Neuronové sítě 1 - Neurony se spojitou přenosovou funkcí

18NES1 - 5. - 6. hodina, LS 2024/25

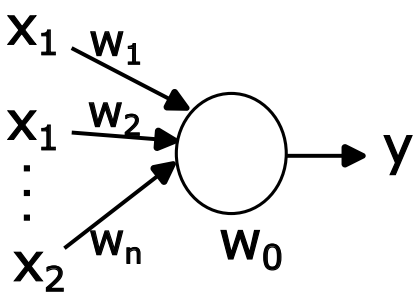
Zuzana Petříčková

16. dubna 2025

Co jsme probírali minule

Perceptron se skokovou přenosovou funkcí a jeho algoritmy učení

- vnitřní potenciál:
  $\xi = \sum_{i=1}^{n} w_i.x_i + w_0$
- výstup: $y = f(\xi)$



- algoritmy učení:
  - Rosenblattovo učení a jeho různé varianty
  - Hebbovo učení

- **skoková** přenosová funkce:

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

Příklady - Dokončení

Příklad 4 - Náhodně generovaná data

- Pokud porovnáváme různé modely nebo algoritmy, je dobré začít s uměle, např. náhodně, generovanými daty
- V tomto případě data nejsou zcela lineárně separabilní (je přidán náhodný šum)
- Generujte data opakovaně (různě) a pozorujte, jak se perceptrony učí. Která varianta Rosenblattova algoritmu je podle vás pro tuto úlohu nejlepší?

Příklad 5 - Náhodně generované shluky

- Potom vygenerujeme data obsahující dva shluky vzorů
- Generujte data opakovaně (různě) a pozorujte, jak se perceptrony učí. Která varianta Rosenblattova algoritmu je podle vás pro tuto úlohu nejlepší?

Příklady - Různé praktické úlohy - Dokončení

Příklad 5 - Písmena letters_example.ipynb

- Využijeme připravenou datovou sadu **letters.csv**
- Písmena byla segmentována z **letters.png**
- Prohlédněte si datovou sadu a zobrazte si některá písmenka
- Vytvoříme testovací množinu: data s přidaným šumem nebo následně vyhlazená
- Naučte perceptron pomocí různých algoritmů (a variant) rozpoznávat jednotlivá písmena.
- Určete chybu klasifikace na trénovací množině i na testovacích množinách (popř. i počet epoch / čas učení)
- S jak moc velkým šumem v datech si ještě perceptron poradil?
- Podívejte se, se kterými písmenky měl perceptron největší problémy.
- Který učící algoritmus si vedl nejlépe?

Příklady - Různé praktické úlohy - Dokončení

Příklad 6 - Ručně psané číslice digits_example.ipynb

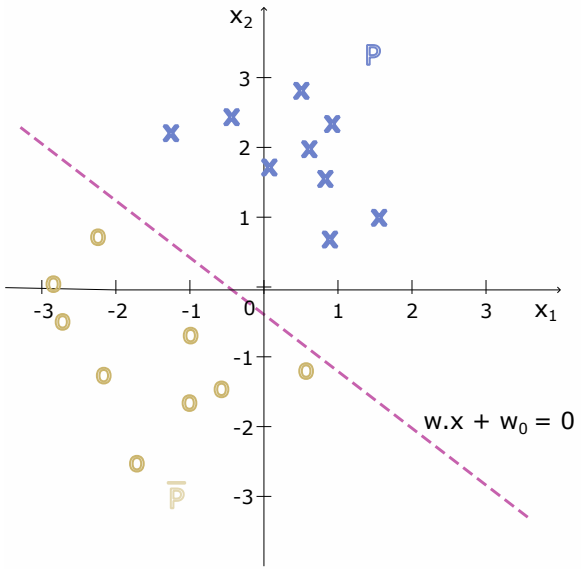
- Využijeme připravenou datovou sadu **OcrData.csv** s ručně psanými číslicemi
- Prohlédněte si datovou sadu a zobrazte si některé číslice (využijte předpřipravený skript)
- Naučte perceptron pomocí různých algoritmů (a variant) rozpoznávat jednotlivé číslice.
- Určete (a porovnejte) chybu klasifikace na trénovací množině (popř. i počet epoch / čas učení)
- Podívejte se, se kterými číslicemi měl perceptron největší problémy
- Který učící algoritmus si vedl nejlépe?

