

TDK dolgozat

Talajvizsgáló rendszer tervezése

Tüü-Szabó Boldizsár

Széchenyi István Egyetem

Műszaki Tudományi Kar

Automatizálási Tanszék

villamosmérnöki MSc. szak 2. évfolyam

automatizálási rendszerek szakirány

Konzulens:

Tormási Alex

egyetemi tanársegéd

Győr

2014 ősz

Tartalomjegyzék

1. Textúrák leírása [1], [2].....	4
1.1. Statisztikai módszerek.....	4
1.1.1 Intenzitás hisztogram [1].....	4
1.2. Local Binary Pattern [2].....	5
1.2.1. Circular LBP	6
1.2.2. Uniform LBP	9
1.2.2. Forgatás-invariáns LBP (rotation invariant LBP)	10
1.2.3. Local Ternary Pattern (LTP)	10
2. Klasszifikációs módszerek	12
2.1. Szupport Vektor Gép [3], [4], [5].....	12
2.1.1. Lagrange duál feladat megoldása (általános eset) [5]	14
2.1.2. A lineáris SVM Lagrange duál feladatának megoldása	15
2.1.3. Puha margójú SVM (Soft margin SVM) [3].....	16
2.1.4. Nemlineáris SVM [3]	17
2.1.5. Több osztályú SVM	19
2.2. Neurális hálózatok [6], [7]	19
2.2.1. Perceptron [6], [7]	20
2.2.2. A perceptron tanulása [6], [7]	22
2.2.4. Többrétegű perceptron [6], [7]	22
2.2.5. Backpropagation algoritmus [6], [7]	23
2.2.6. Backpropagation algoritmus momentummal [6], [7].....	25
2.2.7. Backpropagation algoritmus adaptív tanulási paraméterrel [7]	25
2.2.8. Newton-módszer [6], [7]	27
2.2.9. BFGS (Broyden-Fletcher-Goldfarb-Shanno) eljárás [6], [7]	28
2.2.10. Levenberg-Marquardt algoritmus [6], [7]	28
2.2.11. Konjugált gradiens módszer [6], [7]	29
3. A feladat gyakorlati megvalósítása [8]	31
3. 1. Talaj Textúrájának leírása	31
3.1.1. Circular LBP(8,1).....	31
3.1.2. Uniform LBP	34
3. 2. Klasszifikációs eljárások	35
3.2.1. Tanító pontok előállítása	35
3.2.2. Osztályozás többosztályú SVM-mel	35
3.2.3. Osztályozás előrecsatolt neurális hálózattal	37

Absztrakt

A képfeldolgozás az utóbbi évtizedekben rohamosan fejlődik, és a tudomány fontos területévé vált. A képfeldolgozást számos területén alkalmazzák, például biometrikus azonosításra (ujjlenyomat felismerés, írisz felismerés stb.), de az orvostudományban is fontos szerepet kap. A TMDK dolgozatom célja egy olyan rendszer kidolgozása volt, amely képi információk alapján meghatározza a talaj típusát (fű, homok, aszfalt). Ez az autonóm robotoknál kiemelt jelentőséggel bír. A robot képes lesz a képi információk alapján dönteni arról, hogy milyen útvonalon közlekedjen elkerülve ezzel a nehezebb terepeket, járhatatlan területeket.

A dolgozatom első részében a szakirodalom-kutatás eredményét ismertetem, a feladat megvalósítása során használt textúra-leíró módszerek (LBP-operátor és változatai) valamint klasszifikációs módszerek (szupport vektor gépek, neurális hálózatok) elméleti hátterét foglalom össze.

A dolgozatom második része a feladat gyakorlati megvalósításával foglalkozik. A textúrák leírására a circular LBP-operátort valamint az uniform LBP-operátort használtam. Az egyes talajtípusokról készült képek osztályozására több módszert is kipróbáltam (többsztályú szupport vektor gépek, előrecsatolt neurális hálózatok).

Ismertetem a neurális hálózat struktúrájának (neuronok, rejtett rétegek száma) illetve a neuronok aktivációs függvényének (lineáris, szigmoid stb.) kiválasztása során kapott eredményeket. A neurális hálózatok tanítására több tanulási algoritmust (backpropagation algoritmus és továbbfejlesztett változatai, Levenberg-Marquardt algoritmus, konjugált gradiens módszerek, kvázi Newton módszerek) alkalmaztam, és az egyes algoritmussal kapott eredményeket összehasonlítottam tanulási idő illetve epochszám alapján.

