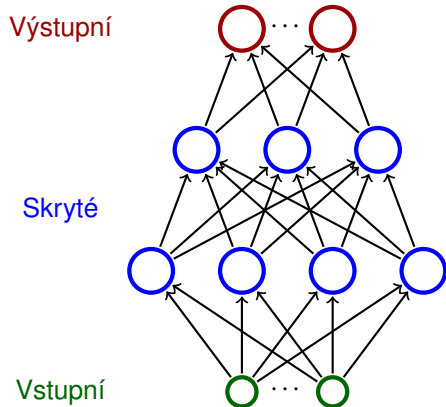


Vícevrstvá síť

Organizační dynamika:



- ▶ Neurony jsou rozděleny do **vrstev** (vstupní a výstupní vrstva, obecně několik skrytých vrstev)
- ▶ Vrstvy číslujeme od 0; vstupní vrstva je nultá
 - ▶ Např. třívrstvá síť se skládá z jedné vstupní, dvou skrytých a jedné výstupní vrstvy.
- ▶ Neurony v ℓ -té vrstvě jsou spojeny se všemi neurony ve vrstvě $\ell + 1$.
- ▶ Vícevrstvou síť lze zadat počty neuronů v jednotlivých vrstvách (např. 2-4-3-2)

Značení:

- ▶ Označme
 - ▶ X množinu vstupních neuronů
 - ▶ Y množinu výstupních neuronů
 - ▶ Z množinu všech neuronů (tedy $X, Y \subseteq Z$)
- ▶ jednotlivé neurony budeme značit indexy i, j apod.
- ▶ ξ_j je vnitřní potenciál neuronu j po skončení výpočtu
- ▶ y_j je stav (výstup) neuronu j po skončení výpočtu
(zde definujeme $y_0 = 1$ jako hodnotu formálního jednotkového vstupu)
- ▶ w_{ji} je váha spoje **od** neuronu i **do** neuronu j
(zejména w_{j0} je váha speciálního jednotkového vstupu, tj. $w_{j0} = -b_j$ kde b_j je bias neuronu j)
- ▶ j_{\leftarrow} je množina všech neuronů, **z nichž** vede spoj do j
(zejména $0 \in j_{\leftarrow}$)
- ▶ j_{\rightarrow} je množina všech neuronů, **do nichž** vede spoj z j

Aktivní dynamika:

- ▶ vnitřní potenciál neuronu j :

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- ▶ aktivační funkce σ_j pro neuron j je libovolná diferencovatelná funkce (upřesním později)
- ▶ Stav nevstupního neuronu $j \in Z \setminus X$ po skončení výpočtu je

$$y_j = \sigma_j(\xi_j)$$

(y_j závisí na konfiguraci \vec{w} a vstupu \vec{x} , proto budu občas psát $y_j(\vec{w}, \vec{x})$)

- ▶ Síť počítá funkci z $\mathbb{R}^{|X|}$ do $\mathbb{R}^{|Y|}$. Výpočet probíhá po vrstvách. Na začátku jsou hodnoty vstupních neuronů nastaveny na vstup sítě. V kroku ℓ jsou vyhodnoceny neurony z ℓ -té vrstvy.

Adaptivní dynamika:

- ▶ Dána množina \mathcal{T} **třéninkových vzorů** tvaru

$$\{(\vec{x}_k, \vec{d}_k) \mid k = 1, \dots, p\}$$

kde každé $\vec{x}_k \in \mathbb{R}^{|\mathcal{X}|}$ je vstupní vektor a každé $\vec{d}_k \in \mathbb{R}^{|\mathcal{Y}|}$ je očekávaný výstup sítě. Pro každé $j \in \mathcal{Y}$ označme d_{kj} očekávaný výstup neuronu j pro vstup \vec{x}_k (vektor \vec{d}_k lze tedy psát jako $(d_{kj})_{j \in \mathcal{Y}}$).

- ▶ **Chybová funkce:**

$$E(\vec{w}) = \sum_{k=1}^p E_k(\vec{w})$$

kde

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in \mathcal{Y}} (y_j(\vec{w}, \vec{x}_k) - d_{kj})^2$$

Dávkový algoritmus (gradientní sestup):

Algoritmus počítá posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \dots$

- ▶ váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- ▶ v t -tém kroku (zde $t = 1, 2, \dots$) je $\vec{w}^{(t)}$ vypočteno takto:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)}$$

kde

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t-1)})$$

je změna váhy w_{ji} v t -tém kroku a $0 < \varepsilon(t) \leq 1$ je rychlost učení v t -tém kroku.

Všimněte si, že $\frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t-1)})$ je komponenta gradientu ∇E , tedy změnu vah v t -tém kroku lze zapsat také takto: $\vec{w}^{(t)} = \vec{w}^{(t-1)} - \varepsilon(t) \cdot \nabla E(\vec{w}^{(t-1)})$.

Online algoritmus:

Algoritmus počítá posloupnost vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \dots$

- ▶ váhy v $\vec{w}^{(0)}$ jsou inicializovány náhodně blízko 0
- ▶ v t -tém kroku (zde $t = 1, 2, \dots$) je $\vec{w}^{(t)}$ vypočteno takto:

$$w_{ji}^{(t)} = \vec{w}^{(t-1)} + \Delta w_{ji}^{(t)}$$

kde

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E_k}{\partial w_{ji}}(w_{ji}^{(t-1)})$$

kde $k = ((t - 1) \bmod p) + 1$ a $0 < \varepsilon(t) \leq 1$ je rychlost učení v t -tém kroku.

Lze použít i **stochastickou verzi** tohoto algoritmu, v níž je k voleno náhodně z $\{1, \dots, p\}$.

