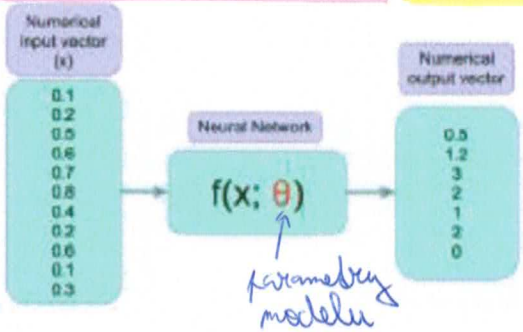
25. Neuronové sítě a jejich trénování (metoda gradientního sestupu, účelová (loss) funkce, výpočetní graf, aktivační funkce, zápis pomocí maticového násobení, ...).

(Sources: SUI, KNN přednášky a internet)

Neuronová síť obecně

Zobecněný popis neuronové sítě

Jedná se o komplexní a flexibilní funkci (i z hlediska matematické definice), která transformuje vstupní data na výstupní veličiny - **Pozor** obojí musí být floaty kvůli tomu, aby šly derivovat.



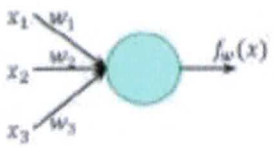
Základní vlastnosti (výhody) neuronové sítě:

- Dá se přizpůsobit pro široký rozsah různých úloh (zpracování obrazu, přirozeného jazyka, ...)
- Oproti ostatním metodám se neuronové sítě dobře škálují (více dat + větší model = lepší výsledky)
- S dostatečným množstvím trénovacích dat zvládne svou komplexitou aproximovat téměř jakoukoliv funkci (i když se jedná např. o transformaci fotografie na pravděpodobnost přítomnosti daného objektu v obraze apod.)

Příklady typů neuronových sítí (detailněji v otázce 26 a 24):

- Konvoluční sítě
- Transformery
- Logistická/lineární regrese

Základní ideou, ze které vychází koncept neuronových sítí je **perceptron**:



$$f(x; w) = w_1x_1 + w_2x_2 + w_3x_3 = \sum_i w_i x_i$$

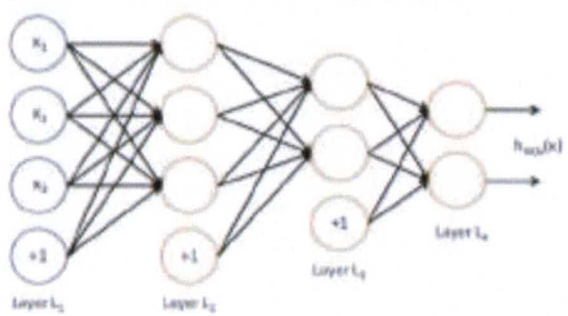
$$(x_1, x_2, x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = f_w(x)$$

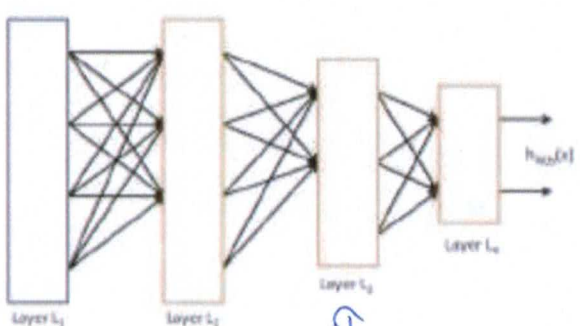
jeden neuron

x_i = vstupní data (jeden float)

w_i = parametr/váha (to, co se snažím učit/optimalizovat během trénování)

Celá neuronová síť je potom změť těchto "neuronů" uspořádaných do vrstev jdoucích za sebou. Většinou se jedná o specializované vrstvy s konkrétními názvy, ale nejobecnější koncept se nazývá MLP (Multilayer Perceptron). MLP se skládá z více vrstev FC (dense/fully connected layer), kde každý neuron vrstvy je spojen s každým neuronem předchozí vrstvy.