1. Textúrák leírása [1], [2]

Számos leíró kidolgoztak a kép tartalmának azonosítására. A leírókat a kép tulajdonságainak jellemzésére használják (pl: textúrák, színek, alakzatok).

Textúra a szín vagy intenzitások térbeli leírása egy képen vagy annak egy részletén.

Jack Sklansky így definiálta a textúra fogalmát: Egy képen egy területnek állandó textúrája van, ha a lokális statisztikák vagy egyéb lokális tulajdonságok állandóak, lassan változnak, vagy megközelítőleg periódikusak.

A textúrákat 2 csoportba lehet osztani:

- Sztochasztikus
- Determinisztikus

A determinisztikus textúrák jól meghatározott geometriai alakzatok ismétlődéséből állnak.

A textúra leírására számos technikát dolgoztak ki. Tuceryan és Jain (1993) a textúra leírására használt módszereket 4 csoportba osztották:

- statisztikai
- geometriai
- modell alapú
- jelfeldolgozó módszerek

1.1. Statisztikai módszerek

Statisztikai módszereket gyakran alkalmaznak a textúra elemzés során.

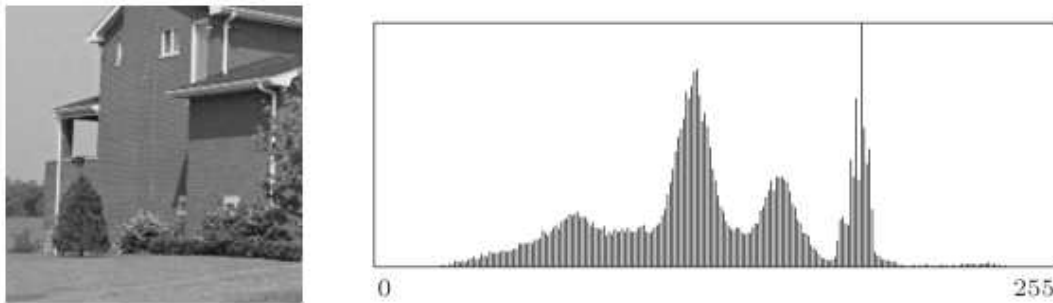
Statisztikai módszert először Kaizer alkalmazott 1955-ben, amikor légi felvételeket elemzett autokorreláció segítségével.

Az elsőrendű módszerek (pl: szórásnégyzet, négyzetes közép, entrópia, intenzitás hisztogram stb.) statisztikai számításokat végeznek az eredeti kép pixelértékeinek felhasználásával.

1.1.1 Intenzitás hisztogram [1]

A képfeldolgozási eljárásokban a képek intenzitáshisztogramja alapvető jelentőséggel bír. A hisztogram metrikusan skálázott tulajdonságok grafikus ábrázolása. Matematikai értelemben a hisztogram egy függvény, ami az egyes diszjunkt osztályokba tartozó megfigyeléseket számolja. A hisztogram, mint grafikon ennek egy ábrázolási módja. Az intenzitáshisztogram azért nagyon népszerű a képfeldolgozási eljárásokban, mert igen szemléletesen mutatja az adott kép intenzitás-eloszlását. Tekintsünk tehát egy 256-szintű

digitális képet, ahol a fekete és a fehér közti tartományt 256 árnyalatra bontottuk fel, úgy, hogy az $i+1$ -edik árnyalat az világosabb, mint az i -edik. Most a szürkességi skálát 256 intervallumra osztjuk, ahol minden intervallumba egyetlen szürkességi érték esik, azaz ($y_0 < y_1 < \dots < y_{255}$). A valószínűségi mintát maga a kép, illetve annak intenzitásértékei adják. A hisztogram i -edik oszlopának magasságát az $[y_i, y_{i+1})$ intervallumba eső elemek száma, azaz a képen lévő i -edik szürkeárnyalatú képpontok száma adja.



