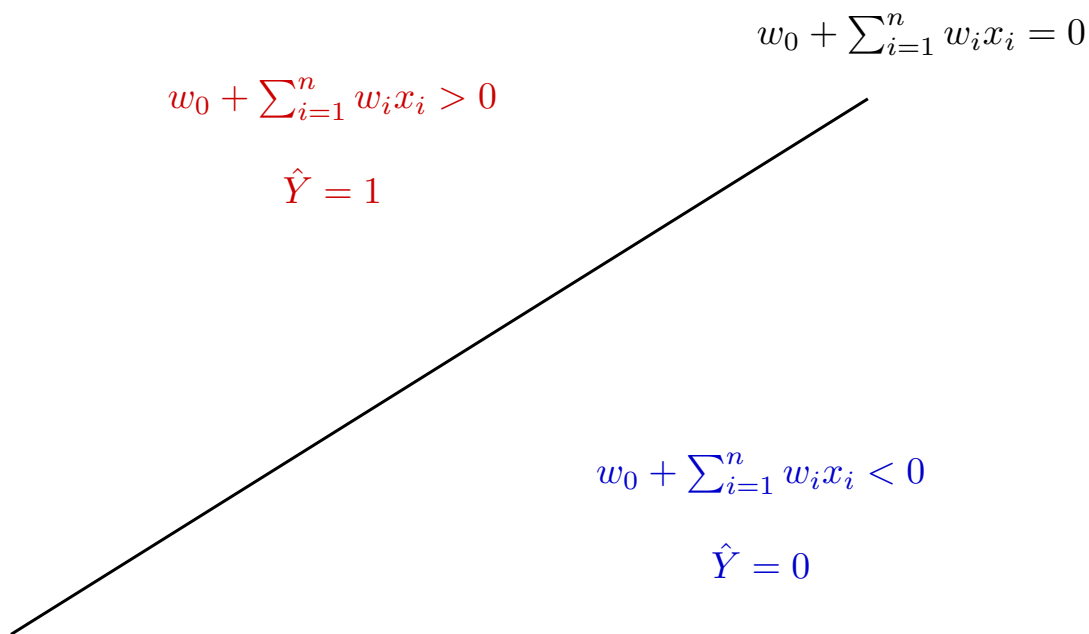
kde funkce f je tzv. *aktivační funkce* (angl. *activation function*), která je v případě perceptronu dána jako skoková funkce

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

- 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 $(x_1, \dots, x_n)^T$ 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, \dots, w_n pak určují nadrovinu, která tyto dva poloprostory separuje.

Dopředný a zpětný chod

- Pro zadané váhy w_0, w_1, \dots, w_n můžeme na základě hodnot vstupu \mathbf{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:

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

kde $\hat{Y} = f(\mathbf{w}^T \mathbf{x} + w_0)$ je predikce v bodě \mathbf{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.

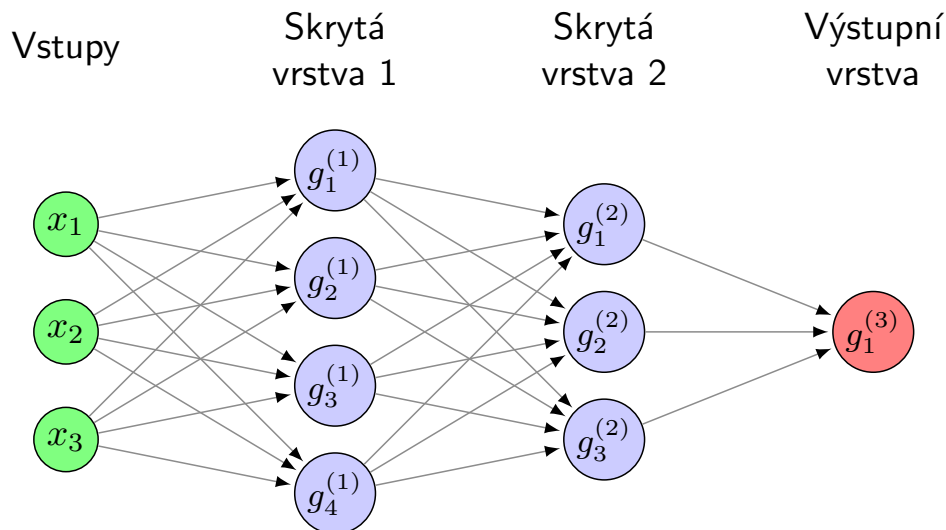
id	X_1	X_2	Y
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

