Neuronové sítě 2 - Úvod do neuronových sítí

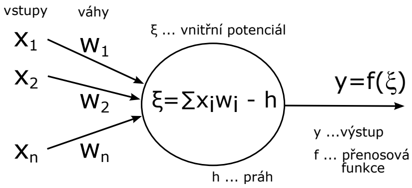
18NES2 - 3. hodina, ZS 2024/25

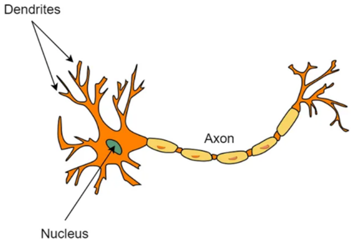
Zuzana Petříčková

16. října 2024

Neuronové sítě 2 - Úvod do neuronových sítí

Matematický model neuronu





- parametry neuronu:
  - vektor vah $\vec{w} = (w_1, \ldots, w_n) \in R^n$,
  - práh h (nebo bias $b = -h$
  - přenosová (aktivační) funkce $f : R \rightarrow R$
- neuron pro vstup $\vec{x} \in R^n$ spočte výstup $y \in R$ jako hodnotu přenosové funkce $f_{\vec{w},h}(\vec{x})$
  - vnitřní potenciál: $\xi = \sum_{i=1}^{n} w_i.x_i - h = \vec{x}\vec{w} - h$
  - výstup: $y = f(\xi)$

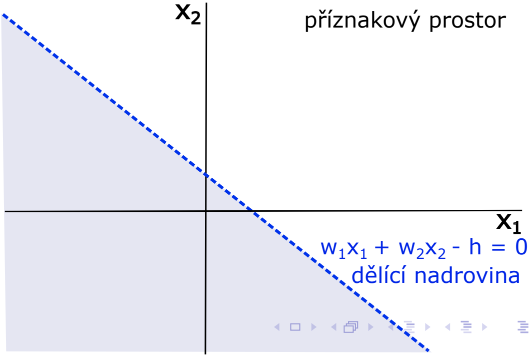
Matematický model neuronu

Geometrická interpretace

- vstupy neuronu si představme jako body v n-rozměrném Euklidovském prostoru (vstupní, příznakový prostor)
- položme vnitřní potenciál neuronu $\xi = 0$ a získáme rovnici dělící nadroviny

$$\xi = w_1 x_1 + w_2 x_2 - h = 0$$

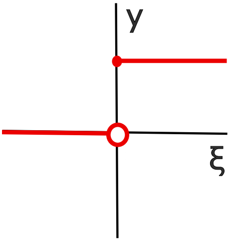
$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{h}{w_2}$$



Matematický model neuronu

- historický model perceptronu používal **skokovou přenosovou funkci:**

- $f(\xi) = 1$ pro $\sum_{i=1}^{n} w_i.x_i \geq h$, tj.
  $\xi = \sum_{i=1}^{n} w_i.x_i - h \geq 0$
  ... neuron je **aktivní**
- $f(\xi) = 0$ pro $\sum_{i=1}^{n} w_i.x_i < h$, tj.
  $\xi = \sum_{i=1}^{n} w_i.x_i - h < 0$
  ... neuron je **pasivní**

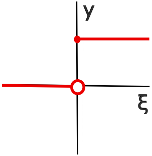


$\rightarrow$ perceptron může sloužit jako **lineární klasifikátor:** klasifikuje vzory do dvou tříd

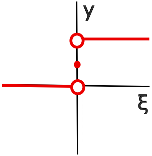
Matematický model neuronu

Varianty skokové přenosové funkce pro binární perceptron:

$$f(\xi) = \begin{cases} 1 & \text{pro } \xi \geq 0 \quad \text{... neuron je aktivní} \\ 0 & \text{pro } \xi < 0 \quad \text{... neuron je pasivní} \end{cases}$$



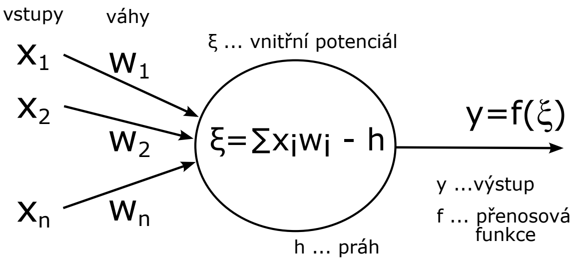
$$f(\xi) = \begin{cases} 1 & \text{pro } \xi > 0 \quad \text{.. neuron je aktivní} \\ 0.5 & \text{pro } \xi = 0 \quad \text{... neuron je tichý} \\ 0 & \text{pro } \xi < 0 \quad \text{... neuron je pasivní} \end{cases}$$