1. ábra kép intenzitás hisztogramja

Az elsőrendű módszerek előnye, hogy gyorsan és egyszerűen számíthatóak. Hátrányuk, hogy nem nyújtanak információt a pixelek egymás közötti kapcsolatáról. 2 teljesen különböző kép, amely egyaránt 50-50%-ban tartalmaz fekete és fehér pixeleket ugyanazt az intenzitás hisztogramot eredményezi, ezért a 2 kép elsőrendű statisztikai módszerekkel nem megkülönböztethető.

A másodrendű módszerek pixelek 2 (általában szomszédos) csoportja közötti kapcsolatot állapítják meg.

Harmad- vagy magasabb rendű módszerek is léteznek, de ritkán használják őket a megnövekedett számítási idő és az értelmezési nehézségek miatt.

1.2. Local Binary Pattern [2]

Nagyon hatékony textúra leíró módszer, ami népszerűségét főként a számítási egyszerűségének köszönheti. Gyakran alkalmazzák objektum felismerésre és követésre. A biometrikus alkalmazások területén szintén népszerű (írisz felismerés, ujjlenyomat felismerés, tenyérlenyomat felismerés, járás felismerés stb.).

A másik pontos tulajdonsága a robusztusság a szürke skála szintek megváltozására pl: a megvilágítás megváltozása miatt.

Az eredeti LBP operátor megalkotása Ojala nevéhez fűződik 1996-ban, aki szürke skálás képek leírására készítette el az operátort. Később azonban készítettek LBP-operátorokat

színes (több csatornás) képek, videók és volumetrikus adatok jellemzésére is. A volumetrikus adatokat széles körben alkalmazzák orvosi alkalmazásoknál (3D CT- és MRI).

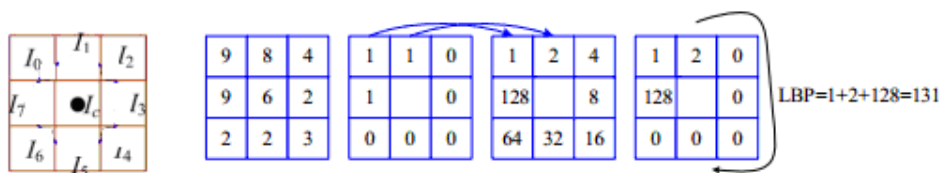
A szürke skálás kép minden pixele köré egy 3*3-as méretű ablakot helyezett. Minden pixelhez kiszámított egy 8 bites bináris számsorozatot az alábbiak szerint:

$$LBP(I_c) = \sum_{i=0}^{P-1} 2^i s(I_i - I_c) \quad (1.1)$$

i : a szomszédos pixelek sorszáma (0-7)

A szomszédos pixel értékéből kivonta a center pixel értékét ($A = I_i - I_c$). Ha a szomszédos pixel a nagyobb értékű ($A > 0$), akkor $s(A) = 1$, különben $s(A) = 0$.

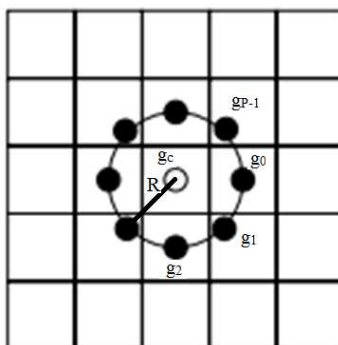
A 8-bites bináris számsorozatokat kapunk, amelyek 256 különböző értéket vehetnek fel. A kapott eredményeket hisztogramon szokás ábrázolni, amit LBP-hisztogramnak neveznek.



2. ábra Local binary pattern számítása

1.2.1. Circular LBP

Olaja 2002-ben kiterjesztette az LBP operátort különböző távolságú szomszédos pixelek vizsgálatára. Extended LBP-nek vagy Circular LBP-nek nevezik, mivel az eredeti LBP-operátor általánosításának tekinthető. Egy tetszőleges, egész pixel sugarú kör kerülete mentén vesszünk fel P db pontot.