Perceptron se skokovou přenosovou funkcí

Aplikace:

- Lineární klasifikátor do dvou tříd
- Realizace logických funkcí

Problém: pokud data nejsou lineárně separabilní (např. XOR)
Co s tím?



1. kvadratické nebo kubické rozšíření příznakového prostoru např. $x_1, x_2, x_1^2, x_2^2, x_1 x_2$
2. neuronová síť s větším počtem perceptronů a vrstev ... jak ji naučit? :(

→ Co takhle použít místo skokové funkce nějakou **spojitou** přenosovou funkci?
→ Umožní nám to řešit i jiné typy úloh (např. regresní)

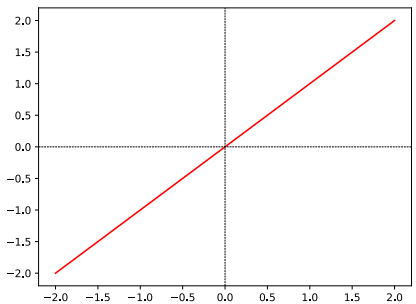
Dnešní hodina

1. **Lineární neuron a úloha lineární regrese**
   - Učení lineárního neuronu metodou LSQ
   - Učení lineárního neuronu gradientní metodou

Lineární neuron

- jeden z nejstarších modelů: ADALINE (Adaptive Linear Element, 1960, Widrow-Hoff)

- přenosová funkce identita: $f(\xi) = \xi$
- výstup neuronu:
  $y = \xi = \sum_{i=1}^{n} w_i.x_i + w_0 = \vec{x}\vec{w} + w_0$
  ($\vec{w}$ je sloupcový vektor)



Cíl učení:

- máme trénovací množinu ve tvaru $T = (X, \vec{d})$

x_{11}	...	x_{1n}	d_1
...		...	...
x_{N1}	...	x_{Nn}	d_N

- výstup neuronu maticově: $\vec{y} = X\vec{w} + w_0$

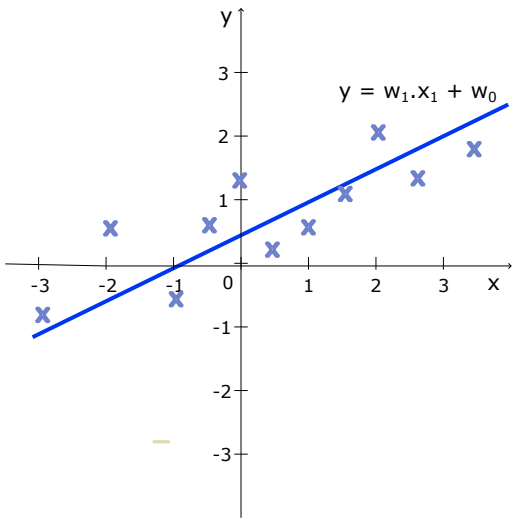
$\rightarrow$ hledáme $\vec{w}$, aby ideálně platilo: $\vec{d} = \vec{y}$, tj. $\vec{d} = X\vec{w} + w_0$

- jedná se o úlohu **lineární regrese**

Lineární neuron - geometrická interpretace

pro jeden vstupní příznak:

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



obecně:

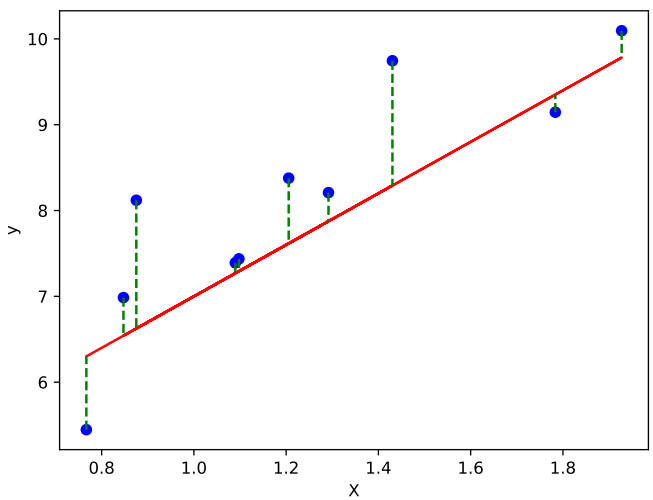
- prokládáme body nadrovinou $y = w_1 x_1 + \cdots + w_n x_n + w_0$ (předpokládáme, že mezi vstupními veličinami $x_1, \ldots, x_n$ a výstupní y je lineární závislost)

Jak učit lineární neuron (tj. model lineární regrese)?