Vícevrstvá síť - gradient chybové funkce

Pro každé w_{ji} máme

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^p \frac{\partial E_k}{\partial w_{ji}}$$

kde pro každé $k = 1, \dots, p$ platí

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial y_j} \cdot \sigma'_j(\xi_j) \cdot y_i$$

a pro každé $j \in Z \setminus X$ dostaneme

$$\frac{\partial E_k}{\partial y_j} = y_j - d_{kj} \quad \text{pro } j \in Y$$

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j \rightarrow} \frac{\partial E_k}{\partial y_r} \cdot \sigma'_r(\xi_r) \cdot w_{rj} \quad \text{pro } j \in Z \setminus (Y \cup X)$$

(Zde všechna y_j jsou ve skutečnosti $y_j(\vec{w}, \vec{x}_k)$).

Vícevrstvá síť - gradient chybové funkce

- ▶ Pokud $\sigma_j(\xi) = \frac{1}{1+e^{-\lambda_j \xi}}$ pro každé $j \in Z$ pak

$$\sigma'_j(\xi_j) = \lambda_j y_j (1 - y_j)$$

a dostaneme přesně gradient z minulé přednášky.

- ▶ Pokud $\sigma_j(\xi) = a \cdot \tanh(b \cdot \xi_j)$ pro každé $j \in Z$ pak

$$\sigma'_j(\xi_j) = \frac{b}{a} (a - y_j)(a + y_j)$$

Vícevrstvá síť - výpočet gradientu

Algoritmicky lze $\frac{\partial E}{\partial w_{ji}}$ spočítat takto:

Polož $\mathcal{E}_{ji} := 0$ (na konci výpočtu bude $\mathcal{E}_{ji} = \frac{\partial E}{\partial w_{ji}}$)

Pro každé $k = 1, \dots, p$ udělej následující

1. spočítej y_j pro každé $j \in Z$ a k -tý vzor, tedy $y_j = y_j(\vec{w}, \vec{x}_k)$ (tj. vyhodnoť síť ve standardním aktivním režimu)
2. pro všechna $j \in Z$ spočítej $\frac{\partial E_k}{\partial y_j}$ pomocí zpětného šíření (viz. následující slajd!)
3. spočítej $\frac{\partial E_k}{\partial w_{ji}}$ pro všechna w_{ji} pomocí vzorce

$$\frac{\partial E_k}{\partial w_{ji}} := \frac{\partial E_k}{\partial y_j} \cdot \sigma'_j(\xi_j) \cdot y_i$$

4. $\mathcal{E}_{ji} := \mathcal{E}_{ji} + \frac{\partial E_k}{\partial w_{ji}}$

Výsledné E_{ji} se rovná $\frac{\partial E}{\partial w_{ji}}$.

Vícevrstvá síť - zpětné šíření

$\frac{\partial E_k}{\partial y_j}$ lze spočítat v lineárním čase (s jednotkovou aritmetikou) pomocí zpětného šíření:

- ▶ spočítáme $\frac{\partial E_k}{\partial y_j}$ pro $j \in Y$ pomocí vzorce $\frac{\partial E_k}{\partial y_j} = y_j - d_{kj}$
- ▶ rekurzivně spočítáme zbylé $\frac{\partial E_k}{\partial y_j}$:

Nechť j je v ℓ -té vrstvě a předpokládejme, že $\frac{\partial E_k}{\partial y_r}$ už máme spočítáno pro všechny neurony z vyšších vrstev (tedy vrstev $\ell + 1, \ell + 2, \dots$).

Pak lze $\frac{\partial E_k}{\partial y_j}$ spočítat pomocí vzorce

$$\frac{\partial E_k}{\partial y_j} = \sum_{r \in j^{\rightarrow}} \frac{\partial E_k}{\partial y_r} \cdot \sigma'_r(\xi_r) \cdot w_{rj}$$

protože všechny neurony $r \in j^{\rightarrow}$ patří do vrstvy $\ell + 1$.

Praktické otázky gradientního sestupu

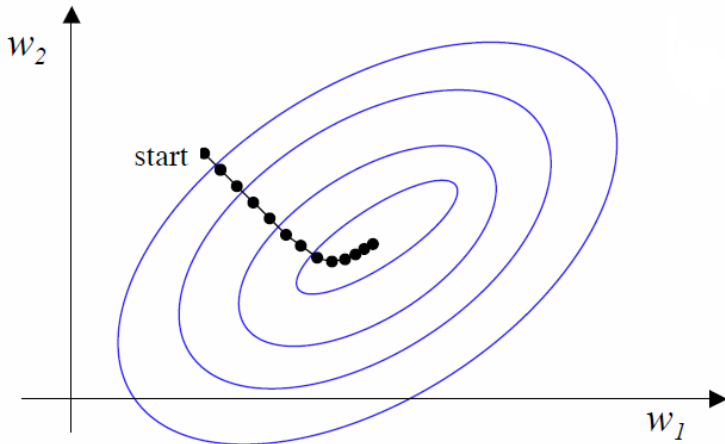
- ▶ Otázky týkající se efektivity učení:
 - ▶ Dávkový nebo online algoritmus?
 - ▶ Je nutné data předzpracovat?
 - ▶ Jak volit iniciální váhy?
 - ▶ Je nutné vhodně volit požadované hodnoty sítě?
 - ▶ Jak volit rychlost učení $\varepsilon(t)$?
 - ▶ Dává gradient nejlepší směr změny vah?
- ▶ Otázky týkající se výsledného modelu:
 - ▶ Kdy zastavit učení?
 - ▶ Kolik vrstev sítě a jak velkých?
 - ▶ Jak velkou tréninkovou množinu použít?
 - ▶ Jak zlepšit vlastnosti (přesnost a schopnost generalizace) výsledného modelu?