3. ábra Extended LBP

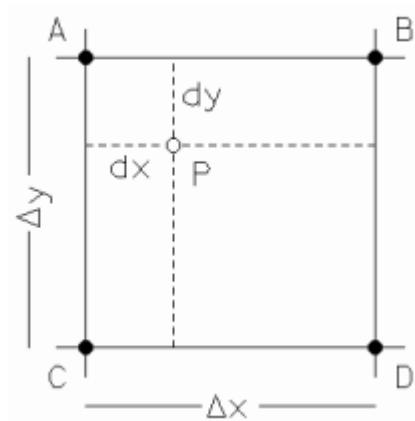
A középső pixel intenzitása: $g_c = I(x, y)$ (1.2)

A pontok intenzitása a kör kerülete mentén: $g_p = I(x_p, y_p)$ $p = 0, 1, \dots, P-1$ (1.3)

A mintavételi pontok koordinátái:

$$\begin{aligned} x_p &= x + R \cos(2\pi p / P) \\ y_p &= y - R \sin(2\pi p / P) \end{aligned} \quad (1.4)$$

Bilineáris interpolációt használunk, ha a mintavételi pont nem a pixel középpontja. A bilineáris interpoláció a síkban mindkét koordináta tengely mentén lineáris interpolációt alkalmaz.



4. ábra bilineáris transzformáció

A P pont intenzitás értéke az alábbi matematikai összefüggéssel számítható:

$$I_p = I_A + dx \cdot (I_B - I_A) + dy \cdot (I_C - I_A) + dx \cdot dy \cdot (I_A - I_B - I_C + I_D) \quad (1.5)$$

Az extended LBP számítása az együttes eloszlásból vezethető le. A helyi textúra a $P+1$ db ($P > 0$) pixel (középső pixel + a P db mintavételi pont a kör mentén) együttes eloszlásával jellemezhető.

$$T = t(g_c, g_0, g_1, \dots, g_{P-1}) \quad (1.6)$$

Információvesztés nélkül kivonhatjuk a középső pixel intenzitásértékét a szomszédos mintavételi pontok intenzitásértékeiből:

$$T = t(g_c, g_0 - g_c, g_1 - g_c, \dots, g_{P-1} - g_c) \quad (1.7)$$

Az együttes eloszlás az eloszlás szorzattá alakításával közelíthető feltéve, hogy a középső pixel intenzitása és az intenzitáskülönbségek statisztikailag függetlenek egymástól:

$$T \approx t(g_c)t(g_0 - g_c, g_1 - g_c, \dots, g_{p-1} - g_c) \quad (1.8)$$

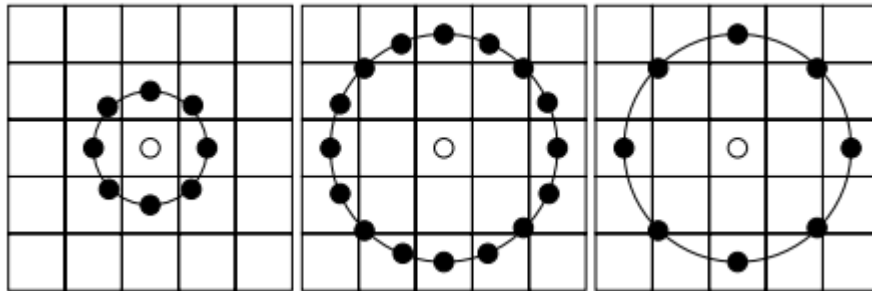
Mivel az első faktor (g_c) nem tartalmaz hasznos információt a helyi textúra leírását illetően, ezért a intenzitáskülönbségek együttes eloszlásával jellemezhetjük a helyi textúrát ($t(g_0 - g_c, g_1 - g_c, \dots, g_{p-1} - g_c)$).

Az eloszlás közelítésére Ojala az intenzitáskülönbségek előjelét vette figyelembe.

$$t(s(g_0 - g_c), s(g_1 - g_c), \dots, s(g_{p-1} - g_c)) \quad (1.9)$$

ahol $s(z)$ lépcsős függvény $s(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \quad (1.10)$

Az LBP értéke az alábbi képlettel számolható ki: $LBP_{p,R}(x_c, y_c) = \sum_{p=0}^{p-1} s(g_p - g_c) 2^p \quad (1.11)$



5. ábra a, LBP(8,1)

b, LBP(16,1)

c, LBP(8,2)

Az ábrán a leggyakrabban alkalmazott LBP-operátorok láthatóak. 8 szomszédos pont esetén $2^8=256$, 16 pont esetén $2^{16}=65536$ különböző LBP értéket kaphatunk egy adott pixelhez. Az LBP értékének a kép összes pixelére való kiszámítása után az eredményeket hisztogramon ábrázoljuk, tehát 256 illetve 65536 oszlopos hisztogramot fogunk kapni.

