Mesterséges Neuronhálók, 2015-16 ősz jegyzet

Lengyel Mihály

2015. november 9.

1. fejezet

2015.09.14

A Piazzán feltett kérdések szerepelhetnek zh-n, valamint megkért mindenkit, hogy legalább néhányan készítsünk jegyzetet.

. A zh-k nem lesznek előre bejelentve, a kérdésekre ott a piazza.

. Az első beadandó már ki van írva, ezek a könnyebb, gyorsabban elkészíthető feladatok. Az ezekhez tartozó adatbázisról a jövő órai esetleges zh-n már lehetnek kérdések.

. Ez az adatbázis egy 6D motion gesture adatbázis, idősorok.

Mesterséges neuronhálók. 1949-ben indult el a téma, akkor egy pszichológus, Donald Hebb indította el. Róla nevezték el azt a tanulási formát ami miatt ezek roppant robosztusak, illetve kb modellezi az agybeli számításokat. Ez a Hebbi(an) tanulást. A lényege, hogy ha két neuron egyszerre aktív akkor a közöttük lévő kapcsolat változik. Ez az agyban nem pontosan így működik, de az elv hasonló. A neuronokat körrel, a köztük lévő kapcsolatokat súlyozott élekkel. Ezek a súlyok $w_{i,j}$-vel jelöltek, ahol i és j a neuronok indexei. Minden neuron több más neurontól kap inputot és produkál inputot.

. $y_i = f(\sum_n^N w_{i,n}x_n + b_i)$ ahol b_i küszöb, y_i a kimenet, x_i az input. (Ide szerkezeti rajz jönne.) A neuronháló kap valamennyi inputot, és produkál egy outputot, mint egy neuron. A kapcsolatok irányítottak. Az emberi agyban kevés, kb 10^11 neuront tartalmaz. Eszerint, kb 10^22 kapcsolat lehetséges. Az emberi agy csak a felületén a tartalmaz neuronokat a szürke rétegben, de ez csak a külső két milliméter a térfogat maradékát a belső magokon kívül a fehér állomány a kapcsolatokat tartalmazza. Így 10^14-en kapcsolat van. Ha minden kapcsolat létezne, akkor az agyunk kb 100* ekkora lenne. Az agyunk fogyasztja el a teljes energiafelhasználásunk 20%-át. Kb 10% a kapcsolódások megvalósítására megy el.

. Itt sajnos lemerültem....

2. fejezet

EA. 15.09.21.

Erre nem tudtam elmenni.

3. fejezet

EA. 15.09.28.

Lejtés. Szükséges egy koordináta rendszer, ahol van egy x-változó. Itt ábrázoljuk az $y = f(x)$-t. Ennek a lejtését megállapíthatjuk, ha veszünk rajta két pontot (az x-en) és a hozzájuk tartozó értékeket kivonjuk egymásból. Ez így még nem ad pontos eredményt, hiszen a két pont között lehetnek hullámok. Teljesen pontos akkor lesz, ha a két pont közötti távolság infinitezimális. Ezt csináljuk a deriválás során pontosan megkaphatjuk a függvény deriválásával, illetve ez lejtés akkor lesz, ha ezt negatív előjellel vesszük.

. Itt $y = f_x(W, b)$-n szeretnénk W és b változtatásával minimumot találni, az f tipikusan előre meghatározott, az x-ek adottak(ezek a mintapontok). Ezt iteratívan határozzuk meg, mindig a lejtés felé lépve, módosítva W-t és b-t a gradiens vektor mentén. Ezt a gradiens vektort kapjuk meg komponensenkénti deriválásával. Ebből összerakunk egy vektort. A vektoriális eredő negatívja felé kell elindulni, így tudjuk a legmeredekebben a minimum felé lépni. W lehet óriási, sok esetben nagyon bonyolult a derivált számolása.