Poznámky o konvergenci gradientního sestupu

- ▶ Chybová funkce může být velmi komplikovaná:
 - ▶ může obsahovat mnoho lokálních minim, učící algoritmus v nich může skončit
 - ▶ může obsahovat mnoho „plochých“ míst s velmi malým gradientem, zejména pokud se aktivační funkce tzv. saturují (tedy jejich hodnoty jsou blízko extrémům; v případě sigmoidálních funkcí to znamená, že mají velké argumenty)
 - ▶ může obsahovat velmi strmá místa, což vede k přeskočení minim gradientním sestupem
- ▶ pro velmi malou rychlost učení máme větší šanci dokonvergovat do lokálního minima, ale konvergence je hodně pomalá
Teorie: pokud $\varepsilon(t) = \frac{1}{t}$ pak dávkový i stochastický gradientní sestup konvergují k lokálnímu minimu (nicméně extrémně pomalu).
- ▶ pro příliš velkou rychlost učení může vektor vah střídavě přeskakovat minimum nebo dokonce divergovat

Gradientní sestup - ilustrace

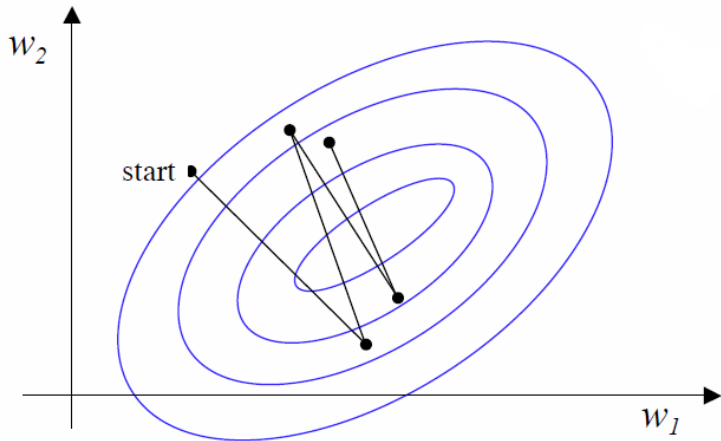
Gradientní sestup s malou rychlostí učení:



Více méně hladká křivka, každý krok je kolmý na vrstevnici.

Gradientní sestup - ilustrace

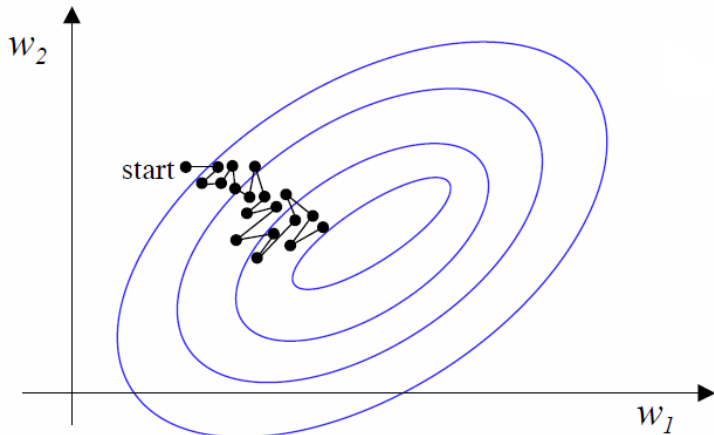
Gradientní sestup s příliš velkou rychlostí učení:



Jednotlivé kroky přeskakují minimum, učení je pomalé.

Gradientní sestup - ilustrace

Online učení:



Kroky nemusí být kolmé na vrstevnice, může hodně kličkovat.

Dávkový vs online učící algoritmus

Výhody dávkového:

- ▶ realizuje přesně gradientní sestup pro chybovou fci (většinou konverguje k lokálnímu minimu)
- ▶ snadno paralelizovatelný (vzory je možné zpracovávat odděleně)

Nevýhody dávkového:

- ▶ paměťová náročnost (musí si pamatovat chyby všech vzorů)
- ▶ redundantní data nepřidávají informaci ke gradientu.

Výhody online (stochastického)

- ▶ stochastický má šanci uniknout z okolí mělkého minima (protože nerealizuje přesný grad. sestup)
- ▶ má menší paměťovou náročnost a snadno se implementuje
- ▶ je rychlejší, zejména na hodně redundantních datech

Nevýhody online (stochastického)

- ▶ není vhodný pro paralelizaci
- ▶ může hodně „kličkovat“ (více než gradientní sestup)

Problémy s gradientním sestupem:

- ▶ Může se stát, že gradient $\nabla E(\vec{w}^{(t)})$ neustále mění směr, ale chyba se postupně zmenšuje.

Toto často nastává, pokud jsme v mělkém údolí, které se mírně svažuje jedním směrem (něco jako U-rampa pro snowboarding, učící algoritmus potom opisuje dráhu snowboardisty)

- ▶ Online algoritmus také může zbytečně kličkovat.

Tento algoritmus se vůbec nemusí pohybovat ve směru největšího spádu!

Řešení: Ke změně vah dané gradientním sestupem přičteme část změny v předchozím kroku, tedy definujeme

$$\Delta w_{ji}^{(t)} = \Delta \vec{w}^{(t)} + \alpha \cdot \Delta w_{ji}^{(t-1)}$$

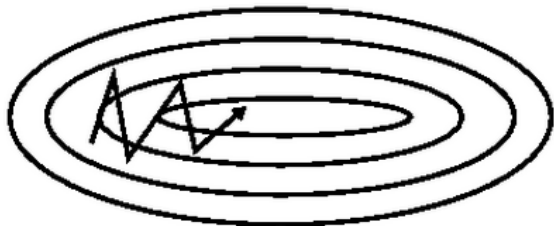
kde $0 < \alpha < 1$.

