Pokročilé metódy analýzy dát 2

úvod do hlbokého učenia

Peter Bednár

Neurónové siete

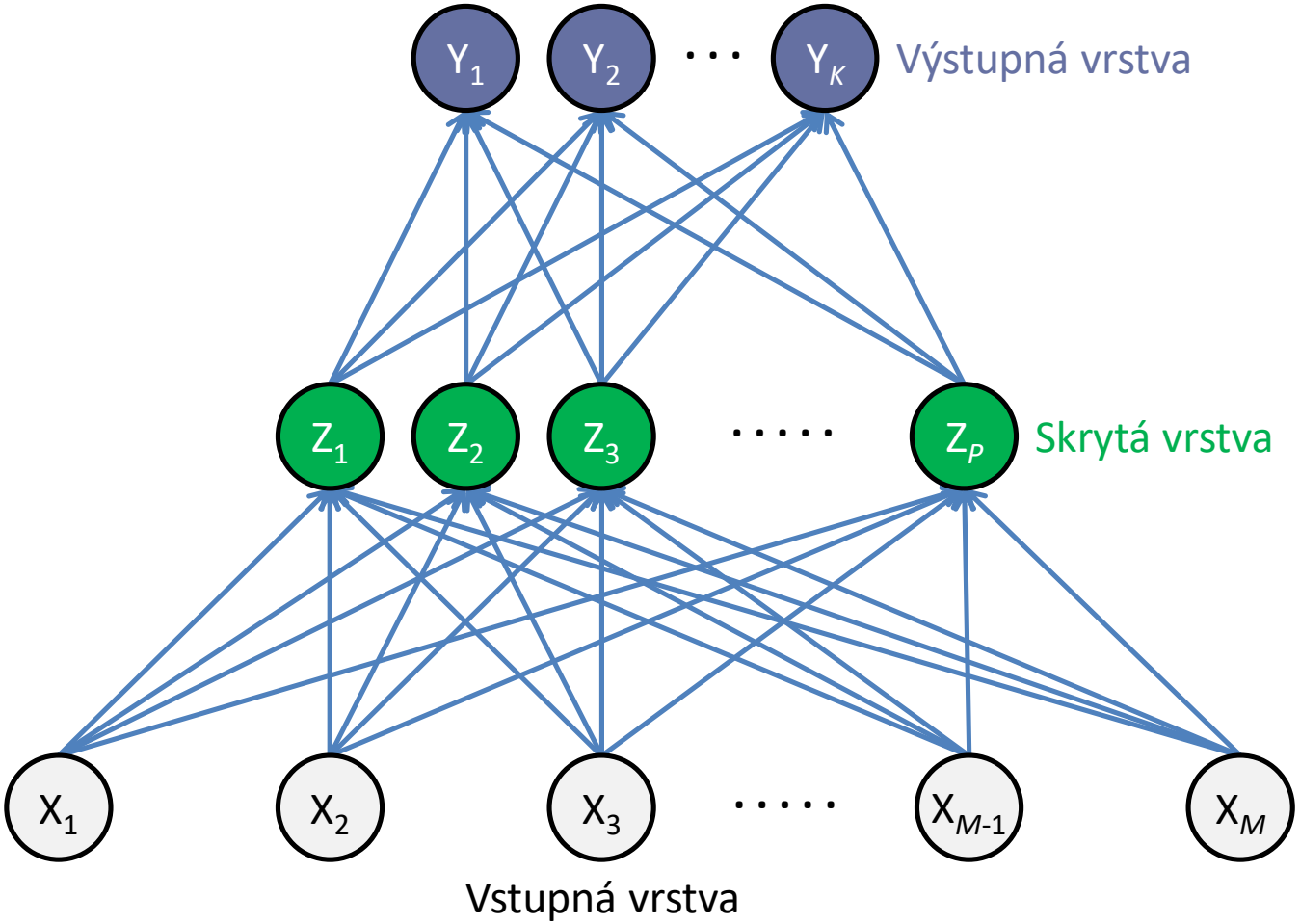
Neurónové siete

- Neurónové siete označujú flexibilný model zložený z výpočtových jednotiek – neurónov, usporiadaných do poprepájaných vrstiev
- Neuróny majú priradené aktivačné funkcie
  - Zvyčajne majú všetky neuróny na jednej vrstve rovnakú aktivačnú funkciu
  - $f(\boldsymbol{x}) = f^{(L)}(\ldots f^{(2)}(f^{(1)}(\boldsymbol{x})))$
- Prepojenia medzi neurónmi zodpovedajú číselným parametrom modelu – tzv. váham
- Znázorňujeme ich graficky – sieťový diagram
  - Uzly zodpovedajú neurónom
  - Hrany zodpovedajú váham

Dopredné neurónové siete (1)

- Základným prediktívnym modelom sú dopredné neurónové siete, ktoré majú nasledujúce vrstvy:
- Vstupnú vrstvu
  - Vstupné neuróny zodpovedajú priamo vstupným atribútom X_1, X_2, ..., X_M
- Jednu (alebo viac) skrytých vrstiev
  - Nelineárne transformujú vstupné dáta
- Výstupnú vrstvu
  - Výstupné neuróny určujú predikciu modelu, môžeme mať naraz viac výstupov Y_1, Y_2, ..., Y_K

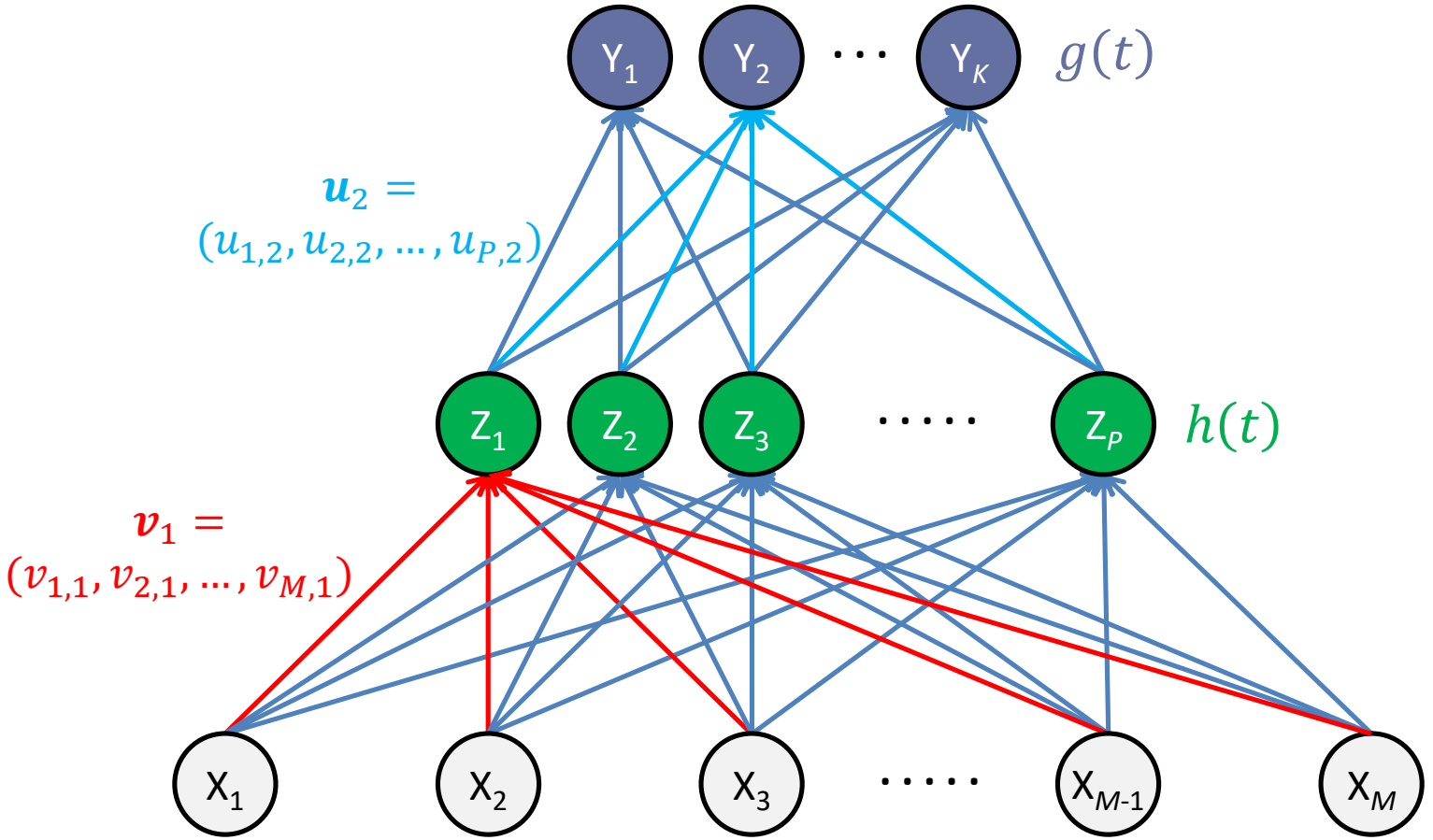
Dopredné neurónové siete (2)