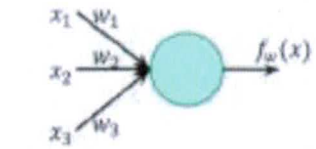
$$(x_1, x_2, x_3) \cdot \begin{pmatrix} w_1^1 & w_1^2 & w_1^3 \\ w_2^1 & w_2^2 & w_2^3 \\ w_3^1 & w_3^2 & w_3^3 \end{pmatrix} \cdots$$

matice vah jedné vrstvy

dnes hlavně plně propojené vrstvy říkáme perceptron proto že fully connected síť se nazývá multi layer perceptron

Interpretace pomocí maticového násobení

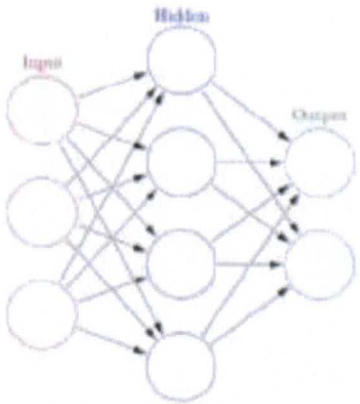
Vzorec pro výpočet výstupu jednoho neuronu jde reprezentovat jako maticové násobení:



$$f(x; w) = w_1x_1 + w_2x_2 + w_3x_3 = \sum_i w_i x_i$$

$$\rightarrow \begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2\\-1\\-1\end{bmatrix} = -1$$

Když vezmeme v potaz celou jednu vrstvu neuronů (v obrázku jde o tu s označením Hidden), jedná se o složení více matic (**každá reprezentuje jeden neuron té jediné vrstvy**). Toto složení matic je možné složit do jedné velké matice.



$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2\\-1\\-1\end{bmatrix} \begin{bmatrix}0\\2\\0\end{bmatrix} \begin{bmatrix}-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\0\end{bmatrix} = -1, 0, -1, -1$$

$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2\\-1\\-1\end{bmatrix} \begin{bmatrix}0\\2\\0\end{bmatrix} \begin{bmatrix}-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\0\end{bmatrix} = -1, 0, -1, -1$$

$$\downarrow$$

$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2 & 0 & -1 & 1\\-1 & 2 & 0 & 1\\-1 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}-1 & 0 & -1 & -1\end{bmatrix}$$

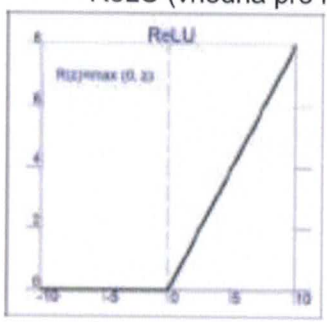
$$f(x; \mathbf{W}) = \mathbf{W}x$$

(na obrázku je násobení vyznačené chybně opačně v rovnicové formě)
Maticový zápis pro aplikaci jedné vrstvy neuronové sítě je na obrázku výše. Kombinace faktu, že princip neuronových sítí spočívá v aplikaci stejných operací nad velkým množstvím dat a že výpočet je realizovatelný maticovým násobením, vede k vysoké efektivitě při využití výpočtů na GPU.

Aktivační funkce/nelinearita – vnáší do sítě nelinearitu

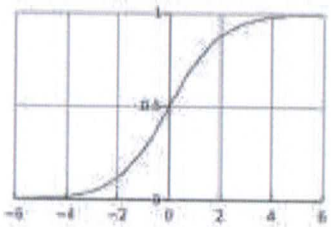
Dnešní neuronové sítě uvažují neuron (základní stavební jednotku) jako aplikace lineární funkce y = wx + b (w = váhy; b = bias). Seskupení více vrstev sítě za sebou by ale nevedlo k ničemu jinému než opět lineární funkci. **Chceme ale modelovat složitější funkce než jen lineární závislosti**, a proto se zavádí **aktivační funkce/nelinearita**. Ta zpracovává každý neuron (každý float) samostatně a aplikuje se na **výstup** každého jednotlivého **neuronu** v celé vrstvě. **Pro každý typ vrstvy/sítě je vhodný jiný typ aktivační funkce.** Mezi nejznámější aktivační funkce patří:

- ReLU (vhodná pro hluboké sítě, konvoluční a rekurentní sítě)



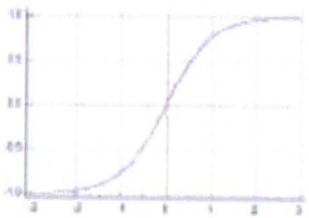
$$f(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{pro } x > 0, \\ 0 & \text{jinak,} \end{cases}$$