$\rightarrow$ funkce **signum**

obdobně pro bipolární perceptron (výstupy -1, 0, 1)

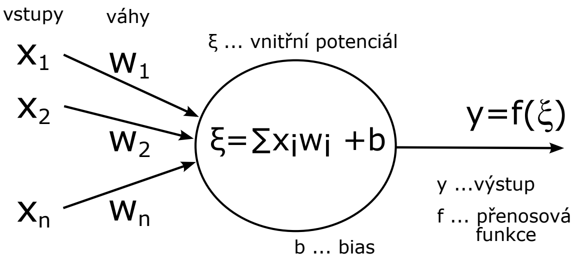
Matematický model neuronu



Klasická definice: práh h

- vnitřní potenciál: $\xi = \sum_{i=1}^{n} w_i . x_i - h = \vec{w}\vec{x}^T - h$
- výstup: $y = f(\xi)$

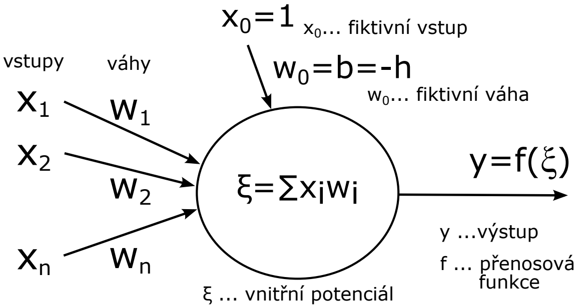
Matematický model neuronu



Modernější definice: práh → bias

- vnitřní potenciál: $\xi = \sum_{i=1}^{n} w_i.x_i + b = \vec{w}\vec{x}^T + b$
- výstup: $y = f(\xi)$

Matematický model neuronu



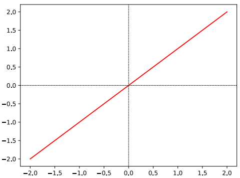
Alternativní definice: zavedení fiktivního vstupu

- rozšířený příznakový prostor ... $\vec{x} = (x_0 = 1, x_1, ..., x_n)$
- rozšířený prostor vah ... $\vec{w} = (w_0 = b = -h, w_1, ..., w_n)$
- vnitřní potenciál: $\xi = \sum_{i=0}^{n} w_i.x_i = \vec{w}\vec{x}^T$
- výstup: $y = f(\xi)$

Matematický model neuronu

A co jiné přenosové (aktivační) funkce?

- $f(\xi) = \xi$ ... *identita* ... **lineární neuron**
  - první zkoumaná spojitá přenosová funkce ($\rightarrow$ lze učit gradientní metodou)
- výstup: $y = \xi = \sum_{i=0}^{n} w_i.x_i = \vec{x}\vec{w}$ $\rightarrow$ při učení hledáme $\vec{w}$, aby platilo: $\vec{d} = \vec{y}$, tj. $\vec{d} = X\vec{w}$
  - jedná se o úlohu **lineární regrese**

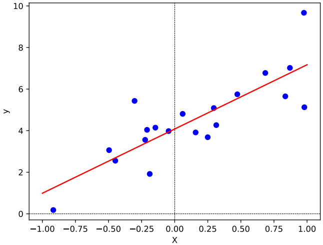


Využití:

- velmi vhodná do výstupní vrstvy pro regresní úlohy
- nehodí se příliš pro klasifikační úlohy a do skrytých vrstev

Lineární neuron - geometrická interpretace

- výstup neuronu: $y = w_1 x + w_0$
- (x_k, d_k) jsou body v rovině
- prokládáme body přímkou:

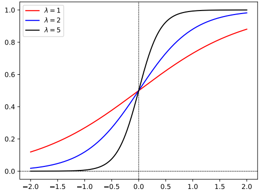


$\rightarrow$ obecně: prokládáme body nadrovinou (předpokládáme, že mezi vstupními veličinami a výstupní je lineární závislost)