Dopredné neurónové siete (3)

- Váhy pre hrany medzi vstupnou a skrytou vrstvou smerujúce do neurónu Z_p označíme $\boldsymbol{v}_p = (v_{1,p}, v_{2,p}, \dots, v_{M,p})$
- Váhy pre hrany medzi skrytou a výstupnou vrstvou smerujúce do neurónu Y_k označíme $\boldsymbol{u}_k = (u_{1,k}, u_{2,k}, \dots, u_{P,k})$
- Neuróny na skrytej a výstupnej vrstve majú priradenú aktivačnú funkciu $h(t)$ a $g(t)$

Dopredné neurónové siete (4)



Dopredné neurónové siete (5)

- Výpočet pre všetky neuróny na skrytej a výstupnej vrstve prebieha rovnako - výstupná hodnota každého neurónu (aktivácia) sa vypočíta ako vážená suma vstupov do neurónu transformovaná aktivačnou funkciou:

$$z_p = h\left(\sum_{j=1}^{M} v_{j,p} x_j\right) = h\left(\boldsymbol{v}_p^T \boldsymbol{x}\right)$$
$$f_k(\boldsymbol{x}) = g\left(\sum_{p=1}^{P} u_{p,k} z_p\right) = g\left(\boldsymbol{u}_k^T \boldsymbol{z}\right)$$

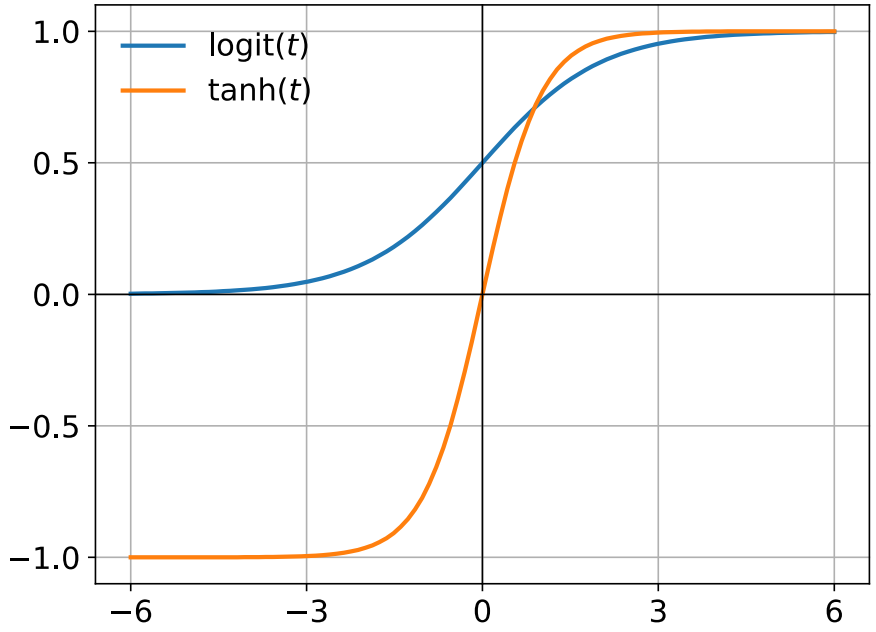
- kde $\boldsymbol{x} = (x_1, \ldots, x_M)^T$ je výstup zo vstupnej vrstvy zodpovedajúci vstupným dátovým atribútom, $\boldsymbol{z} = (z_1, \ldots, z_P)^T$ je výstup zo skrytej vrstvy a $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))^T$ je výsledná predikcia

Aktivačná funkcia na skrytej vrstve

- Ako aktivačná funkcia na skrytej vrstve $h(t)$ sa volí (diferencovateľná) nelineárna funkcia
- Tradične sa používajú napr. sigmoidálne funkcie – logistická funkcia, alebo hyperbolický tangens

$$\text{logit}(t) = \frac{1}{1+e^{-t}}$$

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



Aktivačná funkcia na výstupnej vrstve

- Aktivačná funkcia na výstupnej vrstve $g(t)$ sa volí podľa typu úlohy učenia:
- Pri regresii
  - Na výstupe je priamo predikovaná hodnota bez transformácie (tzn. platí $g(t) = t$)
  - Môžeme naraz predikovať viacero výstupov $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))^T$
- Pri binárnej klasifikácii
  - Stačí mať vo výstupnej vrstve iba jeden neurón s logistickou aktivačnou funkciou, ktorá pretransformuje výstup na odhad pravdepodobnosti $p_1(\boldsymbol{x})$ pre triedu 1 podobne ako pri logistickej regresii

Dopredné neurónové siete (6)

- Pri klasifikácii do jednej z K tried
  - Vo výstupnej vrstve je jeden neurón pre každú triedu, ktorý priamo určuje odhad jej pravdepodobnosti $p_k(\boldsymbol{x}) = f_k(\boldsymbol{x})$ pričom pre klasifikáciu do jednej z K tried musí platiť $0 \leq p_k(\boldsymbol{x}) \leq 1$ a $\sum_{k=1}^{K} p_k(\boldsymbol{x}) = 1$
  - Na transformáciu sa preto používa softmax aktivačná funkcia, ktorá je definovaná ako:

$$g(t_k) = \frac{e^{t_k}}{\sum_{j=1}^{K} e^{t_j}} = \frac{\text{logit}(t_k)}{\sum_{j=1}^{K} \text{logit}(t_j)}$$

    kde $\text{logit}(t)$ je logistická funkcia
  - Pri klasifikácii potom vyberieme triedu, pre ktorú je aktivácia (pravdepodobnosť) maximálna

Model dopredných neurónových sietí

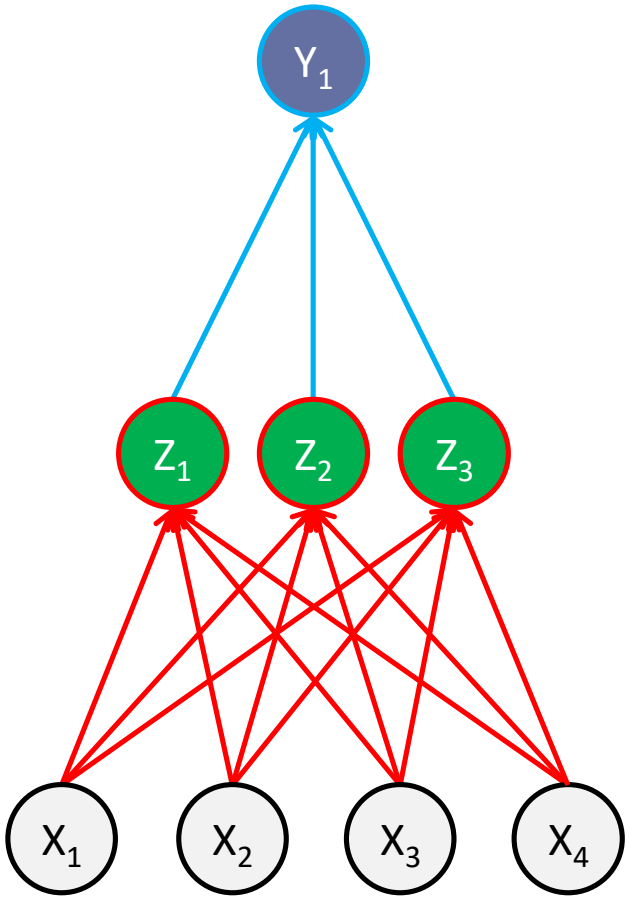
- Výsledný model:
  nelineárna transformácia vstupných atribútov

$$z_p = h\left(\sum_{j=1}^{M} v_{j,p} x_j\right) = h\left(\boldsymbol{v}_p^T \boldsymbol{x}\right)$$