- Sigmoid (vhodná pro mělké sítě [vanishing gradient problem] a pro úlohy binární klasifikace)



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

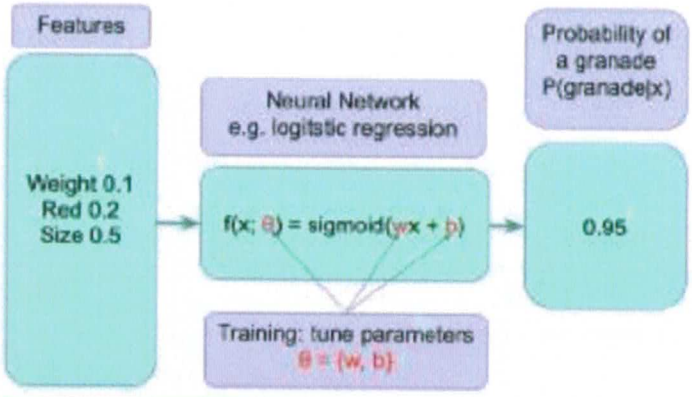
- Tanh (vhodná pro rekurentní sítě a případy, kdy je vhodné pracovat se zápornými hodnotami)



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

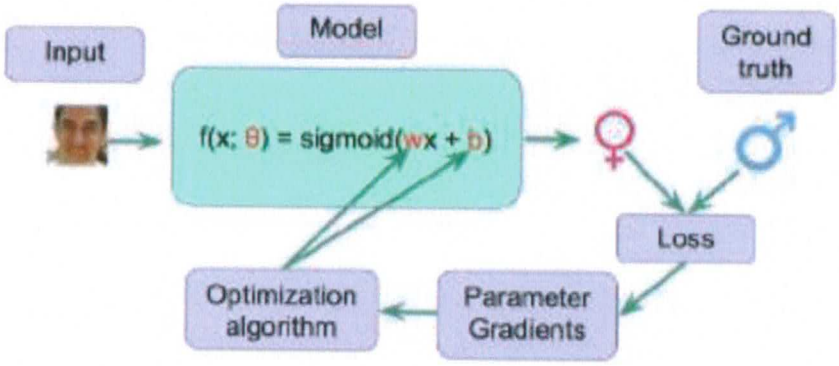
Trénování neuronové sítě

Jedná se o optimalizační postup (konkrétně různé varianty **gradient descent** kvůli dobré paralelizovatelnosti), který optimalizuje parametry neuronové sítě (typicky váhy u jednotlivých neuronů a bias hodnoty) na základě trénovacích dat. Cílem je podle trénovacích dat co nejlépe odhadnout parametry neuronové sítě.



Obecný trénovací loop se skládá z:

- Dopředného průchodu (aplikace výpočtu neuronové sítě na vstup)
- Spočtením loss na základě anotací správných hodnot
- Spočítání gradientů (zpětný průchod)
- Úprava parametrů



Co k natrénování neuronové sítě potřebuji definovat:

- Model neuronové sítě
- Loss funkci
- Optimalizátor (optimalizační algoritmus)
- Data

Jaké druhy trénování jsou (obecně, nejen neuronové sítě):

- **Učení s učitelem** (u neuronových sítí se bavíme většinou o tomhle - učitel jsou poskytnutá data a vzorové výstupy, tedy anotovaná data)

- Učení bez učitele (shlukování, detekce anomálií, apod.)
- Posilované učení

Účelová/loss funkce (nebo také chybová funkce)

Jedná se o funkci, která měří rozdíl výstupů neuronové sítě od požadované podoby výstupu. Její podoba je úzce spjata s úlohou, kterou chceme trénovat. Její design je nutné volit pečlivě, jako u každého optimalizačního algoritmu se i neuronové sítě budou snažit "najít zkratky". Loss funkce slouží pro účely trénování a má za úkol hodnotit kvalitu výstupu způsobem, který:

1) Dostatečně reprezentuje danou úlohu a hodnotí, co od modelu očekáváme
2) Je derivovatelný

Loss funkce neslouží k ohodnocení kvality natrénované neuronové sítě. Pouze ho má navést správným směrem.