Přenosové funkce pro hluboké učení

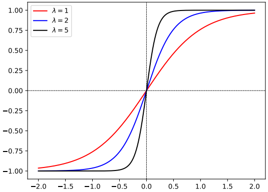
Sigmoidální

- $f(\xi) = \frac{1}{1+e^{-\lambda\xi}}$ ... logsig
- pro binární model
- pro jednovrstvý model se jedná o úlohu **logistické regrese**



Hyperbolický tangens

- $f(\xi) = \frac{1-e^{-2\lambda\xi}}{1+e^{-2\lambda\xi}}$ ... tanh
- pro bipolární model

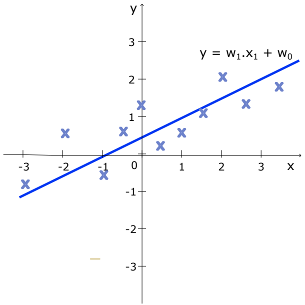


→ „rozvolněná" skoková přenosová funkce

Přenosové funkce pro hluboké učení

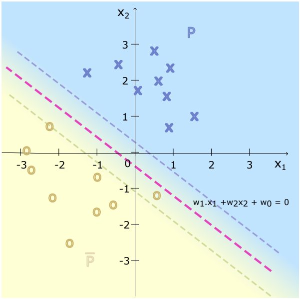
- lineární

Úloha lineární regrese



- sigmoida, hyperbolický tangens

Úloha lineární klasifikace

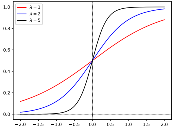


(úloha logistické regrese)

Přenosové funkce pro hluboké učení

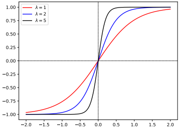
Sigmoidální

- $f(\xi) = \frac{1}{1+e^{-\lambda\xi}}$ ... logsig



Hyperbolický tangens

- $f(\xi) = \frac{1-e^{-2\lambda\xi}}{1+e^{-2\lambda\xi}}$ ... tanh



Využití

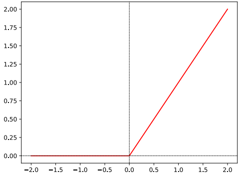
- Hyperbolický tangens: ve skrytých vrstvách hlubokých neuronových sítí, rekurentní neuronové sítě
- Sigmoidální funkce: ve výstupní vrstvě pro úlohu binární klasifikace

Přenosové funkce pro hluboké učení

Pozitivně lineární (ReLU, rectified linear unit)

- $f(\xi) = max(0, \xi) = \begin{cases} x, & \text{pro } \xi > 0 \\ 0, & \text{pro } \xi \leq 0 \end{cases}$

  ... *poslin*



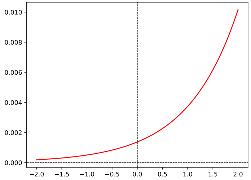
Využití

- ve skrytých vrstvách hlubokých neuronových sítí (efektivní, oblíbená)
    - nesaturuje (řeší problém mizejících gradientů), ale problém mrtvých ReLU (stále pasivní neurony)

Přenosové funkce pro hluboké učení

Softmax

- speciální přenosová funkce pro klasifikaci do více tříd
- zobecnění funkce argmax, převádí číselné hodnoty na pravděpodobnosti
- $f : R^n -> R^n,\ f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$



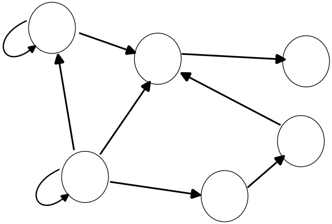
→ hodí se do výstupní vrstvy pro klasifikační úlohy

Neuronová síť

- Skládá se z neuronů, které jsou navzájem pospojovány tzv. hranami
- Výstup jednoho neuronu může být vstupem jednoho nebo více dalších neuronů

Architektura (topologie) neuronové sítě

- Orientovaný graf, neurony představují uzly, synaptické vazby představují hrany



Architektura (topologie) neuronové sítě

Výstupní neurony

- jejich výstupy tvoří dohromady **výstup (odezvu)** neuronové sítě
- **typicky:** nevedou z nich žádné hrany do jiných neuronů

Vstupní neurony

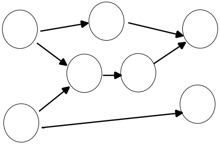
- mají na vstupu vstupní vzory
- **typicky:** nevedou do nich žádné hrany z jiných neuronů