  + lineárna kombinácia

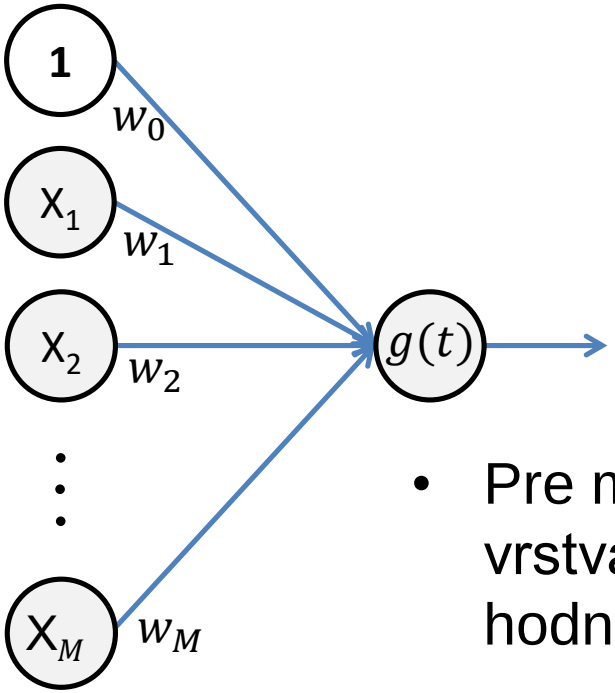
$$f_k(\boldsymbol{x}) = g\left(\sum_{p=1}^{P} u_{p,k} z_p\right) = g\left(\boldsymbol{u}_k^T \boldsymbol{z}\right)$$

- Na rozdiel od bázových funkcií sme zvolili iba jednu funkciu (aktivačnú funkciu na skrytej vrstve), transformácia je však závislá od váh $\boldsymbol{v}_p$, ktoré sa učia podľa dát



Lineárne neurónové siete

- Lineárne modely zodpovedajú dopredným neurónovým sieťam bez skrytej vrstvy:



$$g\left(w_0 + \sum_{j=1}^{M} w_j x_j\right)$$

- Pre model s posunom w_0 je vstupná vrstva rozšírená o konštantný vstup s hodnotou 1 (tzv. *bias* neurónu)

Učenie neurónových sietí

- Váhy $\boldsymbol{w}$ (parametre pre všetky vrstvy) vypočítame minimalizovaním chybovej funkcie na trénovacích dátach $J(f) = J(\boldsymbol{w})$
- **Príklad**
- Regresia s viacerými výstupmi – kvadratická chybová funkcia:

$$J(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{2}\left(y_{i,k} - f_k(\boldsymbol{x}_i)\right)^2$$

- Klasifikácia do viacerých tried – krížová entropia:

$$J(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} -y_{i,k}\log\left(f_k(\boldsymbol{x}_i)\right)$$

Metóda gradientového zostupu (1)

- Pre učenie neurónových sietí sa najčastejšie používa stochastická verzia metódy gradientového zostupu

- Metóda gradientového zostupu (spádu)
- Iteratívna optimalizačná metóda
  1. Začneme zo zvoleného počiatočného bodu $\boldsymbol{w}^{(0)}$
  2. V každej iterácii t zmeníme parametre $\boldsymbol{w}^{(t)} \longrightarrow \boldsymbol{w}^{(t+1)}$ tak, aby sme sa postupne priblížili k minimálnej hodnote funkcie $J(\boldsymbol{w})$

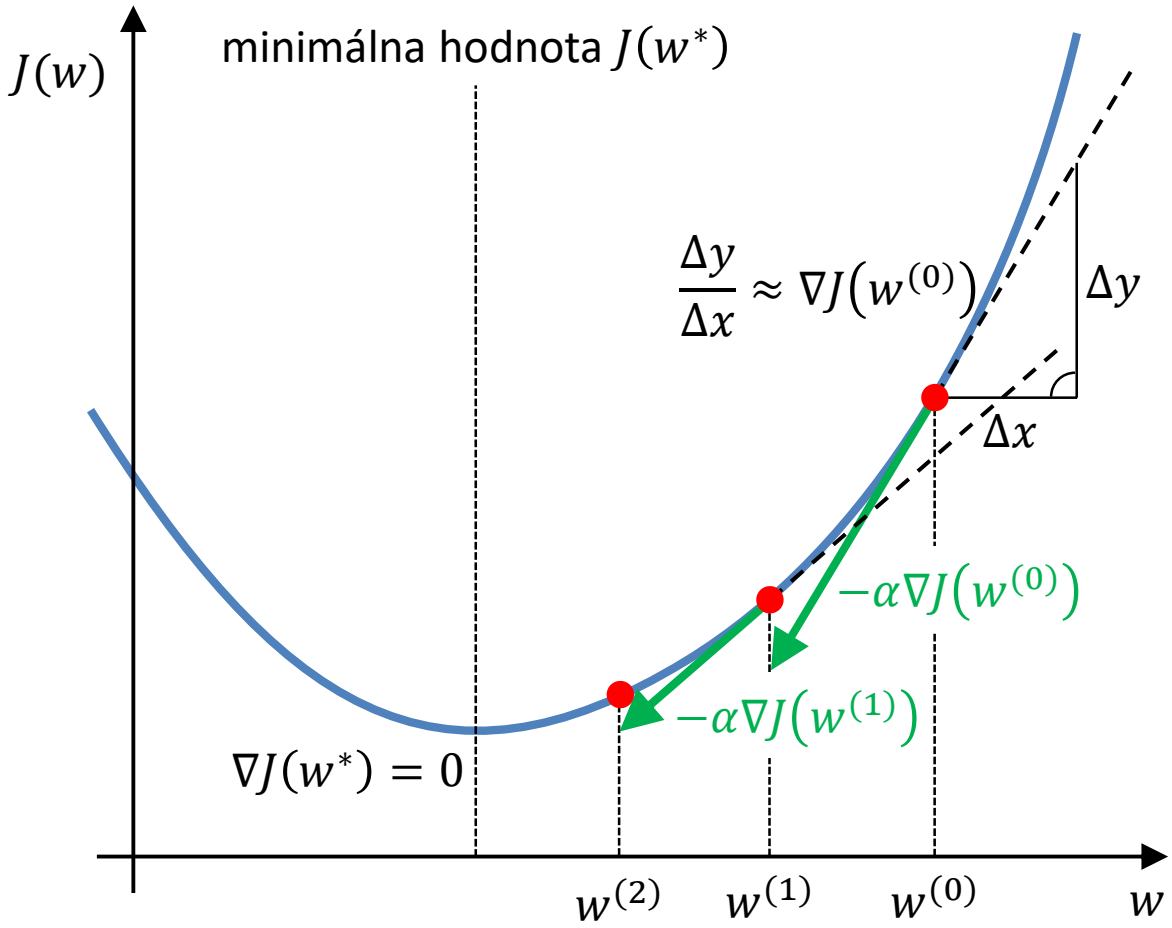
Metóda gradientového zostupu (2)