Moment - ilustrace

Bez momentu:



S momentem:



Volba aktivační funkce

- ▶ Z hlediska rychlosti konvergence je výhodné volit *lichou* aktivační funkci.
- ▶ Standardní sigmoida není lichá, vhodnější je použít hyperbolický tangens:

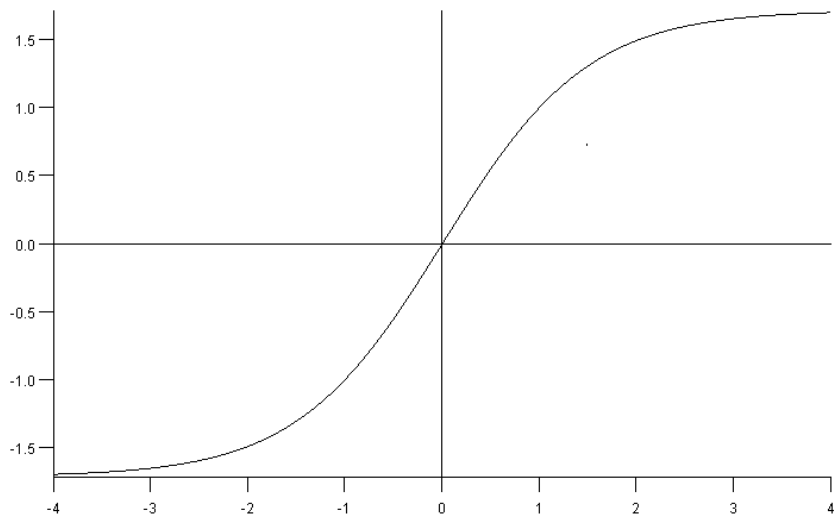
$$\sigma(\xi) = a \cdot \tanh(b \cdot \xi)$$

- ▶ Při optimalizaci lze předpokládat, že sigmoidální funkce jsou fixní a „hýbat“ pouze dalšími parametry.
- ▶ Z technických důvodů *budeme dále předpokládat, že všechny aktivační funkce jsou tvaru*

$$\sigma_j(\xi_j) = a \cdot \tanh(b \cdot \xi_j)$$

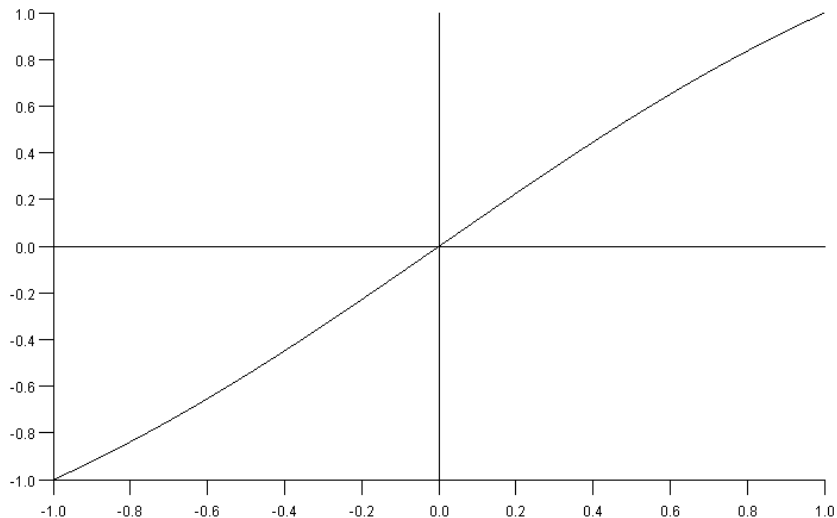
kde $a = 1.7159$ a $b = \frac{2}{3}$.

Volba aktivační funkce



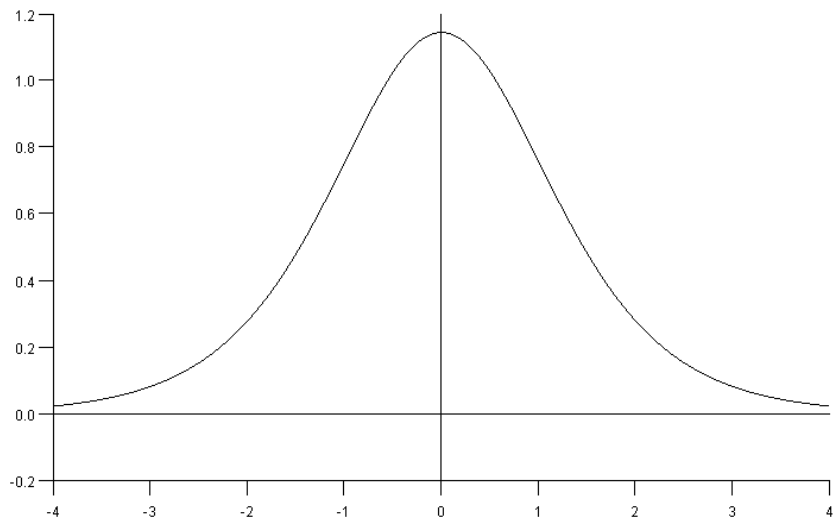
$\sigma(\xi) = 1.7159 \cdot \tanh\left(\frac{2}{3} \cdot \xi\right)$, platí $\lim_{\xi \rightarrow \infty} \sigma(\xi) = 1.7159$ a $\lim_{\xi \rightarrow -\infty} \sigma(\xi) = -1.7159$