. (Itt volt több deriválási példa)

Back propagation. A back propagation a függvény függvényének deriválásán alapszik. Sok sok neuron van a rendszerben, ezek sok kapcsolaton, különböző súlyokon keresztül adják az információt további neuronoknak amik szintén így működnek. A rendszerről az ad információt, hogy milyen bemenetre milyen kimenetet ad. Ezt felfoghatjuk egy $x -> y$ függvénynek. Ezt részekre bonthatjuk és ezt ebben a módszerben tesszük is. Ebben a módszerben birtokunkban vannak ideális (x,y) párok, így meghatározhatjuk a rendszer hibáját is (e), illetve próbáljuk ezt minimalizálni.

. Mondjuk hogy van sok (x,y) minta párunk $[(\vec{x}^{(i)}, y^{(i)})_{i=1..n}]$, ahol az x egy vektor. A $\vec{x}$ input halmazt valamilyen leképezésbe bevíve kapunk egy $Y^{(i)}$ halmazt, ami várhatóan eltér a mintától. Ennek a hibájaként meghatározhatjuk $J(W, b)$ költségfüggvényt amit szeretnénk minimalizálni.

$$J(W, b) = \sum_{i=0}^{N} (Y^{(i)} - y^{(i)})^2 \tag{3.1}$$

Mikor lesz $J(W,b) \geq J(W,b') = J(W,b) + (b'-b) * dJ(w,b)/db$ (1. rendig Taylor sorfejtés). Mit kell a $(b'-b)$-be raknom, hogy a bal oldal mindenképp nagyobb legyen a jobbnál? Akkor járunk jól, ha $-\alpha * \frac{dJ(W,b)}{db}$ -t írunk be, $\alpha > 0$ az garantálja, hogy itt valamit kivonunk. Írhatnánk $(b'-b) = \Delta$. Ennek a deriváltnak a kiszámolása lehet nagyon bonyolult, hiszen J nagyon sokféle módon függhet b-től, W-től, ebbe még belejönnek az y_i-k is stb...

. A későbbiekben, ha sok neuron van egy rétegben, azt csak egy vonallal fogom jelölni mint egy réteg. (Rajz: két vonal középen egy nyíl): két neurális réteg összekötve egy mátrixxal: x, (W_1, b_1), y_1, $y_1 = f(W_1 x + b_1)$ és ez megy tovább több réteggel növekedő indexxel: $y_2 = f(W_2 y_1 + b_2) = f(W_2 f(W_1 x + b_1) + b_2)$, ezek szépen egyre bonyolultabbak. Ezek az f-ek egyenként is összetett függvények, összességében egy hosszú számítást kapunk. Ez egy csúnya deriválás, aminek eredményeként megkapjuk a lejtést amit a gradiensben használhatunk. A szépsége, hogy az utolsó előtti súly hangolása nagyon egyszerű, hiszen ott pontosan tudjuk a hibát, de ha a többi súlyt szeretnénk hangolni akkor tudni kell a hibát az előző rétegekben is. Az előző réteg hibáját a $W_{ik} e_k$ adja meg, az adott neuron csak annyiban felelős. Ez a visszaterjesztett hiba. Ez a deriválás szabályain keresztül kijön, bár csúnya számítás végén. Ez a láncszabályból adódik, ahogy a függvény függvényét deriváljuk. Célszerű a nulla környéken vagy véletlen számoknál indítani, mert ott nem nulla a derivált.

. !!A back propagation eljaras megnezese hf, kovetkezo oran valamilyen levezetesi zh kerdes lesz kov oran.!

Autoencoder. Az autoencoder egy speciális rendszer: van egy input(x) réteg, van egy úgynevezett rejtett(h) réteg és van egy output réteg, amelyet y-al fogunk jelölni, köztük egy W, b és W^T, b^T mátrix. $dim(x) = dim(y) > dim(h)$. A hiba:

$$J(W,b) = \frac{1}{2} \sum_{i=0}^{N} (x^{(i)} - y^{(i)})^2 \tag{3.2}$$

A várt kimenet, ahogy itt látható, a várt kimenet is x, ezért autoencoder, mert leviszi kisebb dimenzióba a világot majd ebből újrakreálja a bemenetet. Itt általában nagyon nagy dimenziós terekről beszélünk, például egy arckép esetén $dim(x) = 1000000$. Pl.: nem felismeres arckep alapjan, ferfira -1 nore +1, ahol van birtokunkban van egy adatbazis. $dim(x) = 10^6$-ben kell valamilyen döntési felületet létrehozni. Sok dimenzióban rengeteg lokális minimum lehetséges, nagyon durva lehet a kapott felület. A lépés sugara miatt gömbökben tudunk lepni. Minél több dimenziós kockát nézünk annál több csúcs van, azaz annál több a maradék, amit ezek nem fednek le (a gömb a sarokba nem ér be), így rengeteg a lokális minimum és nagyon furcsán nézhetnek ki. Ha van egy milliárd adatom, akkor hány dimenziós kell legyen az adatom, hány dimenziót feszít ez ki. Egy milliárd adat kb. 20 dimenziót feszít ki, ezeket jól meg kell határozni, ekkor teljesen vissza lehet állítani az eredetit. Ez bonyolult feladat. Ha keveset csökkentek a dimenziók számán, akkor egyszerűbb a feladat és hibamentesen visszaképezhető az eredeti. Ebből kell olyan eljárást csinálni, hogy jól csökkenthessünk. Ritka reprezentációs autoencoder eseten a belső reprezentáció nagyobb dimenziós, így gyorsabban-kevesebb mintából- tud tanulni de lassabban működik.

. Veszünk egy x, h_1, y_1 rendszert ahol y_1 közelíti az inputot. Ezek után kidobjuk az y_1-et és betanítunk egy v_1 mátrixot ami a pontos bemenetet adja. Majd ezt is kidobjuk és építünk egy h_2-t amibe a W_2 vezet, amiben minden információ szerepel és a W_2^T előállítja belőle a h_1-et és így tovább. Ezt nevezzük stacked autoencodernek. 21 rétegig szoktak elmenni. Kb 6-8 évvel ez előtt jutottak el oda, hogy az autoencoder így betanítható, mert kis lépésekben halad. Az autoencoder univerzális approximátor, de nehezen tanítható, NP teljes.

. Jurgen Schmidhuber deep learning.

4. fejezet

EA. 15.10.05

Az output felírása mátrixokkal és vektorokkal. :

A k és az n. neuron a kapcsolatának súlya:	w_{kn}
Az n. neuron outputja ekkor:	$y_n = f(w_{n1}x_1 + \cdots + w_{nK}x_K + b)$
	$n = 1, \ldots N,\ k = 1 \ldots K$
Egyszerűbben(w és x mint vektorok):	$y_n = f(\vec{w}_n^T \vec{x} + b_n)$
Az y vektor tehát (W mátrix, y,x,b vektorok):	$y = f(W\vec{x} + vecb)$

A W felépítése:

$$W = \begin{matrix} w_{11} & \cdots & w_{K1} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nK} \\ \vdots & & \vdots \\ w_{1K} & \cdots & w_{NK} \end{matrix} \tag{4.1}$$

Hangolás és hibafüggvény. Ha a W-t akarjuk hangolni, akkor ismernünk kell a hibát: $(\vec{y}_{out} - \vec{y})^2$, ahol $\vec{y}$ és $\vec{y}_{out}$ vektorok. Ennek a deriváltját kell kibontanunk a függvény-a-függvényben láncszabály alapján, ebből van az óra elején látott szumma. Ez adja meg, hogy a tanulás merre lép.