- máme trénovací vzor: $(\vec{x}_p, d_p)$
- spočteme skutečný výstup neuronu: $y_p = \vec{x}_p \vec{w} + w_0$
- skutečný a požadovaný výstup se liší:

$$y_p = d_p + e_p \ \ldots \ e_p \text{ je chyba pro jeden vstupní vzor}$$

- chceme aby se skutečný a požadovaný výstup pro každý trénovací vzor co nejméně lišily:



Jak učit lineární neuron (tj. model lineární regrese)?

Jak definujeme chybovou funkci, kterou budeme během učení minimalizovat?

1. SAE (součet absolutních chyb)

$$E = \sum_p |e_p| = \sum_p |d_p - y_p|$$

   - **nevýhoda:** špatně se optimalizuje (není spojitě derivovatelná)

2. SSE/SSQ (součet čtverců chyb) ... **metoda nejmenších čtverců**

$$E = \frac{1}{2} \sum_p e_p^2 = \frac{1}{2} \sum_p (d_p - y_p)^2$$

   - je snazší ji minimalizovat (je spojitě diferencovatelná)
   - odpovídá maximálně věrohodnému odhadu (MLE) za předpokladu gaussovského šumu
   - penalizuje více velké odchylky než malé $\rightarrow$ citlivá k odlehlým vzorům

Jak učit lineární neuron (tj. model lineární regrese)?

Metoda nejmenších čtverců

- minimalizujeme

$$E = \frac{1}{2} \sum_p (d_p - y_p)^2$$

Jak na to?

1. metoda LSQ - založena na explicitním výpočtu
2. gradientní metoda (metoda největšího spádu)

Lineární neuron - učení metodou LSQ

- máme rozšířenou trénovací množinu ve tvaru $T = (X, \vec{d})$

$x_{10} = 1$	x_{11}	...	x_{1n}	d_1
...	...		...	...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_N

$\rightarrow$ hledáme $\vec{w}$, aby platilo: $\vec{d} = \vec{y}$, tj. $\vec{d} = X\vec{w}$

Řešíme tedy soustavu rovnic:

$$\begin{array}{ccccccccc} w_0x_{10} & + & w_1x_{11} & + & \ldots & + & w_nx_{1n} & = & d_1 \\ \ldots & & \ldots & & \ldots & & \ldots & & \ldots \\ w_0x_{N0} & + & w_1x_{N1} & + & \ldots & + & w_nx_{Nn} & = & d_N \end{array}$$

Lineární neuron - učení metodou LSQ

Kdy má soustava právě jedno řešení?

- podmínka: sloupce X musí být lineárně nezávislé, tj. $h(x) = n + 1$
- podmínka na hodnost: $h(X|\vec{d}) = h(X)$

Obecně

- soustava rovnic může mít nekonečně mnoho řešení (nebo žádné)
- $\frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2 = min$, tj. $||X\vec{w} - \vec{d}||^2 = min$
  $\rightarrow$ položíme derivaci této funkce rovnou nule a po troše počítání dostaneme: $(X^T X)\vec{w} - X^T\vec{d} = 0$

Alternativní odvození (Gauss):

$$X\vec{w} = \vec{d}$$

$$X^T(X\vec{w}) = X^T\vec{d}$$

$$(X^T X)\vec{w} = X^T\vec{d}$$

Lineární neuron - učení metodou LSQ

$$(X^T X)\vec{w} = X^T \vec{d}$$

1. pokud existuje inverzní matice, tj. $det(X^T X) \neq 0$:
$$\vec{w} = (X^T X)^{-1} X^T \vec{d}$$
2. pokud $det(X^T X) = 0$ (soustava má nekonečně mnoho nebo žádné řešení):
$\rightarrow$ provedeme regularizaci (s využitím pseudoinverzní matice):
   1. Tichonovova regularizace (ridge regression)
   $$\vec{w} = (X^T X + \lambda I)^{-1} X^T \vec{d}, \lambda > 0$$
   2. Moore-Penroseova pseudoinverze ... řešení s co nejmenšími vahami:
   $$\vec{w} = \lim_{\lambda \to 0+} (X^T X + \lambda I)^{-1} X^T \vec{d}$$