Volba aktivací funkce



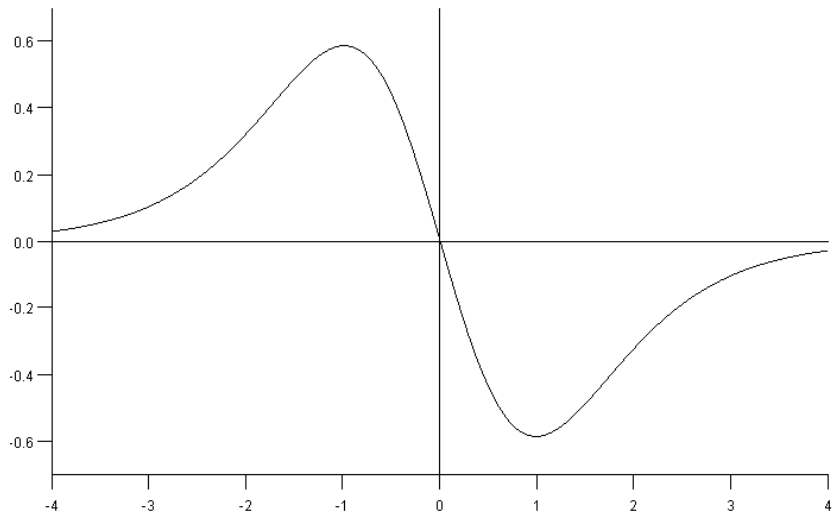
$\sigma(\xi) = 1.7159 \cdot \tanh\left(\frac{2}{3} \cdot \xi\right)$ je téměř lineární na $[-1, 1]$

Volba aktivální funkce



první derivace funkce $\sigma(\xi) = 1.7159 \cdot \tanh\left(\frac{2}{3} \cdot \xi\right)$

Volba aktivční funkce



druhá derivace funkce $\sigma(\xi) = 1.7159 \cdot \tanh\left(\frac{2}{3} \cdot \xi\right)$

Předzpracování vstupů

- ▶ Některé vstupy mohou být mnohem větší než jiné
Př.: Výška člověka vs délka chodidla (oboje v cm), maximální rychlost auta (v km/h) vs cena (v Kč), apod.
- ▶ Velké komponenty vstupů ovlivňují učení více, než ty malé. Navíc příliš velké vstupy mohou zpomalit učení.
- ▶ Numerická data se obvykle normalizují tak, že mají
 - ▶ průměrnou hodnotu = 0 (normalizace odečtením průměru)
 - ▶ rozptyl = 1 (normalizace dělením směrodatnou odchylkou)

Zde průměr a směrodatná odchylka mohou být odhadnuty z náhodného vzorku (např. z tréninkové množiny).

- ▶ Jednotlivé komponenty vstupu by měly mít co nejmenší korelaci (tj. vzájemnou závislost).

(Metody pro odstranění korelace dat jsou např. analýza hlavních komponent (Principal Component Analysis, PCA). Lze ji implementovat i pomocí neuronových sítí (probereme později)).

Iniciální volba vah

- ▶ Iniciálně jsou váhy nastaveny náhodně z daného intervalu $[-w, w]$ kde w závisí na počtu vstupů daného neuronu. Jaké je vhodné w ?
- ▶ Uvažujeme aktivační funkci $1.7159 \cdot \tanh(\frac{2}{3} \cdot \xi)$ pro všechny neurony.
 - ▶ na intervalu $[-1, 1]$ se σ chová téměř lineárně,
 - ▶ extrémy σ'' jsou přibližně v bodech -1 a 1 ,
 - ▶ mimo interval $[-4, 4]$ je σ blízko extrémních hodnot.

Tedy

- ▶ Pro velmi malé w hrozí, že obdržíme téměř lineární model (to bychom mohli dostat s použitím jednovrstvé sítě). Navíc chybová funkce je velmi plochá v blízkosti $\vec{0}$ (malý gradient).
- ▶ Pro w mnohem větší než 1 hrozí, že vnitřní potenciály budou vždy příliš velké a síť se nebude učit (gradient chyby E bude velmi malý, protože hodnota sítě se téměř nezmění se změnou vah).

Chceme tedy zvolit w tak, aby vnitřní potenciály byly zhruba z intervalu $[-1, 1]$.

Iniciální volba vah

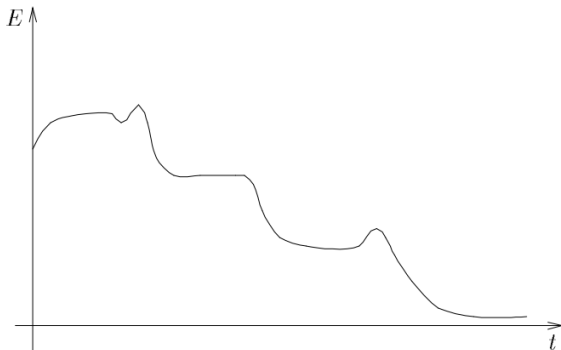
- ▶ Data mají po normalizaci (zhruba) nulovou střední hodnotu, rozptyl (zhruba) 1 a předpokládejme, že jednotlivé komponenty vstupů jsou (téměř) nekorelované.
- ▶ Uvažme neuron j z první vrstvy s d vstupy, předpokládejme, že jeho váhy jsou voleny uniformně z intervalu $[-w, w]$.
- ▶ **Pravidlo:** w je dobré volit tak, že *směrodatná odchylka* vnitřního potenciálu ξ_j (označme ji σ_j) leží na hranici intervalu, na němž je aktivační funkce σ_j téměř lineární tj. v našem případě chceme $\sigma_j \approx 1$.
- ▶ Z našich předpokladů plyne $\sigma_j = \sqrt{\frac{d}{3}} \cdot w$.
tj. v našem případě položíme $w = \frac{\sqrt{3}}{\sqrt{d}}$.
- ▶ Totéž funguje pro vyšší vrstvy, d potom odpovídá počtu neuronů ve vrstvě o jedna nižší.
(Zde je důležité, že je aktivační funkce lichá)