Rajz: linearis rendszer: y<-x retegek, kettő között W,b kapcsolat $y = W * x + b$

Rajz: y->x Q a nyilon

Generatív rendszer. Van x_{inp} Qy. Ahhoz hogy értelmes legyen: $dim(y) \leq dim(x)$. Amennyiben az egyenlet nem stimmel (x_inp Qy) ?meg kell nezni az

autoencodert?.

$$J(Q,y) = 1/2||x_{inp} - Qy||_2^2 = 1/2 * (x_{inp} - Qy)^T - (x_{inp} - Qy) \tag{4.2}$$

$$-\partial J/\partial y = -1/2 * 2Q^T(x_inp - Qy) \tag{4.3}$$

$$\Delta y = \alpha Q^T * (x_inp - Qy) \tag{4.4}$$

$$QTx_{inp} = Q^T Qy \tag{4.5}$$

$$y = (Q^T Q)^{-1} Q^T x_{inp} \tag{4.6}$$

$$\tag{4.7}$$

$$-\frac{\partial J}{\partial y} \tag{4.8}$$

A 4.3-be Q^T a zárójel belsejének deriválása során jön be oda előre, ugyan úgy mint a -1. Amennyiben ez nem teljes rangú, $Qy = Q(QQ^T)^{-1}Q^T x_{inp}$ $Q(QQ^T)^{-1}Q^TQ(QQ^T)^{-1}Q^T = Q(QQ^T)^{-1}Q^T$ tehát projektív matrix. Ezek annyit csinalnak, hogy: van egy N dimenzios matrix es ez belevetít. Ez a

Rajz: Q ket oszlopa kiad ket vektort, ezek menten van y_1y_2, amik vektorialis eredokent kiadjak x_{inp}-et

pszeudoinverz definíciója egyébként, a parhuzamos vetítés az ugyan az mint a merőleges vetítés???.A vektorialis eredo adja meg az x-nek a projekcióját, a Q^TQ (y) által meghatározott térben.

Q előállítása. Mi van, ha az az alter amit az adataink kifeszitenek nem ugyan az, mint amit a Q kifeszít. Ebben az esetben elég lehet kevesebb vektor. Hogy talaljuk meg a Q-t? Egy arc milyen dimenzióban van? 1000*1000 dimenzió? A tapasztalat szerint lehet találni 10 olyan dimenziót, ami az arcok nagy részét kis hibával visszaállítja. Ezt szeretnénk minták alapján megtanulni. Ha eloallitottuk az y-t, aminel jobbat mar nem tudunk eloallitani akkor a Q-t szeretnenk allitanunk. Ekkor a Q szerint kell lederivalni a koltsegfuggvenyt. Most a Q szerint fogunk derivalni, ezzel sok egyenletet osszeraktam egyetlen sorba. Q-val is linearis ez. A kerdes $-\partial J/\partial Q$ ami $K * N$ db egyenlet.

Autoencoder hebbi tanulással. $-\partial J/\partial Q = -1/2 * 2(x_{inp} - Qy) * (-1) * y^T$ (y az T mert így lesz mátrix az eredmény :))-> $\Delta Q = \beta(x_{inp} - Qy)y^T$ ehhez a következő hálózat tartozik: De korrigálni kell még az y értékét is: ($\Delta y =$

Rajz:x y Qy retegek y->Qy Q-val, $Qy = \hat{x}_{inp}$, $Qy- > x_{inp}$ I(identitas) matrixxal, x->y: αQ^T, y->y : I (hozzaadjuk magához)

$\alpha Q^T(x_{inp} - Qy)$ hibavektor $= (x_{inp} - Qy)$. Ez egy autoencoder úgy felírva, hogy látható legyen a hebbi tanulás.

Rajz: retegek: y, x; y->y I, x->y Q-val (Qy lesz), inp->x x_{inp}, x->y αQ^T

Autoencoder hebbi tanulással másképp. Mi az ami a q_ik hangolásában számít? $\Delta q_{ik} = \beta y_k e_i$ ahol e a fenti hibavektor. Ez a hebbi tanulás. A lényeg,

hogy a $\Delta q_{ik} = \beta g(q_{ik}, y_k, e_i)$ Ez a g egy csúnya függvény is lehet, a fenti a lehető legegyszerűbb: lineáris függvény konstanssal megszorozva. Ez a deriválttól függ. Az autoencoder megvalósítható, egy olyan rendszerben, ahol a hebb szabály érvényesül.

5. fejezet

EA. 15.10.12

ZH volt....