Výstup (odezva) neuronové sítě

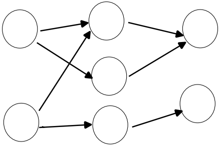
- Výstupy (aktivity) výstupních neuronů

Architektura (topologie) neuronové sítě

- cyklická, rekurentní
- acyklická, dopředná - ,,všechny hrany jdou stejným směrem"
(tj. graf lze topologicky uspořádat)



- hierarchická (vrstevnatá, sekvenční) - dělí se na vrstvy, propojeny jsou jen neurony ze dvou po sobě jdoucích vrstev



Architektura (topologie) neuronové sítě

Vrstevnatá neuronová síť (multi-layer neural network, MLP, 80. léta):

- hierarchická -**sekvenční**- architektura, neurony jsou uspořádány do vrstev
- **dense layers (plně propojené vrstvy)**: všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

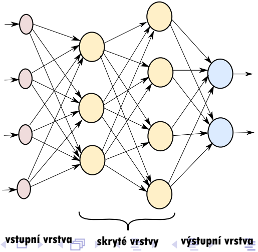
Speciální vstupní vrstva:

- odpovídá vstupům neuronové sítě

Výstupní vrstva

- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů

Zbylé vrstvy jsou **skryté**.

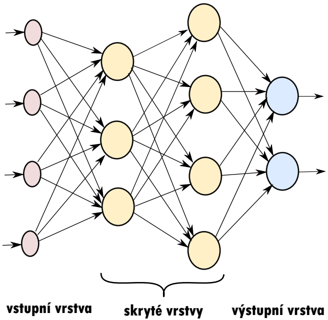


Architektura (topologie) neuronové sítě

Výpočet odezvy u vrstevnaté neuronové sítě (MLP):

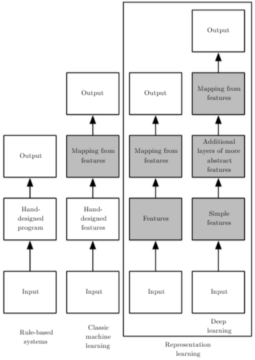
- pro daný vstupní vektor $\vec{x}$ délky n model spočítá výstupní vektor $\vec{y}$ délky m
- výpočet výstupu dopředným průchodem (forward pass)

1. výstup neuronů ve vstupní vrstvě: $y_i = x_i$
2. Postupujeme ve směru od první skryté vrstvy k výstupní a pro každý neuron j spočteme jeho výstup y_j: $y_j = f(\xi_j) = f(\sum_i w_{ij} y_i + b_i)$ (i je index přes neurony ve vrstvě předcházející neuronu j)
3. Výstup sítě $\vec{y} = (y_1, \ldots, y_m)$ tvoří výstupy neuronů ve výstupní vrstvě



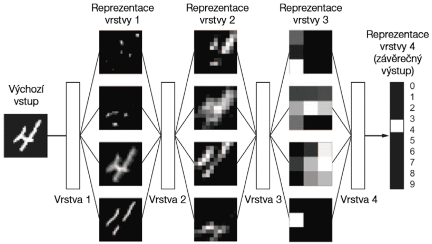
Architektura (topologie) neuronové sítě

- **mělká (shalllow)** - model s jednou skrytou vrstvou
- **hluboká (deep)** - model s více (nebo i mnoha) skrytými vrstvami



I. Goodfellow and Y. Bengio and Aaron Courville: Deep Learning, 2016, Figure 1.5

Konvoluční neuronová síť:

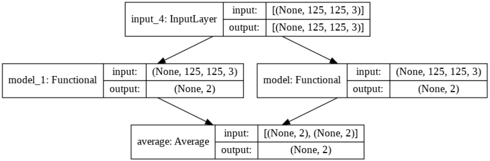


F. Chollet: Deep learning v jazyku Python, obr. 1.6

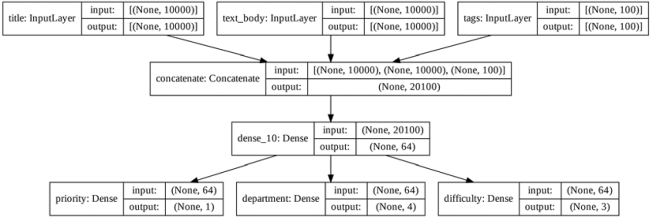
Architektura (topologie) neuronové sítě

- u moderních hlubokých modelů si často se sekvenční (vrstevnatou) architekturou nevystačíme, např.:

Ensemble model

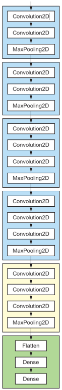


Model s více vstupy a výstupy

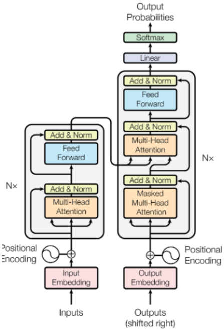


Architektura (topologie) neuronové sítě

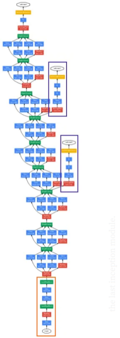
- moderní hluboké sítě mají často složité topologie:



The VGG-16 CNN model (F. Chollet: Deep learning v jazyku Python, obr. 8.15)



The original transformer architecture (https://arxiv.org/abs/1706.03762)



Inception V1 (https://arxiv.org/pdf/1409.4842v1)

Učení neuronové sítě (supervised learning - učení s učitelem)

Data, na základě kterých se model učí

- trénovací množina T
  - množina N trénovacích vzorů
    $T = (X, D) = \{(x_1, d_1), ..., (x_N, d_N)\}$
  - X ... vstupní data (tenzor), D ... požadovaný výstup (tenzor)
- trénovací vzor (training pattern) ... (x_i, d_i),
  - x_i ... vstupní vzor (input pattern)
  - d_i ... požadovaný (očekávaný) výstup (target)
- Příklady vstupních datových tenzorů:
  - vektorová data - 2D tenzory tvaru (vzory, příznaky=features)
  - časové řady a sekvenční data - 3D tenzory tvaru (vzory, čas. úseky, příznaky)
  - obrázky - 4D tenzory tvaru (vzory, výška, šířka, channels)
  - video - 5D tenzory tvaru (vzory, snímky, výška, šířka, channels)

Učení neuronové sítě (supervised learning - učení s učitelem)

Počet výstupních příznaků závisí na úloze, kterou řešíme:

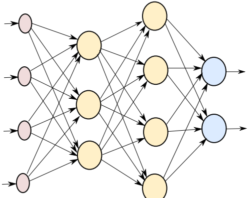
- **Regrese:**
  výstupní tenzor tvaru (vzory, 1) nebo (vzory, výstupní příznaky)
  např. predikce hodnot (např. ceny, teploty)
- **Klasifikace :**
  výstupní tenzor tvaru (vzory, počet tříd)
  např. binární klasifikace $\rightarrow$ výstupní tvar (vzory, 1)
  nebo klasifikace do více tříd $\rightarrow$ tvar (vzory, počet tříd)
- **Sekvenční data:**
  výstupní tenzor tvaru (vzory, časové kroky, výstupní příznaky)
  např. překlad vět $\rightarrow$ tvar (vzory, délka výstupní sekvence, počet slov ve slovníku)
- obrázky, video, ...

Učení vrstevnaté neuronové sítě (MLP model)

- nyní se zaměřme na klasický model vrstevnaté neuronové sítě s n vstupy a m výstupními neurony:

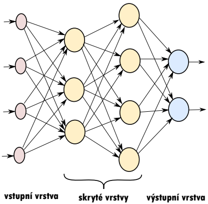
Data, na základě kterých se model učí

- trénovací množina $T = (X, D)$
  - X ... vstupní vzory: 2D tenzor tvaru (N, n), N je počet trénovacích vzorů, n je počet vstupních příznaků
  - D ... požadovaný výstup: 2D tenzor tvaru (N, m), m je počet výstupních příznaků
- trénovací vzor (training pattern) ... $(\vec{x}_i, \vec{d}_i)$,
  - $\vec{x}_i$ ... vektor délky n
  - $\vec{d}_i$ ... vektor délky m



Učení vrstevnaté neuronové sítě (MLP model)

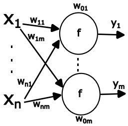
- Máme vrstevnatou neuronovou síť s n vstupními a m výstupními neurony, neurony mají spojitou, diferencovatelnou přenosovou funkci
- Model transformuje vstupní vzory (vektory) x_i na výstupní vzory (vektory) y_i
- **Cíl učení:** Nastavit váhy (a biasy) všech neuronů v síti tak, aby byl skutečný výstup sítě $\vec{y_i}$ stejný jako požadovaný ($\vec{d_i}$).