Požadované hodnoty

- ▶ Požadované hodnoty d_{kj} by měly být voleny v oboru hodnot aktivačních funkcí, což je v našem případě $[-1.716, 1.716]$.
- ▶ Požadované hodnoty příliš blízko extrémům ± 1.716 způsobí, že váhy neomezeně porostou, vnitřní neurony budou mít velké potenciály, gradient chybové funkce bude malý a učení se zpomalí.
- ▶ Proto je vhodné volit požadované hodnoty z intervalu $[-1.716 + \delta, 1.716 - \delta]$. Optimální je, když tento interval odpovídá maximálnímu intervalu na němž jsou aktivační fce lineární. Tedy v našem případě $\delta \approx 0.716$, tj. hodnoty d_{kj} je dobré brát z intervalu $[-1, 1]$.

Obečné zásady pro volbu a změny rychlosti učení ε

- ▶ Je dobré začít s malou rychlostí (např. $\varepsilon = 0.1$).
- ▶ Pokud se chyba zřetelně zmenšuje (učení konverguje), můžeme rychlost nepatrně zvýšit.
- ▶ Pokud se chyba zřetelně zvětšuje (učení diverguje), můžeme rychlost snížit.
- ▶ Krátkodobé zvýšení chyby nemusí nutně znamenat divergenci.



Obr. 2.3: Typický vývoj chyby v čase při učení pomocí backpropagation.

Chceme, aby se neurony učily pokud možno stejně rychle. Více vstupů obvykle znamená rychlejší učení.

Pro každou váhu w_{ji} můžeme mít zvláštní rychlost učení ε_{ji}

- ▶ ε_{ji} lze iniciovat např. $1/|j_{\leftarrow}|$, kde $|j_{\leftarrow}|$ je počet vstupů neuronu j .

- ▶ Po zahájení učení váhu zvolna zvyšujeme

(třeba $\varepsilon_{ji}(t) = K\varepsilon_{ji}(t-1)$ kde $K > 1$)

- ▶ Jakmile se změní znaménko $\Delta w_{ji}^{(t)}$ (tedy $\Delta w_{ji}^{(t-1)} \cdot \Delta w_{ji}^{(t)} < 0$), váhu snížíme

(třeba takto $\varepsilon_{ji}(t) = \frac{1}{2}\varepsilon_{ji}(t-1)$)

Rychlost a směr - přesněji

Na gradientní sestup se můžeme dívat obecněji takto:

$$\Delta \vec{w}^{(t)} = r(t) \cdot s(t)$$

kde $r(t)$ je rychlost změny vah a $s(t)$ je směr změny vah.

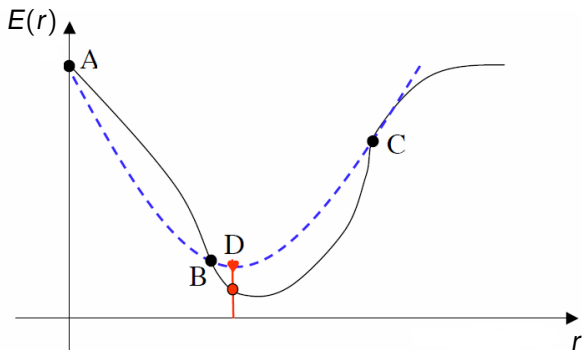
V našem případě: $r(t) = \varepsilon(t)$ a $s(t) = -\nabla E(\vec{w}^{(t-1)})$

(nebo $s(t) = -\nabla E_k(\vec{w}^{(t-1)})$ pro online učení).

- ▶ Ideální by bylo volit $r(t)$ tak, abychom minimalizovali $E(\vec{w}^{(t-1)} + r(t) \cdot s(t))$.
Nebo se aspoň chceme přesunout (ve směru $s(t)$) do místa, v němž začne gradient opět růst.
- ▶ Toho lze (přibližně) dosáhnout malými přesuny bez změny směru - nedostaneme ovšem o mnoho lepší algoritmus než standardní grad. sestup (který stále mění směr).
- ▶ Existují lepší metody, např. **parabolická interpolace** chybové funkce.

Rychlost a směr - parabolická interpolace

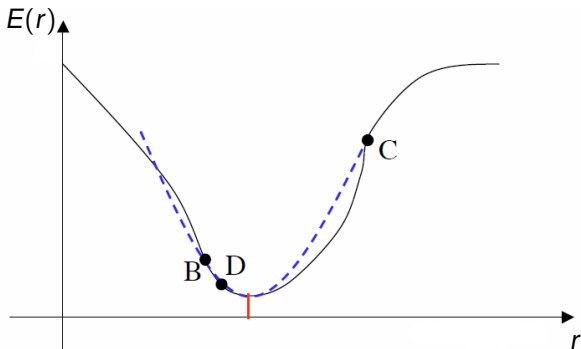
Označme $E(r) = E(\vec{w}^{(t-1)} + r \cdot s(t))$; minimalizujeme $E(r)$.