Lineární neuron - učení metodou LSQ

Příklad 1 ... $\vec{w} = (X^T X)^{-1} X^T \vec{d}$

x_0	x_1	x_2	d
+1	-1	-1	+1
+1	-1	+1	+1
+1	+1	-1	+1
+1	+1	+1	-1

$$X^T X = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

$$(X^T X)^{-1} = \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix}$$

Lineární neuron - učení metodou LSQ

Příklad 1 ... $\vec{w} = (X^T X)^{-1} X^T \vec{d}$

x_0	x_1	x_2	d
+1	-1	-1	+1
+1	-1	+1	+1
+1	+1	-1	+1
+ 1	+1	+1	-1

$$\vec{w} = \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}$$

Lineární neuron - učení metodou LSQ

Příklad 2 ... $\vec{w} = (X^T X)^{-1} X^T \vec{d}$

x_0	x_1	x_2	d
+1	+1	-1	+1
+1	+1	+1	-1

$$X^T X = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

$$h(X^T X) = 2 \rightarrow det(X^T X) = 0 \rightarrow (X^T X)^{-1} neexistuje$$

Provedeme regularizaci:

$$\vec{w} = (X^T X + \lambda I)^{-1} X^T \vec{d}, \lambda > 0$$

$$\vec{w} = \lim_{\lambda \to 0+} (X^T X + \lambda I)^{-1} X^T \vec{d}$$

Lineární neuron - učení metodou LSQ

Příklad 2 ... $\vec{w} = (X^T X + \lambda I)^{-1} X^T \vec{d}, \lambda > 0$

x_0	x_1	x_2	d
+1	+1	-1	+1
+1	+1	+1	-1

$$X^T X + \lambda I = \begin{pmatrix} 2+\lambda & 2 & 0 \\ 2 & 2+\lambda & 0 \\ 0 & 0 & 2+\lambda \end{pmatrix}$$

Po větší troše počítání:

$$(X^T X + \lambda I)^{-1} = \begin{pmatrix} \frac{2+\lambda}{\lambda^2+4\lambda} & -\frac{2}{\lambda^2+4\lambda} & 0 \\ -\frac{2}{\lambda^2+4\lambda} & \frac{2+\lambda}{\lambda^2+4\lambda} & 0 \\ 0 & 0 & \frac{1}{2+\lambda} \end{pmatrix}$$

Lineární neuron - učení metodou LSQ

Příklad 2 ... $\vec{w} = (X^T X + \lambda I)^{-1} X^T \vec{d} = X^+ \vec{d}, \lambda > 0$

$$X^+ = (X^T X + \lambda I)^{-1} X^T = \begin{pmatrix} \frac{2+\lambda}{\lambda^2+4\lambda} & -\frac{2}{\lambda^2+4\lambda} & 0 \\ -\frac{2}{\lambda^2+4\lambda} & \frac{2+\lambda}{\lambda^2+4\lambda} & 0 \\ 0 & 0 & \frac{1}{2+\lambda} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\lambda}{\lambda^2+4\lambda} & \frac{\lambda}{\lambda^2+4\lambda} \\ \frac{\lambda}{\lambda^2+4\lambda} & \frac{\lambda}{\lambda^2+4\lambda} \\ -\frac{1}{2+\lambda} & \frac{1}{2+\lambda} \end{pmatrix} = \begin{pmatrix} \frac{1}{\lambda+4} & \frac{1}{\lambda+4} \\ \frac{1}{\lambda+4} & \frac{1}{\lambda+4} \\ -\frac{1}{2+\lambda} & \frac{1}{2+\lambda} \end{pmatrix}$$

Lineární neuron - učení metodou LSQ

Příklad 2 ... $\vec{w} = (X^T X + \lambda I)^{-1} X^T \vec{d} = X^+ \vec{d}, \lambda > 0$

- $\lambda = 1$:

$$\vec{w} = X^+ \vec{d} = \begin{pmatrix} \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} \\ -\frac{1}{3} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -\frac{2}{3} \end{pmatrix}$$