Matematická definice učení/optimalizace:

$$J(D,\theta) = \sum_{D=\{(x_i,y_i),\dots\}} loss\left(f\left(x_i,\theta\right),y_i\right)$$

$$\theta^* = arg\min_{\theta} J(\mathbf{D},\theta)$$

Hledáme tedy parametry θ takové, že ∀p∈P: loss(f(x, θ), y) ≤ loss(f(x, p), y), kde P je množina všech možných parametrů.

Příklady loss funkcí můžou být:

- Cross entropie (používá se pro binární klasifikaci)

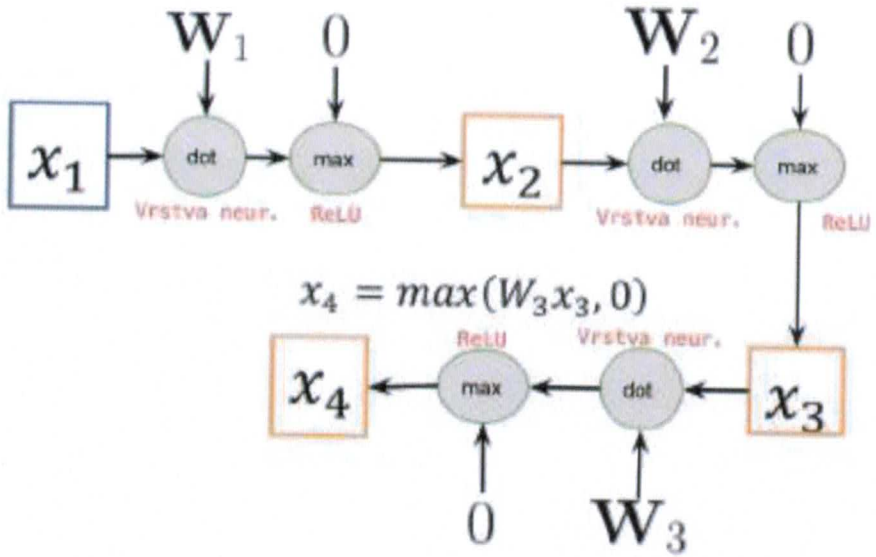
$$loss(x,y) = \begin{cases} log(f(x;\theta)) & \textbf{if} \quad y = male \\ log(1-f(x;\theta)) & \textbf{if} \quad y = female \end{cases}$$

- Suma čtverců (Mean Squared Error)

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Výpočetní graf a derivace

Dopředný průchod (výpočet celé neuronové sítě včetně aktivací a dodatečných výpočtů) jde reprezentovat výpočetním grafem. Tento graf zobrazuje tok dat systémem. Jediná podmínka tohoto grafu je, že **musí být acyklický**. Tento graf se na přednáškách vždy používal v kontextu počítání gradientů/derivací. Gradienty jsou následně využity pro účely optimalizace s využitím některé z variant algoritmu **gradientního sestupu**.