Předpokládejme, že jsme schopni nalézt body A, B, C takové, že $E(A) > E(B)$ a $E(C) > E(B)$. Pak lze tyto body proložit parabolou a nalézt její minimum D . Toto D je dobrým odhadem minima $E(r)$.

Rychlost a směr - parabolická interpolace

Parabolickou interpolaci lze dále iterovat, čímž dosáhneme ještě lepšího odhadu:



Je jasné, že $E(B) > E(D)$ a $E(C) > E(D)$, proto lze použít stejný postup jako předtím.

Optimální směr

Zbývá otázka, jestli je záporný gradient správným směrem.

Rychlost $r(t)$ jsme volili tak, abychom minimalizovali

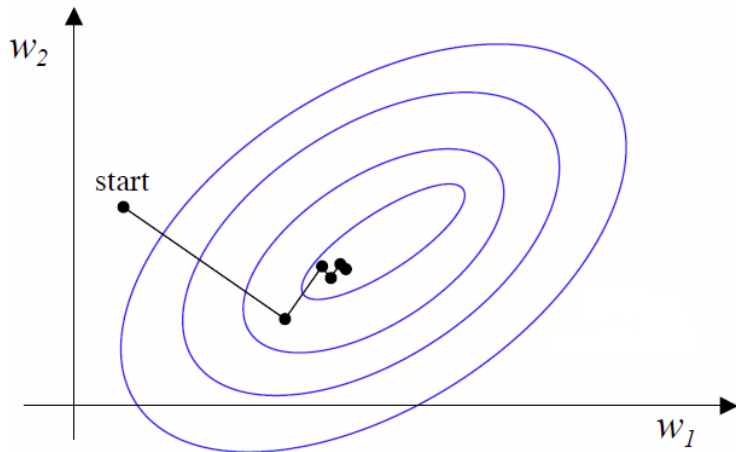
$$\begin{aligned} E(\vec{w}^{(t)}) &= E(\vec{w}^{(t-1)} + r(t) \cdot s(t)) \\ &= E(\vec{w}^{(t-1)} + r(t) \cdot (-\nabla E(\vec{w}^{(t-1)}))) \end{aligned}$$

To ovšem znamená, že derivace $E(\vec{w}^{(t)})$ podle $r(t)$ (zde bereme $r(t)$ jako nezávislou proměnnou) bude 0, tedy

$$\begin{aligned} \frac{\delta E}{\delta r}(\vec{w}^{(t)}) &= \sum_{j,i} \frac{\delta E}{\delta w_{ji}}(\vec{w}^{(t)}) \cdot \left(-\frac{\delta E}{\delta w_{ji}}(\vec{w}^{(t-1)}) \right) \\ &= \nabla E(\vec{w}^{(t)}) \cdot (-\nabla E(\vec{w}^{(t-1)})) \\ &= 0 \end{aligned}$$

Tj. nový a starý směr jsou vzájemně kolmé, výpočet tedy „kličkuje“.

Gradientní sestup s optimální rychlostí



Obrázek: Neural Computation, Dr John A. Bullinaria

Rychlost a směr - přesněji

Řešení: Do nového směru zahrneme částečně i předchozí směr a tím zmenšíme klíčování.

$$s(t) = -\nabla E(\vec{w}^{(t-1)}) + \beta \cdot s(t-1)$$

Jak určit β ? Metoda **sdružených gradientů** je založena na tom, že nový směr by měl co nejméně kazit minimalizaci dosaženou v předchozím směru. Chceme nalézt nový směr $s(t)$ takový, že gradient funkce E v novém bodě $\vec{w}^{(t)} = \vec{w}^{(t-1)} + r(t) \cdot s(t)$ ve starém směru $s(t-1)$ je nulový:

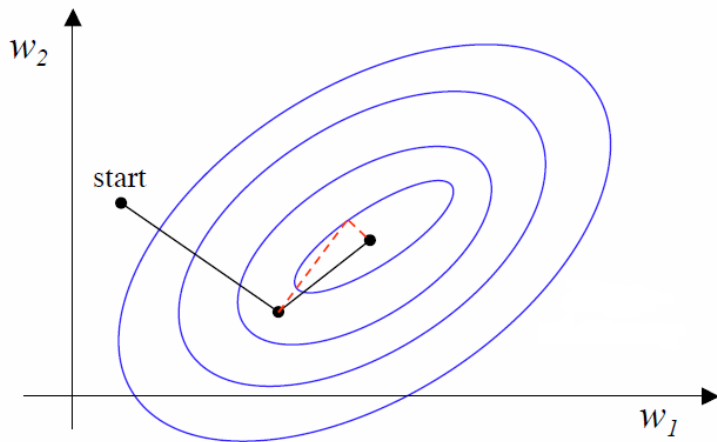
$$s(t-1) \cdot \nabla E(\vec{w}^{(t)}) = 0$$

Vhodné β , které to splňuje je dáno následujícím pravidlem (Polak-Ribiere):

$$\beta = \frac{(\nabla E(\vec{w}^{(t)}) - \nabla E(\vec{w}^{(t-1)})) \cdot \nabla E(\vec{w}^{(t)})}{\nabla E(\vec{w}^{(t-1)}) \cdot \nabla E(\vec{w}^{(t-1)})}$$

(Pokud by E byla kvadratická funkce, pak to konverguje v nejvýše n krocích)

Gradientní sestup s optimální rychlostí



Obrázek: Neural Computation, Dr John A. Bullinaria

Existuje mnoho metod druhého řádu, které jsou přesnější, ale obvykle výpočetně mnohem náročnější (příliš se nepoužívají). Např. Newtonova metoda, Levenberg-Marquardt, atd.

Většina těchto metod vyžaduje výpočet (nebo aspoň aproximaci) druhé derivace chybové funkce nebo funkce sítě (tj. Hessián).