- $\lambda = \frac{1}{10}$:

$$\vec{w} = X^+ \vec{d} = \begin{pmatrix} \frac{10}{41} & \frac{10}{41} \\ \frac{10}{41} & \frac{10}{41} \\ -\frac{10}{21} & \frac{10}{21} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -\frac{20}{21} \end{pmatrix}$$

- $\lambda \to 0$:

$$\vec{w} = X^+ \vec{d} = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

Příklady - Ukázky učících algoritmů LSQ v Pythonu

linear_neuron.ipynb

- Metoda LSQ na Příkladech 1 a 2 ze slidů
- Tři různé implementace algoritmu:
  - **S využitím knihovní funkce** (standardní lineární regrese, LSQ)
  - Vlastní implementace **s využitím pseudoinverzní matice** (Moore-Penroseova pseudoinverze, zhruba ekvivalentní knihovní verzi)
  - Vlastní implementace **s využitím Tichonovovy regularizace** (hrubší aproximace, ale poradí si i s ,,horšími" případy)
- Pozorování: lineární neuron není příliš vhodný pro řešení klasifikačních úloh

Příklady - Úloha lineární regrese

Příklad 3 a Příklad 4 - úloha lineární regrese v jednorozměrném a dvourozměrném vstupním prostoru

- Uměle generovaná data: trénovací vzory generujeme na základě známé funkce a přidáme k nim náhodný šum
- Můžeme tak snadno pohledem na naučené váhy poznat, zda se neuron naučil danou úlohu nebo ne
- Můžeme experimentovat s množstvím šumu v trénovací množině
- Zobrazíme si výslednou regresní přímku/rovinu
- Otázky: Jak moc se naučené váhy a bias blíží skutečným hodnotám? Podívejte se i na hodnoty chyb (MSE a SSE).

Shrnutí metody LSQ

Výhody:

- Poskytuje přesné analytické řešení, pokud existuje inverze $X^T X$
- Výpočetně efektivní pro malé datasety (přímá inverze matice)
- Funguje dobře, pokud je výstup lineárně závislý na vstupech
- Lze rozšířit o regularizační techniky (např. Moore-Penroseova pseudoinverze, Tichonovovu regularizaci)

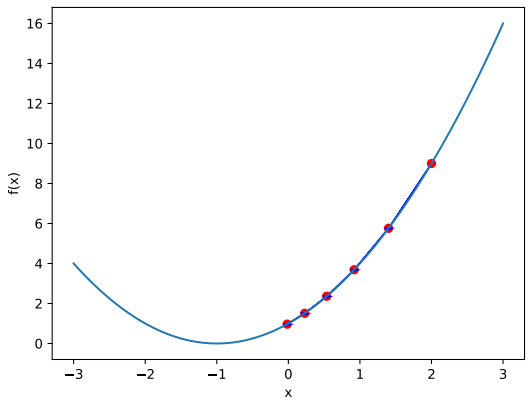
Nevýhody:

- Citlivost na šum a odlehlé hodnoty v datech
- Výpočetně náročná pro velké datasety (inverze velkých matic je nákladná)
- Regularizace je nutná v případech, kdy je $X^T X$ singulární
- Nevhodná pro klasifikační úlohy

Odbočka: gradientní metoda (metoda největšího spádu)

Úloha:

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



$\rightarrow$ **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 vstupní příznak: $x_i(t+1) = x_i(t) - \alpha \frac{\partial f}{\partial x_i}$

Odbočka: gradientní metoda (metoda největšího spádu)

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?

- začneme s $1 \gg \alpha_0 > 0$ a postupně ho během učení zmenšujeme
- stanovíme rozumnou rychlost poklesu: $\sum_{i=0}^{\infty} \alpha_i = \infty$, $\sum_{i=0}^{\infty} \alpha_i^2 < \infty$
- různé heuristiky: např. $\alpha_j = \frac{\alpha_0}{1+j}$ (Robbins-Monro, 1951)