Učení neuronové sítě

Začněme s jednou plně propojenou vrstvou:

- vrstva předpokládá na vstupu 2D-tenzor tvaru (vzory, číselné příznaky)
- vrstvu MLP můžeme reprezentovat maticí vah **W** a vektorem biasů **b**
- vrstva transformuje vstupní tenzor X na výstupní tenzor Y:
  $Y = f(XW + \vec{b})$
  X je tvaru (vzory, vstupní příznaky), Y je tvaru (vzory, výstupní příznaky), W je tvaru (vstupní příznaky, výstupní příznaky)
- **Cíl učení**: Nastavit W a b, tak, aby byl skutečný výstup modelu Y stejný jako požadovaný D.



Učení neuronové sítě

Základní princip (zjednodušeně)

1. náhodně inicializuj parametry modelu (váhy a biasy, tj. W a b)
2. opakuj trénovací cyklus:
   - připrav dávku (batch) trénovacích vzorů X a odpovídajících požadovaných výstupů D
   - spočti skutečný výstup (predikci) modelu ... pro jednu vrstvu by to bylo $Y = f(XW + \vec{b})$
   - spočti chybu modelu (jak moc se liší Y a D)
   - aktualizuj W a b tak, aby se chyba modelu o něco zmenšila

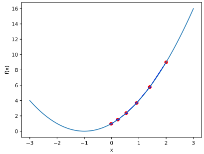
→ **gradientní metoda (gradient descent)**

- podmínkou je spojitá přenosová funkce

Gradientní metoda (metoda největšího spádu, gradient descent)

Úloha (zjednodušeně):

- máme funkci $f(\vec{x}) : R^n \to R$
- hledáme $\vec{x}$, pro které je $f(\vec{x})$ minimální



$\to$ **řešení gradientní metodou:**

1. začneme v nějakém počátečním bodě $\vec{x}(0)$
2. spočteme gradient $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
   gradient vyjadřuje směr a velikost největšího růstu funkce v daném bodě
3. v cyklu se posunujeme ,,o kousek'' proti směru gradientu:
   $\vec{x}(t+1) = \vec{x}(t) + \alpha \nabla f(\vec{x})$
   α je malé kladné číslo (délka kroku, parametr učení)
   pro jeden parametr: $x_i(t+1) = x_i(t) - \alpha \frac{\partial f}{\partial x_i}$

Gradientní metoda (metoda největšího spádu, gradient descent)

Problémy:

- pro malé α je učení pomalé
- pro velké α kmitá (přeskakuje řešení)
- nemusí najít globální minimum (např. uvízne v lokálním)

Jak tedy nastavit parametr učení? $\rightarrow$ různé heuristiky:

- parametr učení postupně klesá, např. dle vzorce $\alpha_j = \frac{\alpha_0}{1+j}$ (Robins-Moore, 1951)
- využití momentů,...

Gradientní metoda (metoda největšího spádu, gradient descent)

Jakou chybovou funkci budeme minimalizovat? (nejprve pro 1 neuron)

- chceme aby se skutečný výstup neuronu y_p se co nejméně lišil od požadovaného d_p
- první pokus $\frac{1}{2}\sum_{p=1}^{N}|d_p - y_p|$
- lépe: $\frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2$ ... součet čtverců
  $\rightarrow$ **metoda nejmenších čtverců**

Střední kvadratická chyba (MSE):

- pro 1 neuron: $E(\vec{w}) = \frac{1}{2N}\sum_{p=1}^{N}(d_p - y_p)^2$
- pro m výstupních neuronů:
  $E(\vec{w}) = \frac{1}{2Nm}\sum_{i=1}^{m}\sum_{p=1}^{N}(d_{pi} - y_{pi})^2$

lze použít i jinou chybovou funkci (např. **cross entropy**)

Gradientní metoda (metoda největšího spádu, gradient descent)