- 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. $X_3 = X_1 \text{ AND } X_2$.)
- To nám napovídá, že s využitím skládání perceptronů by se daly získat mnohem lepší modely.

43 Vícevrstvá neuronová síť

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

$$\mathbf{g} = \mathbf{g}^{(l)} \circ \mathbf{g}^{(l-1)} \circ \dots \circ \mathbf{g}^{(2)} \circ \mathbf{g}^{(1)}.$$

- Např. pro $l = 3$ tak máme

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}^{(3)}(\mathbf{g}^{(2)}(\mathbf{g}^{(1)}(\mathbf{x}))).$$

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 požadujeme, 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*

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

Aktivační funkce pro výstupní vrstvy

Ve výstupní vrstvě je naším cílem převést hodnoty napočítané v předchozích vrstvách na hodnoty použitelné k predikci. Nejtýpější 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, sigmoída*,

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

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

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

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

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

kde $\boldsymbol{\xi} = (\xi_1, \dots, \xi_c)^T$ 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 $\hat{P}(Y = i | \mathbf{X} = \mathbf{x})$.

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

44 Učení neuronových sítí

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 $\hat{p} = \hat{P}(Y = 1 | \mathbf{X} = \mathbf{x})$.

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

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

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

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**

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

vzhledem k parametrům sítě \mathbf{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 $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^n$, kde $\mathbf{g} = (g_1, \dots, g_n)^T$ tak platí

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

- 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 $\mathbf{w}_i^{(1)} = (w_{i;1}^{(1)}, w_{i;2}^{(1)})^T$, 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

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

Při kvadratické ztrátové funkci tedy **minimalizujeme**

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

Vybrané složky gradientu jsou:

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

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

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 množství 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 $\mathbf{w} = (w_1, \dots, w_m)^T$ a trénovací data $(Y_1, \mathbf{x}_1), \dots, (Y_N, \mathbf{x}_N)$.
- Inicializujeme všechny váhy \mathbf{w} jako malá náhodná čísla.
- Opakujeme dokud nejsou splněna kritéria zastavení:

– Pro trénovací množinu spočteme průměrnou chybu predikce

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

– Spočteme gradient

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

– Provedeme přepočít vah

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

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 řídky rozmístěné a především je v nich zkoumaná funkční hodnota často blízka 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

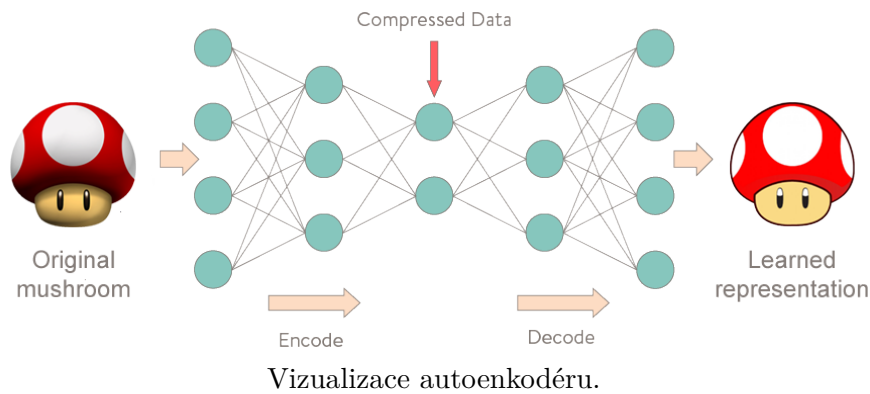
- Přidávání dalších vrstev může vést k [přeučování](#) (zvyšuje se počet parametrů).
- Často se tedy v jednotlivých vrstvách používají různé [regularizace](#). Například se k účelové funkci přidává člen penalizující velikosti vah všech neuronů v dané vrstvě.
- Další používanou formou regularizace je [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.