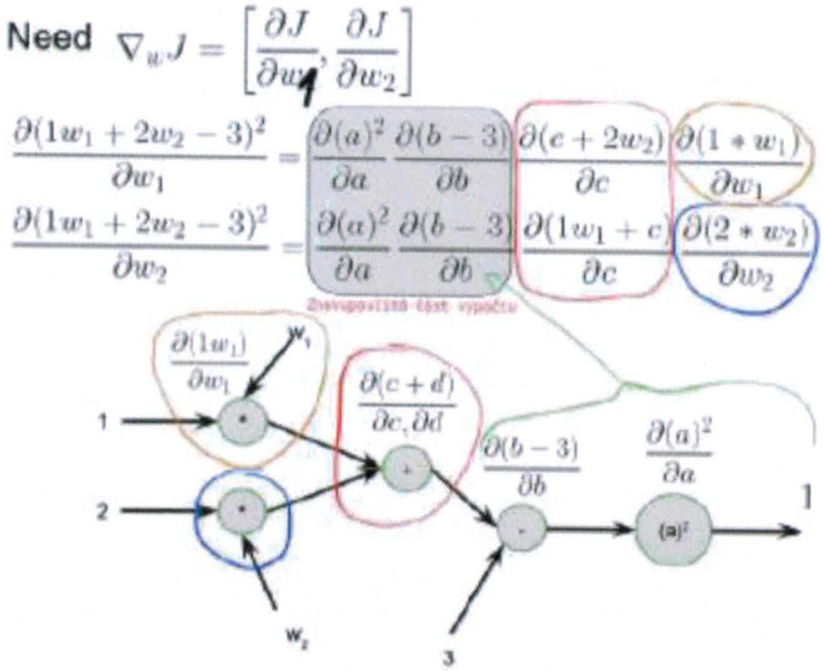


Principu derivace a upravování parametrů/vah v neuronových sítí se říká zpětný průchod. Ve zpětném průchodu jde už ale pouze o maticové násobení dle výstupů neuronové sítě (derivace funkcí jsou předem spočtené). Jde o maticové násobení díky faktu, že výpočty neuronové sítě lze reprezentovat jako tento acyklický výpočetní graf. Jedná se tedy o vnořené funkce f1(f2(f3(x))), na které lze při derivování aplikovat chain rule. Tudíž ono násobení se bere odtud:

$$F'(x) = f'(g(x)) \cdot g'(x)$$

- V případě trénování je součástí zpětné propagace i **derivace loss funkce**, bez toho bychom síť nic nenaučili.
- Nesmíme zapomenout, že aplikace jednotlivých vrstev jsou maticová násobení, tudíž i jejich derivace bude zahrnovat matice.
- Při zpětném průchodu se počítají derivace pro každý parametr každé vrstvy sítě.
- Díky vlastnostem výpočetního grafu je ale možné výpočty z posledních vrstev sítě recyklovat při postupném zpětném průchodu.

Příklad derivace na základě výpočetního grafu:



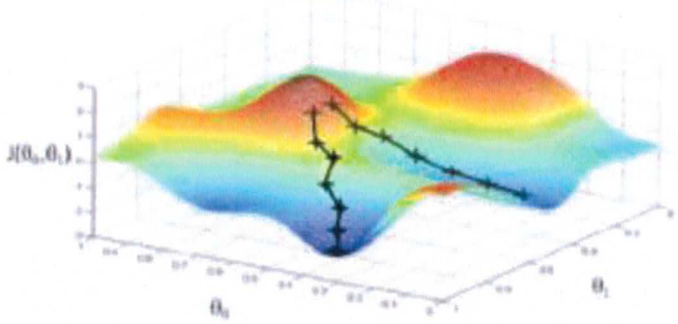
V každém kroku se provede substituce za část grafu "kam chci jít" při zpětném průchodu a vzhledem k té substituci derivuju (viz. substituce c). Zbytek je otázka chain rule.
Takto spočítaná a dosazená derivace/gradient následně udává směr kroku algoritmu **Gradient descent** – parciální derivace vzhledem k w_1 a w_2 protože chci znát směr kroku v prostoru parametrů neuronové sítě.

Gradient descent

K učení neuronové sítě (hledání vhodných parametrů) se využívají různé varianty algoritmu gradientního sestupu. Jde o algoritmy, kde se v každém kroku provádí posun v prostoru parametrů na základě spočítaného gradientu (derivace). Jak je ukázáno taky na vzorci ("běžný" gradient descent), jde vždy o krok z původního bodu v prostoru parametrů o nějakou vzdálenost. Tento algoritmus **se může zaseknout** v lokálním minimu. Důvod volby zrovna tohoto algoritmu je jednoduchá implementace a paralelizovatelnost. Metody založené na GD počítají s tím, že se budou počítat nad celou trénovací sadou dat, což je nereálné. Využívá se proto nejvhodnější možná aproximace - trénování po dávkách (**mini-batch + stochastic mini-batch gradient descent**).

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(D,\theta)}{\partial \theta_j}$$

$$J(D,\theta) = \sum_{D=\{(x_i,y_i),\ldots\}} loss\left(f\left(x_i,\theta\right),y_i\right)$$



Délka kroku se určuje konstantou (learning rate) a závisí na volbě vývojáře. S velkou hodnotou konstanty učení může dojít k "přeskakování" hledaného optima. S malými kroky může naopak docházet k zaseknutí v lokálním optimu. Různé další implementace GD se pokouší tyto problémy řešit. Příklady takovýchto adaptivních implementací optimalizačního algoritmu:

- Adam
- AdamW
- SGD

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele no.body.the.sad.slider.boy.