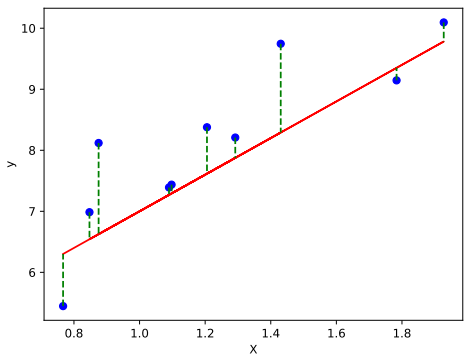
Učení lineárního neuronu gradientní metodou

Připomenutí: lineární neuron (model lineární regrese)

- výstup neuronu: $y = \xi = \sum_{i=0}^{n} w_i.x_i = \vec{x}\vec{w}$
- maticově: $\vec{y} = X\vec{w}$ ($\vec{w}$ je sloupcový vektor)

Metoda nejmenších čtverců

- chceme aby se skutečný výstup neuronu y_p se co nejméně lišil od požadovaného d_p



- minimalizujeme součet čtverců (SSE):

$$E = \frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2$$

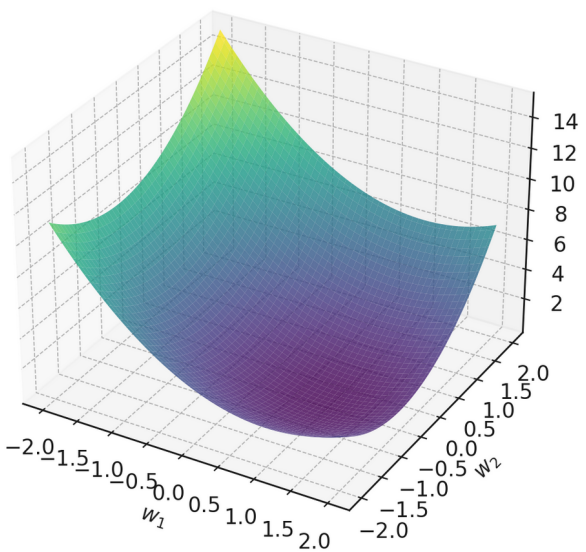
Učení lineárního neuronu gradientní metodou

Řešení gradientní metodou

- budeme minimalizovat chybovou funkci SSE v prostoru vah:

$$E(\vec{w}) = \frac{1}{2}\sum_{p=1}^{N}(d_p - y_p)^2 = \frac{1}{2}\sum_{p=1}^{N}\left(d_p - \sum_{i=0}^{n} w_i.x_{pi}\right)^2 = \sum_{p=1}^{N} E_p(\vec{w})$$

- $E_p(\vec{w})$ je chybová funkce pro jeden vzor



- jedná se o kvadratickou, tedy konvexní funkci, proto by gradientní metoda při správném nastavení parametrů neměla mít problém najít její globální minimum

Učení lineárního neuronu gradientní metodou

1. Nejprve pro chybovou funkci

$$E_p(\vec{w}) = \frac{1}{2}(d_p - y_p)^2 = \frac{1}{2}\left(d_p - f(\sum_{i=0}^{n} w_i.x_{pi})\right)^2$$

spočteme její parciální derivace:

$$\frac{\partial E_p}{\partial w_i} = \frac{\partial E_p}{\partial y_p}\frac{\partial y_p}{\partial w_i} = -(d_p - y_p)x_{pi}$$

2. Potom sestavíme adaptační pravidlo:

$$w_i(t+1) = w_i(t) - \alpha\frac{\partial E_p}{\partial w_i} = w_i(t) + \alpha(d_p - y_p)x_{pi}$$

pro vektor vah:

$$\vec{w}(t+1) = \vec{w}(t) - \alpha\nabla E_p(\vec{w}) = \vec{w}(t) + \alpha(d_p - y_p)\vec{x}_p^T$$

Učení lineárního neuronu gradientní metodou

Obecné schéma algoritmu (GD, gradient descent)

1. Inicializuj váhy malými náhodnými reálnými hodnotami .... $\vec{w}(0) = (w_0, w_1, ..., w_n)^T$
   Inicializuj parametr učení α_0.... $1 \gg \alpha_0 > 0$
2. Předlož další trénovací vzor $(\vec{x}_t, d_t)$ a spočti skutečný výstup neuronu: $y_t = \vec{x}_t \vec{w}$
3. Adaptuj váhy:

$$\vec{w}(t+1) = \vec{w}(t) + \alpha_t(d_t - y_t)\vec{x}_t^T$$

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

Učení lineárního neuronu gradientní metodou

Jak předkládat trénovací vzory? .... různé strategie:

1. **iterativně po epochách (Online GD)**:
   - během jedné epochy se každý vzor předloží právě jednou, v rámci každé epochy vzory náhodně uspořádáme
   - počet epoch .... kolikrát se předloží celá trénovací množina
2. **dávkově po epochách (Batch GD)**:
   - celá trénovací množina se předloží najednou a váhy se adaptují také najednou

$$\vec{y} = X\vec{w}$$

$$\vec{w}(t+1) = \vec{w}(t) + \alpha_t \sum_{p=1}^{N} (d_p - y_p)\vec{x}_p^T = \vec{w}(t) + \alpha_t X^T(\vec{d} - \vec{y})$$

3. **dávkově po minibatchích (SGD, stochastic gradient descent)**
   - trénovací množina se náhodně rozdělí na dávky (minibatch) a dávky se jedna po druhé předloží se dávkově

Učení lineárního neuronu gradientní metodou

Jak předkládat trénovací vzory? .... různé strategie:

1. **iterativně po epochách (Online GD)**:
   - rychlé učení (v počtu epoch), ale poměrně nestabilní (algoritmus snižuje chybu pro aktuální vzor $\rightarrow$ chyba se může zvýšit pro ostatní vzory)
   - větší náhodnost, větší citlivost na odlehlé vzory a na volbu hyperparametrů (parametr učení ap.)
2. **dávkově po epochách (Batch GD)**:
   - stabilní algoritmus
   - efektivní pro malá data, větší prostorové nároky pro velká data
3. **dávkově po minibatchích (SGD, stochastic gradient descent)**
   - spojuje výhody obou metod
   - uplatní se spíše pro hluboké neuronové sítě a velké datové sady

Učení lineárního neuronu gradientní metodou

Jak inicializovat váhy?

- učení by mělo začít v náhodném bodě
- proto váhy inicializuje na malé náhodné hodnoty (ideálně se střední hodnotou 0) raději, než přímo na nulu
- příliš velké nebo jedním směrem vychýlené váhy povedou k tomu, že se lineární neuron bude učit obtížně

Konstantní či adaptivní parametr učení?

- pokus parametr učení zvolíme konstantní, algoritmus může na konci učení začít oscilovat
- algoritmus je velmi citlivý na volbu parametru učení

Jak a jak často aktualizovat parametr učení?

- typicky jednou za epochu e:

$$\alpha_e = \frac{\alpha_0}{e}, \alpha_e = \frac{\alpha_0}{\sqrt{e}}$$

Učení lineárního neuronu gradientní metodou

Kdy ukončit učení?

- velmi důležitá otázka
- Uplatňují se různé strategie:
  1. předem daný počet epoch
  2. jakmile je průměrná chyba dostatečně malá ... $E < E_{min}$
  3. jakmile přestane klesat chyba na validační množině dat ... **early stopping**
  4. jakmile je přírůstek vah Δw moc malý ... $|\Delta w| < \delta_{min}$

Early stopping

- využívá pomocnou datovou sadu - **validační množina**
  - měla by být zcela nezávislá na trénovací množině dat
  - umožňuje nám průběžně sledovat, jak dobře model zobecňuje
- pokud chyba na validační množině dat několik epoch za sebou roste, učení ukončíme

Učení lineárního neuronu gradientní metodou

Co když jsou vstupní vzory „velké”?

- to by mohlo způsobovat velké problémy při učení (ovlivní to rychlost a stabilitu učení, zobecňování,..)