- Pri metóde gradientového zostupu využijeme vlastnosti gradientu: gradient funkcie v bode $\nabla J\left(\boldsymbol{w}^{(t)}\right)$ určuje smer najväčšieho rastu funkcie (smernica dotyčnice)
- Keďže chceme funkciu minimalizovať, posunieme váhy v opačnom smere (znamienko –)
- Výsledné pravidlo metódy gradientového zostupu:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha \nabla J\left(\boldsymbol{w}^{(t)}\right)$$

- Parameter metódy (hyper-parameter, ktorý musíme nastaviť) $0 < \alpha$ – rýchlosť učenia – určuje, o koľko sa váhy $\boldsymbol{w}$ zmenia pri každom kroku

Metóda gradientového zostupu (3)



Spätné šírenie chyby (1)

- Pri výpočte gradientu môžeme derivovať funkciu $J(\boldsymbol{w})$ po členoch pre jednotlivé trénovacie príklady
- Napr. pre kvadratickú chybu:

$$J(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{2}\left(y_{i,k} - f_k(\boldsymbol{x}_i)\right)^2 = \frac{1}{N}\frac{1}{2}\sum_{i=1}^{N} R_i$$

- Za členy R_i dosadíme model neurónovej siete:

$$f_k(\boldsymbol{x}_i) = g\left(\sum_{p=1}^{P} u_{p,k} z_{p,i}\right) = g\left(\boldsymbol{u}_k^T \boldsymbol{z}_i\right)$$

$$z_{p,i} = h\left(\sum_{j=1}^{M} v_{j,p} x_{i,j}\right) = h\left(\boldsymbol{v}_p^T \boldsymbol{x}_i\right)$$

Spätné šírenie chyby (2)

- Po dosadení odvodíme derivácie pre jednotlivé váhy ako:

$$\frac{\partial R_i}{\partial u_{p,k}} = -2\left(y_{i,k} - f_k(\boldsymbol{x}_i)\right) g'\left(\boldsymbol{u}_k^T \boldsymbol{z}_i\right) z_{i,p}$$

$$\frac{\partial R_i}{\partial v_{j,p}} = \left[-\sum_{k=1}^{K} 2\left(y_{i,k} - f_k(\boldsymbol{x}_i)\right) g'\left(\boldsymbol{u}_k^T \boldsymbol{z}_i\right) u_{p,k}\right] h'\left(\boldsymbol{v}_p^T \boldsymbol{x}_i\right) x_{i,j}$$

- Po dosadení členov R_i pre všetky trénovacie príklady zmeníme v každom kroku t gradientovej metódy jednotlivé váhy podľa pravidla:

$$u_{p,k}^{(t+1)} \leftarrow u_{p,k}^{(t)} - \alpha \sum_{i=1}^{N} \frac{\partial R_i}{\partial u_{p,k}^{(t)}}$$

$$v_{j,p}^{(t+1)} \leftarrow v_{j,p}^{(t)} - \alpha \sum_{i=1}^{N} \frac{\partial R_i}{\partial v_{j,p}^{(t)}}$$

Spätné šírenie chyby (3)

- Členy gradientu si môžeme rozdeliť na každej vrstve na "chybu" a vstup z predchádzajúcej vrstvy:

$$\frac{\partial R_i}{\partial u_{p,k}} = -2\left(y_{i,k} - f_k(\boldsymbol{x}_i)\right) g'\left(\boldsymbol{u}_k^T \boldsymbol{z}_i\right) z_{i,p}$$

$$\frac{\partial R_i}{\partial v_{j,p}} = \left[-\sum_{k=1}^{K} \overbrace{2\left(y_{i,k} - f_k(\boldsymbol{x}_i)\right) g'\left(\boldsymbol{u}_k^T \boldsymbol{z}_i\right)}^{} u_{p,k}\right] h'\left(\boldsymbol{v}_p^T \boldsymbol{x}_i\right) x_{i,j}$$

- Pričom vďaka pravidlu zreťazenia pri derivovaní platí, že chyba na predchádzajúcej vrstve je vždy závislá od váženej chyby nasledujúcej vrstvy
- Pri učení sa najprv vypočíta predikcia podľa aktuálnych váh z ktorej sa určí chyba na výstupe, ktorá sa spätne šíri na predchádzajúce vrstvy

Metóda stochastického gradientového zostupu (1)

- V základnom pravidle gradientovej metódy sa gradient vypočíta na celej trénovacej množine pri každej zmene váh
  - Nevýhodné hlavne pri veľkom počte trénovacích príkladov

- Aby sa urýchlilo konvergovanie, gradient je možné v každom kroku vypočítať iba približne na náhodne zvolenej podmnožine m trénovacích príkladov – tzv. *mini-batch* učenie
  - m (veľkosť podmnožiny) je hyper-parameter, ktorý musíme zvoliť
- Hraničný prípad je, keď sa v každom kroku vypočíta gradient iba pre 1 príklad, zmenia sa váhy, a pokračuje sa ďalším príkladom – tzv. *online* učenie

Metóda stochastického gradientového zostupu (2)

- Učenie najčastejšie prebieha po epochách
- Počet epoch si musíme stanoviť podľa priebehu učenia (hyper-parameter)

1. Náhodne preusporiadame trénovacie príklady
2. Vyberieme m nasledujúcich príkladov (mini-batch)
3. Vypočítame gradient na vybraných príkladoch a prepočítame váhy
4. Ak sme prešli všetky trénovacie príklady, pokračujeme ďalšou epochou (krok 1), inak vyberieme nasledujúce príklady a pokračujeme ďalšou iteráciou (krok 2)

Inicializácia váh a regularizácia učenia (1)

- Ako inicializovať počiatočné váhy $\boldsymbol{w}^{(0)}$?
- Pri inicializácii sa používa heuristika, kedy sa váhy inicializujú náhodne tak, aby mali malú absolútnu hodnotu:
  - Vstup do neurónu na skrytej vrstve je daný $t = \sum_{j=1}^{M} v_{j,p} x_j = \boldsymbol{v}_p^T \boldsymbol{x}$
  - Pri sigmoidálnych funkciách, ak budú mať váhy $\boldsymbol{v}_p$ a vstupné dáta $\boldsymbol{x}$ malú absolútnu hodnotu, vstup t bude v okolí 0 hodnoty, kde má sigmoidálna funkcia $h(t)$ približne lineárny priebeh – celý model neurónovej siete bude lineárny

Inicializácia váh a regularizácia učenia (2)

- Tzn. na začiatku učenia začneme s jednoduchým modelom ktorý aproximuje lineárnu funkciu
- Pri učení sa budú váhy postupne meniť podľa dát, a ak je to potrebné, model sa stane nelineárnym zvyšovaním $\left|\boldsymbol{v}_p^T \boldsymbol{x}\right|$
- Jednoduchým spôsobom regularizácie učenia neurónových sietí je predčasné ukončenie učenia
  - Trénovacia chyba nemusí byť minimálna, ale pri ohraničenom modeli sa môže zabrániť preučeniu a zníži sa zovšeobecnená chyba