Srác beszel arról, hogy ő hogy csinálta/csinálja és mik voltak a problémák.. Kapott egy linket, a deeplearning.net-ről. Ezen van egy (GSdA?) példa program valami autoencoderrel, ami zajt generál a példában és ezzel próbálja megkülönböztetni a példában lévő hasznos információt. Ez egy adatbázist használ amiben képek vannak a számjegyekről, a rendszer a számjegyeket ismeri fel. Az adatbázis beolvasó függvényt kell módosítani, hogy a miénket olvassa be(6DMG-ben matlab fileok vannak, *.mat). A példa theano es a numpy, scipy.io -t használ. Az output tartalmaz data-t, noise-ot és gest-et(0-19). Az inputot a data-ba kell tenni??. A beolvasott adat egy-egy $14 * T_i$ dim mátrix, ahol T az idő i pedig az index. Ez a 14 dimenzió a sok mindent tartalmaz. Ebből csak az xyz koordináták (position) kellenek. Így kapunk egy $3 * T_i$ dimenziós vektorsorozatot. A T viszont nem lehet változó, de az adatbázisban T különböző lenne minden mozdulatra, mert változó ideig tartott felvenni egy-egy mozdulatot. Meg kell találni a maximumot és kiegészíteni rá valamilyen módszerrel a rövidebb mozdulatokat. Az output a 0-19 gest, az input az előbb leírt. $n_{in} = 3 * T$ $n_{out} = 20$. Lehet még finomítani, de így le kéne futnia. Ezek használata neki egyszerűen ment, de a CUDA gyorsítással problémái voltak. Először GPU nélkül ajánlott futtatni. A következő probléma a megfelelő formátumra hozás volt. A beolvasó írása bonyolult volt. Az 5600 matlab file 5600 dict-be megy, ezekből mindből ki kell szedni az adatot, ezen ő egyenként végigiterált. Lesz egy training egy validation és egy test. Egy closed test-el fogja megállapítani a tanulás hatékonyságát. Ez mind benne van a rendszerben. Kérés volt, hogy nézzük, hogy a második réteg mennyit javít.

A táblára került dolgok-kommentárok magyarázat nélkül. A rejtett rétegekben mindig ReLU-t használjunk A nem linearitás mindig legyen ReLU. A végső osztályzásnál, ahol osztályozás van ott azt a speciálisat kell. Az auto az linearis.
Bayesopt/GRD SEARCH
AE,

Milacski Ádám kiegészítései. Az input nem ideális még így sem, mert az xyz-k lehetnek negativak is, így a ReLU nem biztos, hogy tudja rekonstruálni a negatív értékeket. Tehát valahogy pozitív értékeket kell csinálni ezekből, ez a normalizáló lépés. Interpolálni is kell, hiszen a mozdulat hossza változó. Matlabban van ilyen beépített függvény interp1 néven, valószínű pythonban is van ilyen. Ennek 3 argumentuma van: interp1(hol, mit vesz fel, hol kerem). Ez siman csak a minták között interpolálja a paramétereket lineárisan. Kell meg egy normalizáló lépés is: x_i-t y_i-t, z_i-t normalizaljuk $[0,1]$-be. Ez lesz végül az input.

. AE: x=[x1,..xN] input
x W2f(W1X+[b1,..bN])
f-> ReLU
f(W1X+[b1,..bN]) egy kompresszio
W2 pedig egy projekcio. Ketto egyutt a ReLU
Ez az egesz egyben nemlehet identitas.
W1 sormerete > W1 oszlopmerete.
W1 sormerete = X oszlopmerete
W2 merete = W1T merete
$\sum_{i=1}^{N} ||x_i - w_2 f(w_1 x_i + b)||_2^2$ ez a hiba, ezt akarom minimalizalni.
A stacked autoencoder: $X_0 = X$ es rekurzivan csinaljuk az elozot.
Rajz($x- > h- > \hat{x}(h << x)$) az elozoben a kompresszalt a h az egesz az $\hat{x}$
Kiindulunk az x_0-bol es gyartunk egy h-t mint az elobbes kapnank egy $\hat{x}$-et.
Ezutan eldobjuk az xhat-et es az uj X1 lesz a h. Ebbol csinalunk h2-t amibol rekonstrulajuk a h=hhat-et. A h merete mindig csokken, kb 0.9-re.
$\frac{1}{2}\sum_t ||x_t - \hat{x}_t||_2^2$ ez a rekonstrukcios hiba.
A lepes vegen lesz egy osztalyozo reteg, legyen ez $\hat{Y}$ Egy illetve ket reteg utan is be lehet rakni. A kerdes, hogy hany reteg utan lesz ez jobb.