45 Příklady speciálních architektur

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 [redukcii 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 dekodovací části.

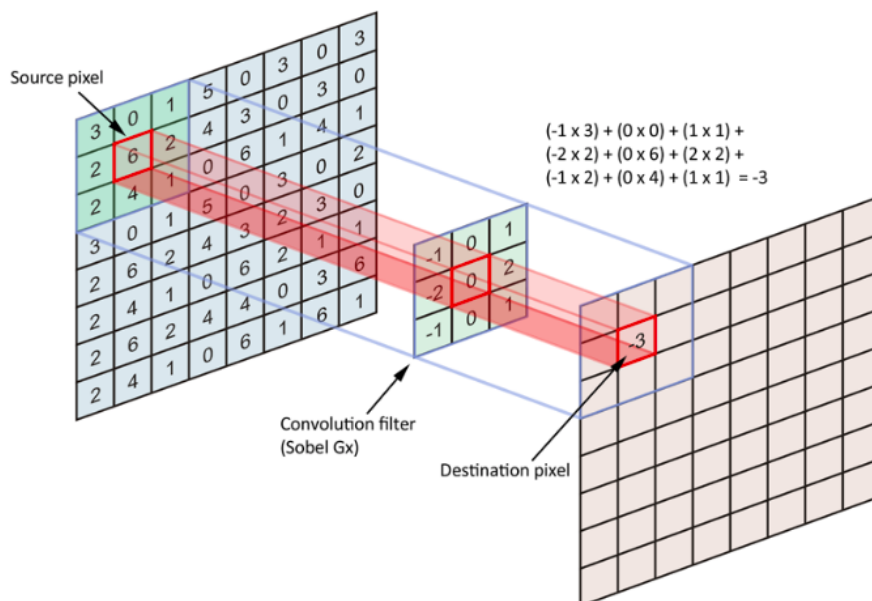
Autoenkodéry



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 sensorické rozpoznávání**: tj. v našem mozku jsou velmi abstraktní neurony (příznaky), které jsou vysoko nad sensorickým 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

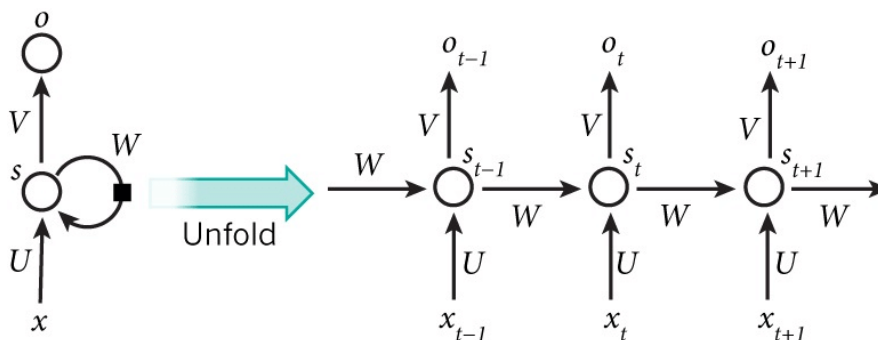


Příklad výpočtu konvoluce mezi sousedními vrstvami CNN.

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](#).

46 Literatura

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).

ChangeLog

Verze	Datum	Autor	Log
1.0	15.09.2018	KK	Výchozí verze pro rok 2018/2019.
1.0	4.12.2019	DV	Výchozí verze pro rok 2019/2020.
1.0	10.12.2019	DV	Výchozí verze pro rok 2020/2021.
1.0	9.12.2021	DV	Výchozí verze pro rok 2021/2022.