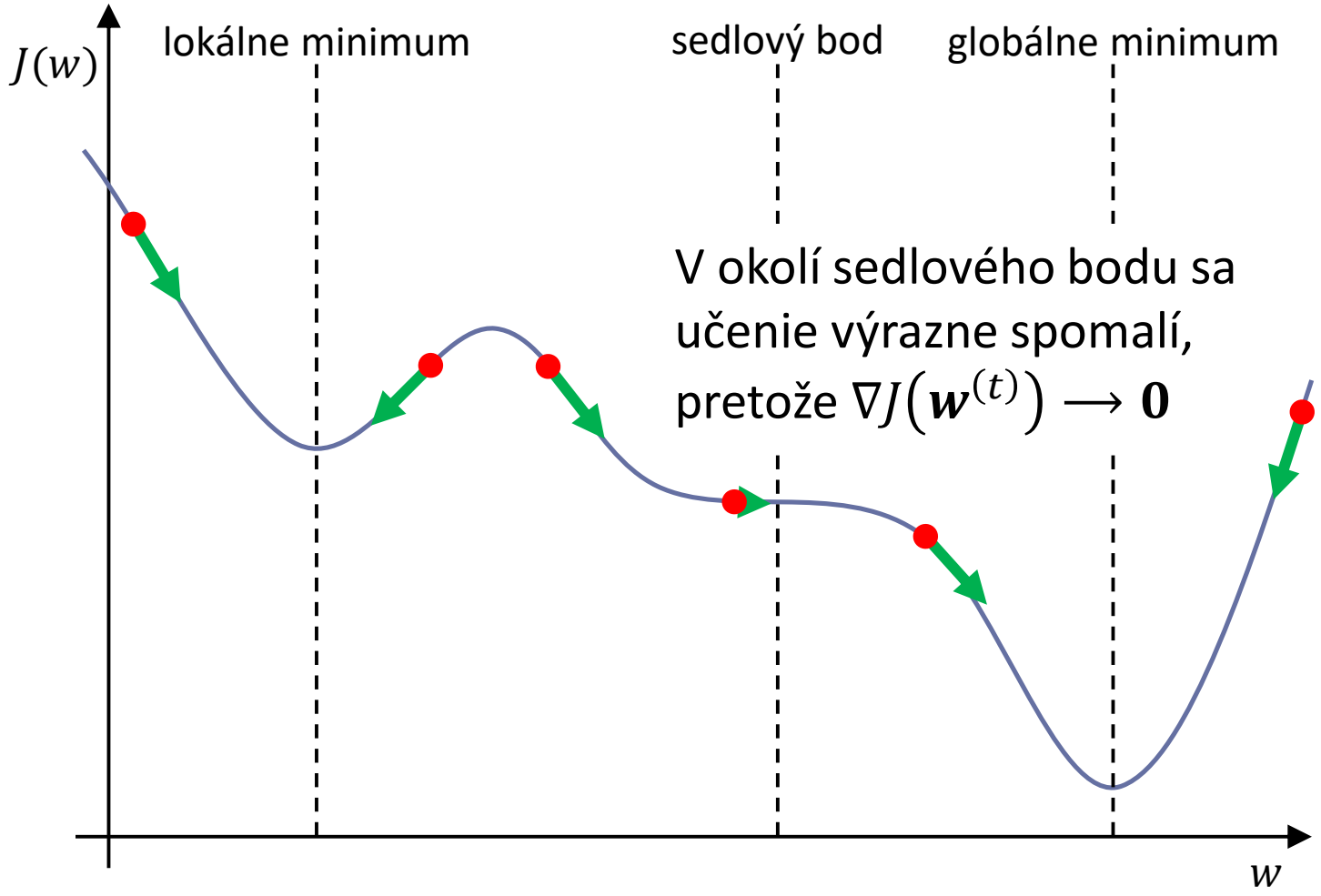
Inicializácia váh a regularizácia učenia (3)

- Populárnou schémou je napr. Glorot (Xavier) inicializácia, pri ktorej sú váhy zvolené náhodne z normálneho rozdelenia s 0 strednou hodnotou a štandardnou odchýlkou $\sqrt{2/(m+k)}$, kde m je počet váh, ktoré vstupujú do neurónu a k počet váh, ktoré z neurónu vystupujú
- Pri učení je takisto vhodné normalizovať vstupné dáta na 0 strednú hodnotu a 1 štandardnú odchýlku

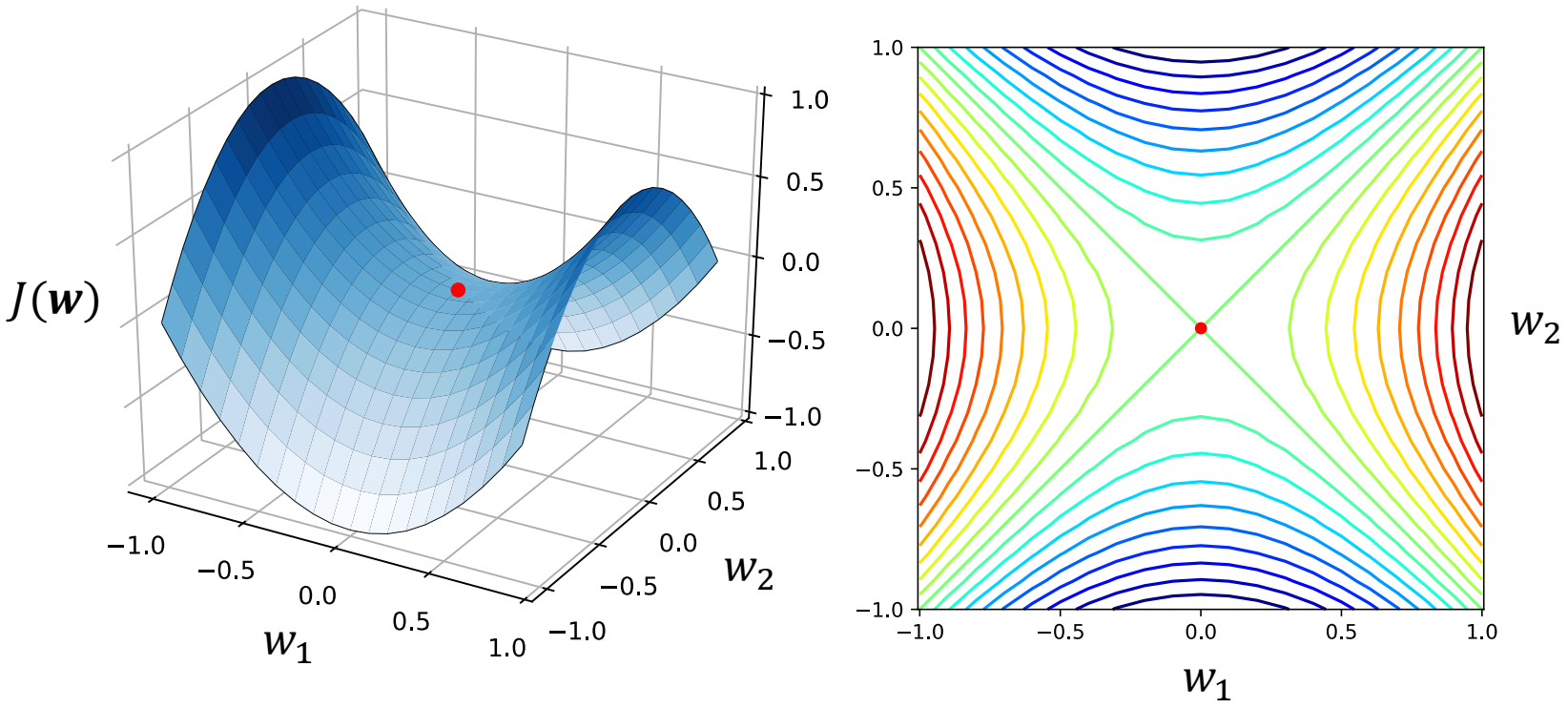
Metóda gradientového zostupu – vlastnosti (1)

- Pre konvexné funkcie gradientová metóda vždy skonverguje do globálneho minima nezávisle od počiatočného bodu $\boldsymbol{w}^{(0)}$
  - Presnosť, s akou sa priblíži k minimu, závisí od rýchlosti učenia (na začiatku môžeme zvoliť väčšiu rýchlosť, ktorú postupne znižujeme)
- Neurónové siete sú však nekonvexné funkcie
  - Gradientová metóda môže pri učení uviaznuť v lokálnom minime, alebo sedlovom bode
- To platí pre obyčajnú aj stochastickú verziu

Metóda gradientového zostupu – vlastnosti (2)