Osztalyozas. $y\ f(\hat{W}h + [b\ , ..b_{\ n}])y = [0, 0...0, 1, 0..0]$ ez az i. osztalynak megfelelo. y C elemu binarisvektor.
f altalaban itt softmax. ez $softmax(z) = \frac{e^{z_i}}{\sum_j e^{z_j}} \in [0,1]$ ez egy kiemeles, a legerosebb elem kozeliti majd... ?????
koltseg: cross-entropy $-\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} * log(\hat{y}_{ji})$ C az osztalyok szama, N a mintak szama
ez kiad egy
OPT/hyperOPT/spearmint

Cross validation. Ez abbol all, hogy itt van egy X reteg, de ezt felbontjuk sok reszre:

1. sorbarendezem a mintakat
2. felbontom ezt a matrixot 3 reszre legalabb
3. az elso lesz az X amit vegul hasznalok 80% ez a tanito halmaz

4. pl a masodik 10% V mint validacios halmaz, ez arra jo, hogy eloszor X-en tanitunk es teszteljuk a V-n megtalalva az idealis tanulasi ratat, majd XV-n tanitunk.

5. a 3. 10% T mint teszt halmaz

1. veletlen sorbarendezes

2. 3 fele vagas -> XVT

3. tobb hiperparameter kombinacio kiprobalasa X-en tanitva es V-n tesztelve

4. az elozo lepes alapjan kivalasztom a legjobb parameter kombot

5. tanitok XV-n tesztelek T-n

Azert jo ez a modszer mert egyaltalan nem egyertelmu, hogy meddig kell iteralni. Az iteraciok soran az hiba egy ideig szepen csokken majd ugrik fel es beall fixre. Ez a tultanulas, mert utana mar az X specifikus dolgait tanuljuk. Az X-en valo hiba tovabb csokkenne, de a tanult dolgok mar nem.

6. fejezet

EA. 15.10.19

Nezzuk meg a hazikat....

1. gyakorló feladat kis magyarazata.. $x^(pos)$ ket valoszinusegi valtozobol allo vektor.
Az x_1 x_2 azok az ertekek, sorsolni kell az eloszlasbol.
Ezek gauss eloszlasok, (4,4) kozpontnal es (-4,-4)-nel. Ezek korul helyezkednek el a pontok.
A tokeletes dontesi felulet y=-x, de ezt csak nagy elemszamnal kapjuk meg.
30nal nem jott volna ki.

Miert latszik a peldabol hogy hebbi tanulas. (rajz: retegek: e, h; ->e: x; y->h: Q (e=x-Qh); e->h: Q^T ($\Delta h = \alpha Q^T(x - Qh) = \alpha Q^T e$); h->h: I ($h(t+1) = h(t) + \delta h$)

$$J = 1/2 * ||x - Qh||_2^2 \tag{6.1}$$

$$\frac{\partial J}{\partial Q} = -(x - Qh) * h^T \tag{6.2}$$

$$\Delta Q = -\beta * \frac{\partial J}{\partial Q} = \beta * (x - Qh)h^T) = \beta e h^T \tag{6.3}$$

$$\Delta q_{ij} = \beta * h_j e_i \tag{6.4}$$

Ez a leheto legegyszerubb hebbi szabaly. Ez akkor van ha linearis halo. Ha nem linearis halo akkor nem ilyen egyszeru de nem sokkal bonyolultabb.
Olyat nem kerdez, ami mogott ugy erezzuk, hogy vegtelen szamolas all mogotte. Ha ilyet talalunk akkor valoszinuleg rossz uton jarunk.