Lze nalézt v literatuře, např.

Haykin, Neural Networks and Learning Machines

Schopnost generalizace

V klasifikačních problémech se dá generalizace popsat jako schopnost vyrovnat se s novými vzory.

Pokud síť trénujeme na náhodně vybraných datech, není ideální přesně klasifikovat tréninkové vzory.

Pokud aproximujeme funkční závislost vstupů a očekávaných výstupů pak obvykle nechceme aby funkce sítě vracela přesné hodnoty pro tréninkové vzory.

Exaktněji: Obvykle se předpokládá, že tréninková množina byla generována takto:

$$d_{kj} = g_j(\vec{x}_k) + \Theta_{kj}$$

kde g_j je „správná“ funkce výstupního neuronu $j \in Y$ a Θ_{kj} je náhodný šum. Chceme, aby síť pokud možno realizovala funkce g_j .

Kdy zastavit učení?

Standardní kritérium: Chyba E je dostatečně malá.

Další možnost: po několik iterací je gradient chyby malý.

(Výhodou tohoto kritéria je, že se nemusí počítat chyba E)

Problém: Příliš dlouhé učení způsobí, že síť „opisuje“ tréninkové vzory (je přetrénovaná). Důsledkem je špatná generalizace.

Řešení: Množinu vzorů rozdělíme do následujících množin

- ▶ **tréninková** (např. 60%) - podle těchto vzorů se síť učí
- ▶ **validační** (např. 20%) - používá se k zastavení učení.
- ▶ **testovací** (např. 20%) - používá se po skončení učení k testování přesnosti sítě, tedy srovnání několika natrénovaných sítí.

Trénink, testování, validace

Obvykle se realizuje několik iterací (cca 5) tréninku na tréninkové množině. Poté se vyhodnotí chyba E na validační množině.

Ideálně chceme zastavit v minimu chyby na validační množině. V praxi se sleduje, zda chyba na validační množině klesá. Jakmile začne růst (nebo roste po několik iterací za sebou), učení zastavíme.

Problém: Co když máme příliš málo vzorů?

Můžeme tréninkovou množinu rozdělit na K skupin S_1, \dots, S_K . Trénujeme v K fázích, v ℓ -té fázi provedeme následující:

- ▶ trénujeme na $S_1 \cup \dots \cup S_{\ell-1} \cup S_{\ell+1} \cup \dots \cup S_K$
- ▶ spočteme chybu e_ℓ funkce E na skupině S_ℓ

Celková chyba je potom průměr $e = \frac{1}{K} \sum_{\ell=1}^K e_\ell$.

Extrémní verze: $K =$ počet vzorů (používá se při extrémně málo vzorech)

Podobný problém jako v případě délky učení:

- ▶ Příliš malá síť není schopna dostatečně zachytit tréninkovou množinu a bude mít stále velkou chybu
- ▶ Příliš velká síť má tendenci přesně opsat tréninkové vzory - špatná generalizace

Řešení: Optimální počet neuronů :-)

- ▶ teoretické výsledky existují, ale jsou většinou nepoužitelné
- ▶ existují učící algoritmy, které postupně přidávají neurony (konstruktivní algoritmy) nebo odstraňují spoje a neurony (prořezávání)
- ▶ V praxi se počet vrstev a neuronů stanovuje experimentálně (prostě se to zkusí, uvidí, opraví) a/nebo na základě zkušeností.

Příliš malá nebo nereprezentativní tréninková množina vede ke špatné generalizaci

Příliš velká zvyšuje složitost učení. V případě dávkového algoritmu způsobuje velkou chybu (chyby na vzorech se sčítají), což může vést k přetrénování.

Pravidlo pro klasifikační úlohy: počet vzorů by měl zhruba odpovídat W/δ kde

- ▶ W je počet vah v síti
- ▶ $0 < \delta < 1$ je tolerovaná chyba na testovací množině (tj. tolerujeme δ chybně klasifikovaných vzorů z testovací množiny)

Regularizace - upadání vah (weight decay)

Generalizaci lze zlepšit tak, že v síti necháme jen „nutné“ neurony a spoje. Možná heuristika spočívá v odstranění malých vah. Penalizací velkých vah dostaneme silnější indikaci důležitosti vah.

V každém kroku učení zmenšíme uměle všechny váhy

$$w_{ji}^{(m)} = (1 - \xi)(w_{ji}^{(m-1)} + \Delta w_{ji}^{(m)})$$

Idea: Nedůležité váhy budou velmi rychle klesat k 0 (potom je můžeme vyhodit). Důležité váhy dokážou „přetlačit“ klesání a zůstanou dostatečně velké.

Toto je ekvivalentní gradientnímu sestupu s konstantní rychlostí učení ε , pokud chybovou funkci modifikujeme takto:

$$E'(\vec{w}) = E(\vec{w}) + \frac{2\xi}{\varepsilon}(\vec{w} \cdot \vec{w})$$

Zde **regularizační člen** $\frac{2\xi}{\varepsilon}(\vec{w} \cdot \vec{w})$ penalizuje velké váhy.

Praxe - Matice zmatenosti

Confusion matrix

Síť má za úkol klasifikovat objekty do K tříd T_1, \dots, T_K .
Confusion matrix je tabulka, jejíž pole v i -tém řádku a j -tém sloupci obsahuje počet objektů z třídy T_i , které byly klasifikovány jako objekty z třídy T_j .

Example confusion matrix

		Predicted		
		Cat	Dog	Rabbit
Actual	Cat	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11

Zdroj: http://en.wikipedia.org/wiki/Confusion_matrix