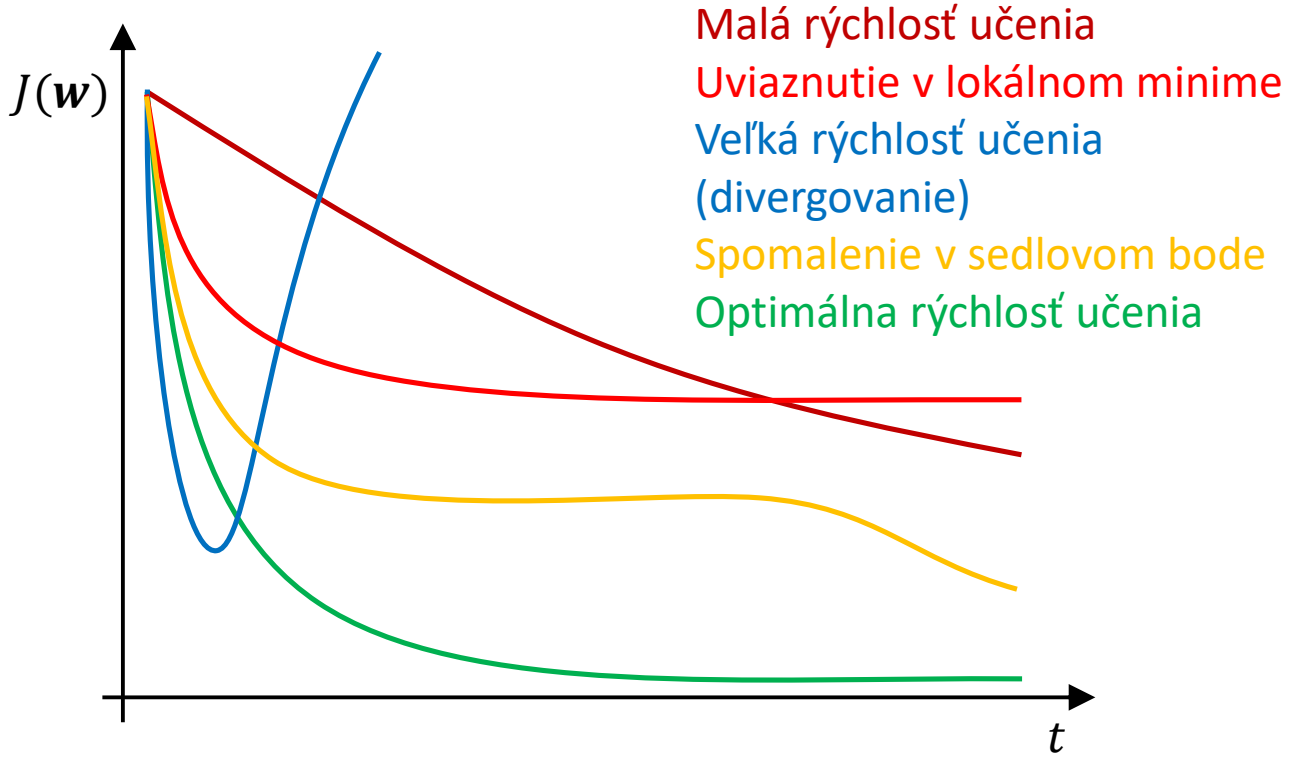
Príklad sedlového bodu



V sedlovom bode [0,0] má funkcia v smere atribútu w_1 minimum, a v smere atribútu w_2 maximum.

Priebeh učenia a kontrola konvergovania

- Počas učenia preto musíme kontrolovať chybu na trénovacích dátach $J(\boldsymbol{w})$ po jednotlivých krokoch a optimálne nastaviť rýchlosť učenia

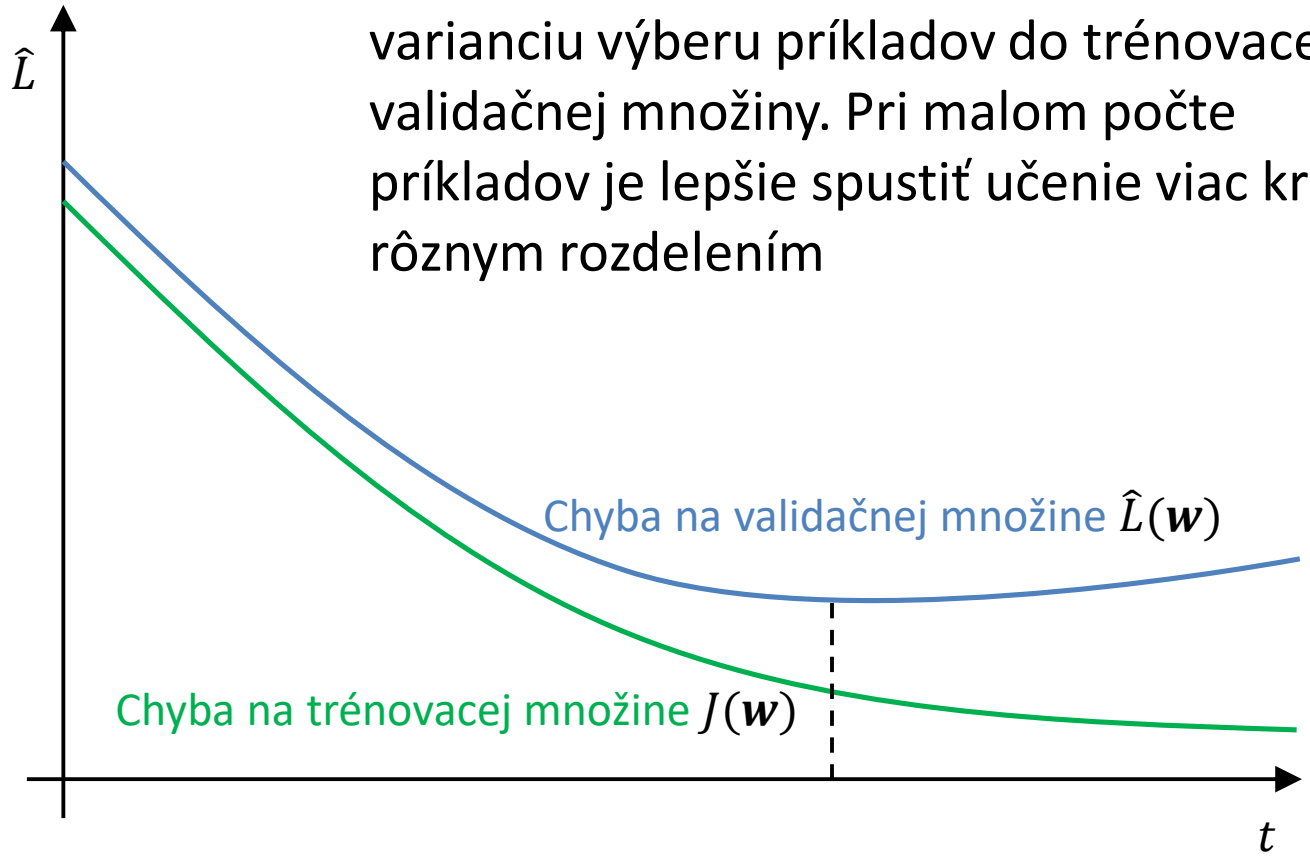


Učenie neurónových sietí – zhrnutie (1)