Řešení: normalizace hodnot vstupních příznaků

- min-max normalizace na interval $[-1, 1]$:

$$X_{ij}^{new} = 2 * \frac{X_{ij} - m_j}{M_j - m_j} - 1$$

$m_j = \min_k(X_{kj})$, $M_j = \max_k(X_{kj})$

- normalizace podle směrodatné odchylky:

$$X_{ij}^{new} = \frac{X_{ij} - E(X_{kj})}{S(X_{kj})}$$

  - $E(X_{kj}) = \frac{1}{N} \sum_{k=1}^{N} X_{kj}$ je průměr (střední hodnota) sloupce j matice X
  - $S(X_{kj}) = \frac{1}{N-1} \sum_{k=1}^{N} (X_{kj} - E(X_{kj}))^2$ je směrodatná odchylka sloupce j matice X

Učení lineárního neuronu gradientní metodou

Zhodnocení, zda se neuron dobře naučil danou regresní úlohu

- různé chybové funkce:

- SAE (Sum absolute error):
  $E(\vec{w}) = \sum_{p=1}^{N} |d_p - y_p|$
- SSE (Sum squared error):
  $E(\vec{w}) = \sum_{p=1}^{N} (d_p - y_p)^2$
- MAE (Mean absolute error):
  ... nejlépe čitelná pro člověka, průměrná odchylka skutečného a požadovaného výstupu
  $E(\vec{w}) = \frac{1}{N} \sum_{p=1}^{N} |d_p - y_p|$
- MSE (Mean squared error):
  ... nejlepší pro porovnání, průměrná střední kvadratická odchylka
  $E(\vec{w}) = \frac{1}{N} \sum_{p=1}^{N} (d_p - y_p)^2$

Učení lineárního neuronu gradientní metodou

Zhodnocení, zda se neuron dobře naučil danou regresní úlohu

- pro regresní úlohu spočteme MSE a SSE na trénovací množině dat
- pro zhodnocení, jak dobře model zobecňuje, spočteme MSE a SSE i na **testovací množině dat**
  - měla by být zcela nezávislá na trénovací i validační množině dat (měla by obsahovat zcela jiné vzory)

Jak vytvořit validační a testovací množinu?

- pro umělé úlohy je náhodně vygenerujeme (např. ze stejného pravděpodobnostního rozdělení, popř. přidáme další šum)
- pro úlohy nad reálnými daty je zvykem data před učením náhodně rozdělit na trénovací, validační a testovací podmnožiny, např. v poměru 70-15-15.

Příklady - Ukázky gradientní metody v Pythonu

linear_neuron.ipynb
Gradientní metoda na Příkladech 1 a 2 ze slidů

- Ukážeme si, jak zásaní je pro gradientní metodu správná volba hyperparametrů (parametr učení, to, zda je adaptivní, nebo ne, vhodná inicializace vah).
- Srovnání gradientní metody s metodou LSQ
  - Pozorovaní: gradientní metoda je náročnější na ladění hyperparametrů a pro malé úlohy je i časově náročnější
  - Pro složitější úlohy nicméně gradientní metoda může dát lepší výsledky než LSQ
- Ukážeme si, jak postupně můžeme vyladit hyperparametry při řešení konkrétní úlohy.

Příklady - Úloha lineární regrese

Příklad 3 a Příklad 4 - úloha lineární regrese v jednorozměrném a dvourozměrném vstupním prostoru

- Ukážeme si využití testovací a validační množiny dat během učení a při zhodnocení toho, jak dobře se lineární neron úlohu naučil
- Opět budeme expeimentovat s natavením hyperparametrů a budeme se je snažit vyladit pro danou úlohu
- Porovnáme mezi sebou iterativní a dávkový algoritmus
- Vyzkoušíme si techniku early stopping

Příklad - dobrovolný domácí úkol za bonusové body

- Modifikujte data 4 (definujte si vlastní, unikátní lineární funkci s unikátním šumem).
- Experimentujte pro tuto úlohu s nastavením hyperparametrů (parametr učení, počet epoch, učící strategie,...) a snažte se je co nejlépe vyladit.
- Stručně zhodnoťte výsledky experimentu (jaké nastavení hyperparametrů byste pro danou úlohu doporučili a proč)
- Srovnejte nejlepší dosažené výsledky s metodou LSQ
- Výsledný notebook (včetně textového zhodnocení) mi můžete poslat na email

Gradientní metoda - shrnutí

- gradientní metoda je schopná řešit úlohu lineární regrese stejně dobře jako klasická LSQ metoda, **ALE** je náročnější na aplikaci:
  - je to lokální metoda: průběh i výsledek bude pokaždé trochu jiný
  - metoda je velmi citlivá na správné nastavení různých hyperparametrů
- výhodou oproti LSQ je, že ji můžeme aplikovat na výrazně širší okruh úloh, kde si klasická LSQ metoda neporadí