Linalg... $\Delta h = \alpha Q^T(x - Qh) = 0 -> Q^T x - Q^T Q h = 0$

$$(AB)^T = B^T A^T \quad (6.5)$$

$$(Q^T Q)^T = (Q^T T Q^T) = (Q Q^T) \quad (6.6)$$

$$=> (QQ^T) szimmetrikus. \quad (6.7)$$

$$Pozitivszemidefinit : \lambda_i(Q^T Q) \leq 0 <=> \forall \vec{z} : \vec{z}^T (Q^T Q)\vec{z} \leq 0 \quad (6.8)$$

$$(\vec{z}^T Q^T)(Q\vec{z}) = (Q\vec{z})^T (Q\vec{z}) = ||Q\vec{z}||_2^2 \quad (6.9)$$

$$(6.10)$$

$Q^T Q = PDP^{-1} = PDP^T, D = diag(\lambda_i)$ lehet ilyen alakra hozni a matrixot, mivel szimmetrikus. Ez a !Jordan-felbontas!. ??? Ezt a reszt ki kell egesziteni. ??? $P^T P = I$
$PP^T = I$
Ha van 0 sajatertek, akkor a P-be barmilyen vektor beirhato, ekkor a P-t feltolthetjuk ortogonalis vektorral. Ezek kozul csak azok erdekesek ahol a sajatertekek nemnullak. Ezert a P lambda PT-t megszorozzuk onmagaval, akkor I jon ki.
$(Q^T Q)^{-1} = (PDP^T)^{-1} = (P^T)^{-1} D^{-1} P^{-1} = Q^{-1}(Q^T)^{-1}$

.

$$J(Q,\vec{h}) = \sum_{i=1}^{n}[x-Qh]_i^2 \tag{6.11}$$

$$\frac{\partial J(Q,\vec{h})}{\partial h_k} \tag{6.12}$$

$$\frac{\partial \sum_{i=1}^{n}(x_i - Q_i * h)^2}{\partial h_k} \tag{6.13}$$

$$\sum_{i=1}^{n}\frac{\partial (x_i - Q_i h)^2}{\partial h_k} \tag{6.14}$$

$$\sum_{i=1}^{n} 2 * (x_i - Q_i h) * \frac{\partial x_i - Q_i h}{\partial h_k} \tag{6.15}$$

$$\sum_{i=1}^{n} 2 * (x_i - Q_i h) * (\frac{\partial x_i}{\partial h_k} - \frac{\partial Q_i h}{\partial h_k}) \tag{6.16}$$

$$\sum_{i=1}^{n} 2 * (x_i - Q_i h) * \frac{\partial \sum_{j=1}^{m} Q_{ij} h_j}{\partial h_k}) \tag{6.17}$$

$$\sum_{i=1}^{n} 2 * (x_i - Q_i h) * \sum_{j=1}^{m}\frac{\partial Q_{ij} h_j}{\partial h_k}) \tag{6.18}$$

$$j \neq k => \frac{\partial q_{ij} h_j}{\partial h_k} = 0 \tag{6.19}$$

$$j = k => \frac{\partial q_{ij} h_j}{\partial h_k} = q_{ik} \tag{6.20}$$

$$\sum_{i=1}^{n} 2 * (x_i - Q_i h) * (-q_{ik}) \tag{6.21}$$

$$\sum_{i=1}^{n} -2 * (x_i - Q_i h) * (q_{ik}) \tag{6.22}$$

$$-2Q^T(x-Qh) \tag{6.23}$$

$$\frac{\partial J(Q,h)}{\partial Q} = -2(x-Qh)h^T \tag{6.24}$$

$$\frac{\partial J(Q,h)}{\partial Q_{kl}} =? \tag{6.25}$$

$$\tag{6.26}$$