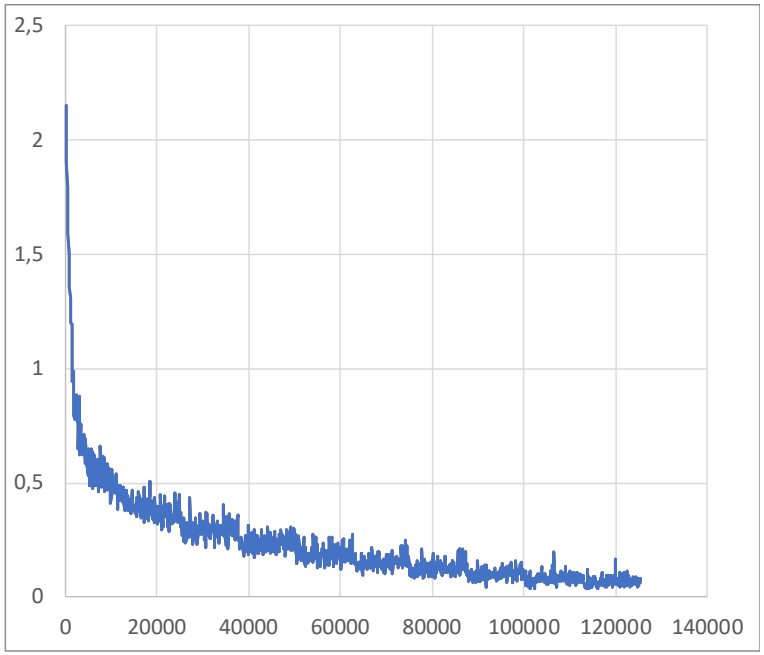
- Potrebujeme monitorovať a) konvergovanie gradientovej metódy a b) odhad zovšeobecnenej chyby, aby sme mohli učenie regulovať
- Dáta sa zvyčajne náhodne rozdelia na tri podmnožiny:
  - Na trénovacej množine učíme model a monitorujeme konvergovanie
  - Na validačnej množine vyhodnotíme odhad zovšeobecnenej chyby, podľa ktorého sa rozhodneme o predčasnom ukončení učenia
  - Na testovacej množine vyhodnotíme kvalitu výsledného modelu

Učenie neurónových sietí – zhrnutie (2)

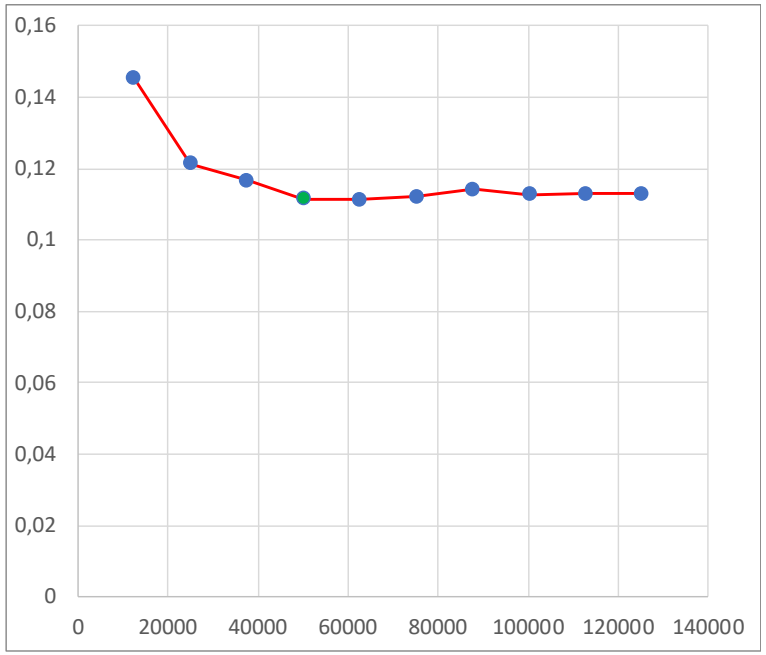
- Obe chyby sú iba odhadom – pozor na varianciu výberu príkladov do trénovacej a validačnej množiny. Pri malom počte príkladov je lepšie spustiť učenie viac krát s rôznym rozdelením



Učenie neurónových sietí – zhrnutie (3)



Trénovacia chyba – fluktuácie sú spôsobené varianciou stochastického gradientového zostupu. Pre lepšie sledovanie trendu je lepšie spriemerniť hodnoty pre viacero iterácií.



Zovšeobecnená chyba počas učenia – odhad na validačnej množine po každej epoche.

Neurónové siete – zhrnutie

- **Výhody**:
- Flexibilný modulárny model:
  - Môžeme pridať viac neurónov, alebo viac skrytých vrstiev, čím sa zvýši "kapacita" modelu - vieme ním reprezentovať zložité závislosti

- **Nevýhody**:
- Nekonvexná optimalizačná úloha - štandardná metóda učenia nemusí vždy optimálne fungovať
- Model náchylnejší na preučenie - regularizácia učenia je veľmi dôležitá zvlášť pri veľkom počte neurónov

Rozšírenia gradientovej metódy

- Snažíme sa zrýchliť konvergovanie
- Snažíme sa zabrániť uviaznutiu v lokálnom minime, alebo spomaleniu učenia v sedlovom bode
  - Momentové metódy
  - Metódy s adaptívnou rýchlosťou učenia

Gradientová metóda s momentom (1)

- **Pravidlo**
- Okrem vektora váh budeme v každom kroku prepočítavať aj vektor momentu $\boldsymbol{m}^{(t)}$, ktorý má rovnaký rozmer ako vektor váh. Na začiatku učenia je vektor momentu inicializovaný na $\mathbf{0}$. Váhy a moment prepočítame v každom kroku podľa pravidiel:

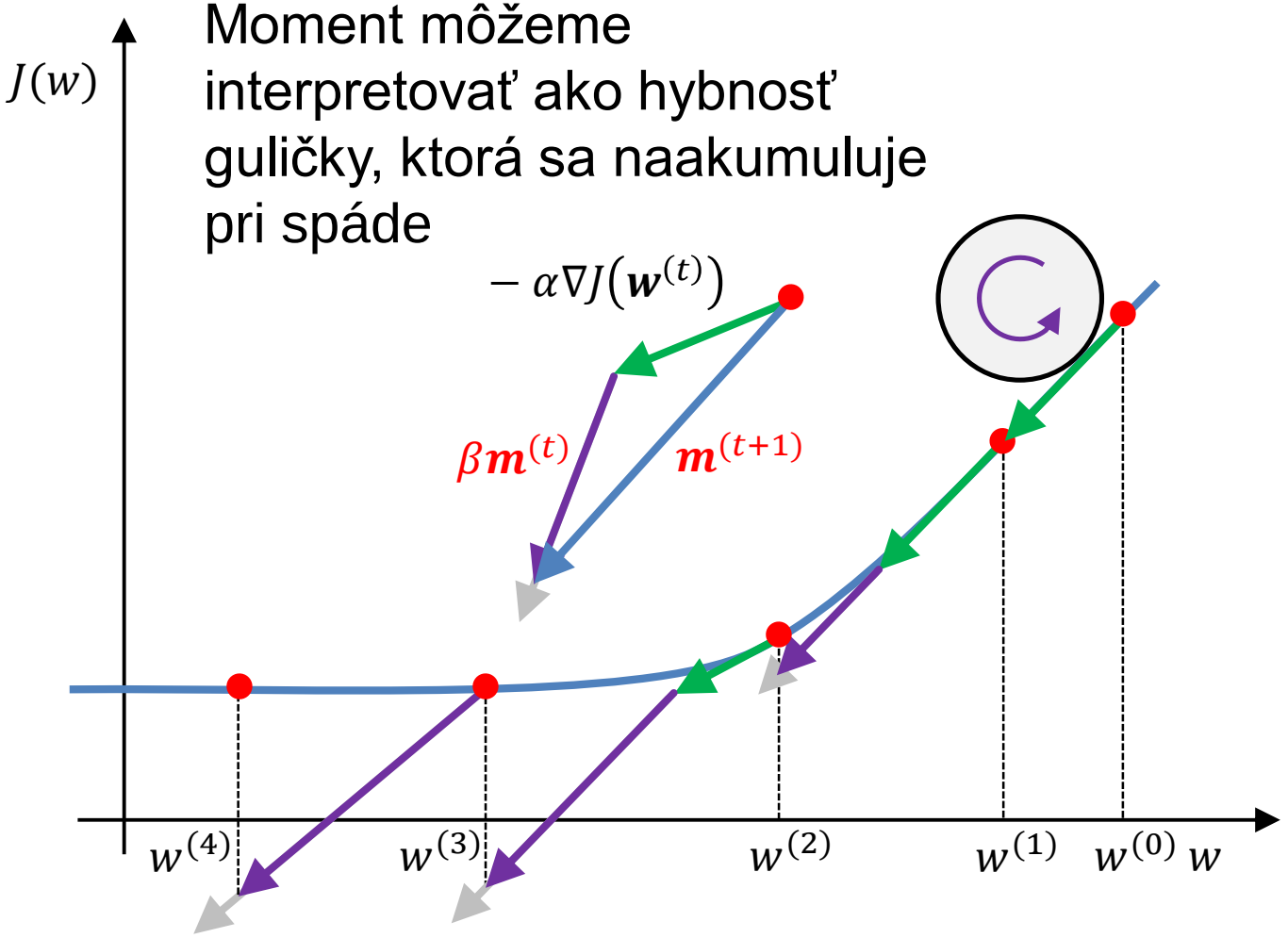
$$\boldsymbol{m}^{(t+1)} \leftarrow \beta \boldsymbol{m}^{(t)} - \alpha \nabla J\left(\boldsymbol{w}^{(t)}\right)$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{m}^{(t+1)}$$