Obecné schéma algoritmu

1. Inicializuj váhy a biasy malými náhodnými reálnými hodnotami
   Inicializuj parametr učení α_0 ... $1 > \alpha_0 > 0$
2. Předlož další dávku trénovacích vzorů (X_t, D_t) a spočti pro ně skutečnou odezvu modelu Y_t a chybe E_t
3. Adaptuj všechny váhy a biasy (indexované pomocí i):

$$w_i(t+1) = w_i(t) - \alpha_t \frac{\partial E_t}{\partial w_i}$$

$$b_i(t+1) = b_i(t) - \alpha_t \frac{\partial E_t}{\partial b_i}$$

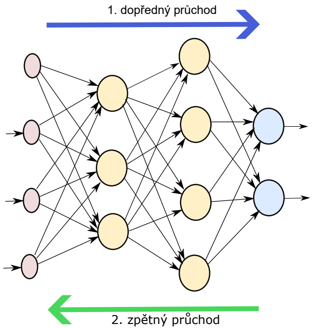
4. Případně aktualizuj parametr učení : $\alpha_t \rightarrow \alpha_{t+1}$
5. Pokud není konec, přejdi ke kroku 2.

Algoritmus zpětného šíření (Backpropagation)

(Werbos, Rumelhart, 1974-1986)

Základní princip

1. Spočteme skutečnou odezvu sítě pro daný trénovací vzor.
   - od vstupní vrstvy směrem k výstupní
2. Porovnáme skutečnou a požadovanou odezvu sítě.
3. Adaptujeme váhy a prahy:
   - proti směru gradientu chybové funkce
   - od výstupní vrstvy směrem ke vstupní



Algoritmus zpětného šíření - diskuse učící strategie

Jak předkládat trénovací vzory?

1. **iterativně po epochách (online GD)**: během jedné epochy se každý vzor předloží právě jednou, v rámci každé epochy vzory náhodně uspořádáme
   - maximální počet epoch .... kolikrát se předloží celá trénovací množina
2. **dávkově po epochách (batch GD)**:
   - celá trénovací množina se předloží najednou a váhy se adaptují najednou pro celou trénovací množinu
3. **dávkově po mini-batchích (SGD, stochastic gradient descent)**
   - v každé epoše se trénovací množina náhodně rozdělí na malé podmnožiny vzorů (mini-batch) a ty se iterativně předloží (v rámci mini-batche dávkově)

Algoritmus zpětného šíření - diskuse učící strategie

- **Online GD**
  - rychlé učení, ale poměrně nestabilní (algoritmus snižuje chybu pro aktuální vzor → chyba se může z zvýšit pro ostatní vzory)
  - vyšší citlivost na odlehlé vzory a na volbu hyperparametrů, náhodnost (ale možnost úniku z lokálních minim)
- **Batch GD**
  - stabilnější, efektivní pro malá data
  - výpočetně a paměťově náročný pro velká data
  - vyšší citlivost na šum v datech
- **Mini-batch SGD**
  - spojuje výhody obou předchozích strategií
  - používá se pro velké datové sady a hluboké sítě

Algoritmus zpětného šíření - diskuse učící strategie

Kdy ukončit učení?

1. předem daný maximální počet epoch
2. jakmile přestane klesat chyba na validační množině dat: **early stopping**
3. jakmile je přírůstek vah Δw moc malý ... $|\Delta w| < \delta_{min}$
4. jakmile je průměrná chyba dostatečně malá ... $E < E_{min}$
5. časový limit

Vrstevnatá neuronová síť učená gradientní metodou - analýza modelu

Výhody:

- Jednoduchý univerzální model s poměrně dobrými aproximačními a generalizačními schopnostmi
- Univerzální aproximátor – zvládá aproximaci jakékoliv spojité funkce (pro nelineární přenos. fce stačí jedna vrstva). Ale problém učení je NP-úplný.
- Vhodný pro úlohy klasifikace i regrese.
- Schopnost zachytit komplexní nelineární vztahy.
- Využívá backpropagation pro efektivní učení gradientní metodou.
- Dobře zobecňuje

Vrstevnatá neuronová síť učená gradientní metodou - analýza modelu

Nevýhody:

- Je nutné správně nastavit a vyladit hyperparametry
- Omezení na tvar vstupních a výstupních dat
- Pomalá konvergence.
- Lokální metoda učení – může nalézt suboptimální řešení.
- Náchylnost k přeučení – pokud není dobře nastavená regularizace nebo early stopping.
- Nemá zabudované mechanismy pro zohlednění prostorové struktury dat
- Citlivost na inicializaci, trénovací data a hyperparametry.