- Kde $0 < \alpha$ je rýchlosť učenia a $0 \leq \beta < 1$ je momentová konštanta (zvyčajne sa volí napr. 0.9)

Gradientová metóda s momentom (2)

Moment môžeme interpretovať ako hybnosť guličky, ktorá sa naakumuluje pri spáde



Gradientová metóda s momentom (3)

- **Výhody**
- V niektorých prípadoch nám umožňuje zachovať dostatočnú rýchlosť učenia v sedlovom bode, resp. "vyskočiť" z lokálneho minima
- Zrýchli sa učenie v oblastiach, kde je veľký gradient v jednom smere (pre jeden parameter) ale malý v inom
- **Nevýhody**
- Väčšie oscilácie v minime

Nesterovov moment

- **Pravidlo**

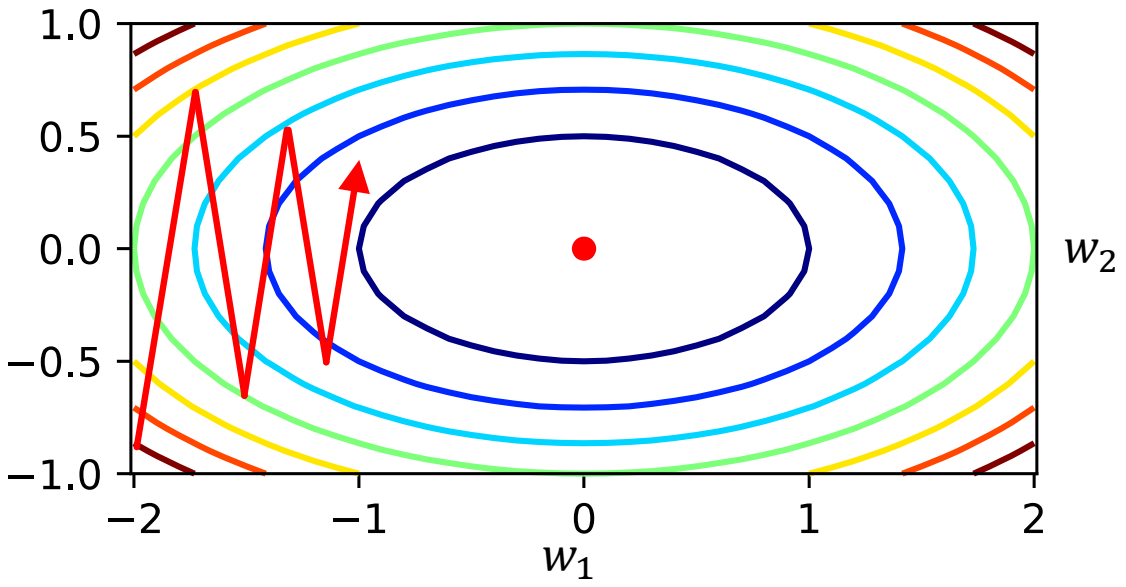
$$\boldsymbol{m}^{(t+1)} \leftarrow \beta \boldsymbol{m}^{(t)} - \alpha \nabla J\left(\boldsymbol{w}^{(t)} + \beta \boldsymbol{m}^{(t)}\right)$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{m}^{(t+1)}$$

- Nezávisle od toho, aký je gradient v aktuálnom bode $\boldsymbol{w}^{(t)}$, vieme, že sa váhy zmenia aspoň o $\beta \boldsymbol{m}^{(t)}$, tzn. gradient môžeme odhadnúť až po priblížení v bode $\boldsymbol{w}^{(t)} + \beta \boldsymbol{m}^{(t)}$
  - Skôr začneme spomaľovať, keď hodnota funkcie v bode $\boldsymbol{w}^{(t)} + \beta \boldsymbol{m}^{(t)}$ začne rásť, čo zníži oscilácie

Metódy s adaptívnou rýchlosťou učenia

- V prípade, kedy je gradient v jednom smere oveľa väčší než v inom, štandardná gradientová metóda osciluje, čo spôsobuje pomalé učenie
  - Chceme zrýchliť učenie v jednom smere, ale tlmiť oscilácie v inom, tzn. musíme mať nezávislú rýchlosť pre každý parameter



Adagrad (1)

- Vyjadríme si gradientové pravidlo pre jednotlivé parametre w_j

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \alpha \frac{\partial J\left(\boldsymbol{w}^{(t)}\right)}{\partial w_j} = w_j^{(t)} - \alpha g_j^{(t)}$$

  - Ako $g_j^{(t)}$ sme označili zložku gradientu pre parameter w_j pri iterácii t
  - Pri štandardnom pravidle zmeníme všetky váhy s rovnakou rýchlosťou učenia α. Adagrad prepočíta v každom kroku t rýchlosť učenia nezávisle pre každý parameter

Adagrad (2)

- **Pravidlo**:

$$w_j^{(t)} \leftarrow w_j^{(t)} - \frac{\alpha}{\sqrt{\sum_{i=1}^{t}\left(g_j^{(i)}\right)^2 + \varepsilon}} g_j^{(t)}$$

- Adagrad postupne zmenšuje rýchlosť učenia pre všetky váhy – avšak pre menej často menené parametre sa rýchlosť učenia zníži menej výrazne (ε je len malé číslo aby sa zabránilo deleniu 0)
- Výhodou je, že už nemusí byť potrebné manuálne nastavovať α (stačí ponechať prednastavenú hodnotu $\alpha = 0.01$)

RMSprop

- Pri Adagrade sa v menovateli akumulujú všetky predchádzajúce hodnoty gradientu
  - Učenie sa môže časom úplne utlmiť
- RMSprop akumuluje v menovateli iba kĺzavý priemer $n_{j,t}$, pričom sa využíva rekurentný zápis:

$$n_{j,t} = \gamma n_{j,t-1} + (1-\gamma)\left(g_j^{(t)}\right)^2$$

- Kde γ je tlmiaci faktor, ktorý postupne potlačí predchádzajúce prírastky, najčastejšie sa volí $\gamma = 0.9$, s počiatočnou rýchlosťou $\alpha = 0.001$

Adam

- Adam kombinuje výhody momentovej metódy a adaptívnej zmeny rýchlosti učenia nezávislej pre každý parameter
- Moment môžeme zapísať ako kĺzavý priemer gradientu, tzn. metóda udržiava kĺzavý priemer:
  - $m_{j,t} = \beta_1 m_{j,t-1} + (1-\beta_1) g_j^{(t)}$ pre moment
  - $n_{j,t} = \beta_2 n_{j,t-1} + (1-\beta_2) \left(g_j^{(t)}\right)^2$ pre adaptívnu zmenu rýchlosti podobne ako pri RMSprop
- Parametre metódy: $\beta_1 = 0.9$ – momentová konštanta (zodpovedá β), $\beta_2 = 0.999$ – tlmiaci faktor (zodpovedá γ), $\alpha = 0.001$ – rýchlosť učenia