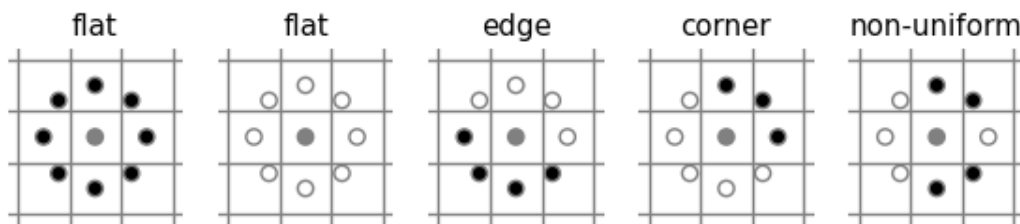
A LBP(8,1)-operátor hasonlít Ojala eredeti LBP-operátorára, hiszen ott is 8 környező pont intenzitásértékét hasonlította össze a középső pixel értékével, viszont az LMP(8,1) esetén nem

minden pont esik a pixel középpontjába (bilineáris interpolációt alkalmazunk). Egyes alkalmazások során szükséges van arra, hogy az operátor forgatás-invariáns legyen. Ilyen operátor megalkotására az LBP(8,1) jobb kiindulási alapként szolgál.

1.2.2. Uniform LBP

Bevezették az uniform pattern fogalmát a zaj csökkentésére az LBP- hisztogramban. Az uniform LBP-operátor bevezetését az indokolta, hogy néhány bitsorozat sokkal többször fordul elő, mint a többi. Ojala 2002-ben megállapította, hogy LBP(8,1) esetén (8 a szomszédos pontok száma, a kör sugara 1 pixel hosszúságú) az előálló bitminták közel 90 %-a uniform. LBP(16,2) esetén ez az arány kb. 70 %-os.

Egy LBP akkor uniform, ha a bitmintán körkörösén végigmenve legfeljebb 2 db 0-1 vagy 1-0 átmenetet kapunk. Pl: 0000000 (0 átmenet) 01110000 (2 átmenet) and 11001111 (2 átmenet) uniformok, de pl. 11001001 (4 átmenet) and 01010010 (6 átmenet) nem azok.



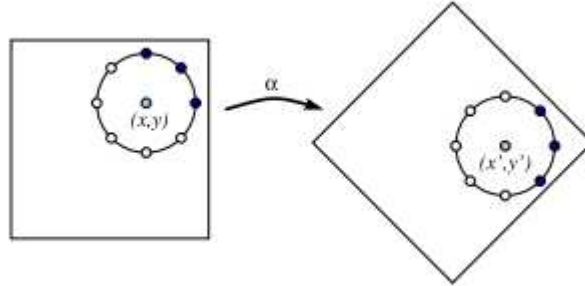
6. ábra uniform és nem uniform minták

Az ábrán 8 bites LBP-bitsorozatokra láthatunk példákat. A fekete pontok jelölik a 0 értékeket, a fehérek az 1 bitértékeket. Ha minden szomszédos pixel értéke kisebb mint a középső pixelé, akkor csak 0 bitekből álló LBP-bitsorozatot kapunk (fekete pontok). Ha viszont minden szomszédos pixel értéke nagyobb mint a középső pixelé, akkor csak 1 bitekből álló bitsorozatunk lesz (fehér pontok). Ezeket, a csak azonos értékű bitekből álló bitsorozatokat a szakirodalom flat jelzővel illeti.

8 szomszédos pont esetén 256 különböző bitmintát kaphatunk, amelyből 58 lesz uniform. A nem uniform mintákat nem különböztetjük meg egymástól, így 59 különböző címkénk lesz, szemben az eredeti LBP-vel, amikor 256 volt. 16 mintavételi pont esetén 65536 helyett csak 243 címkénk lesz, mivel 242 bitsorozat uniform.

1.2.2. Forgatás-invariáns LBP (rotation invariant LBP)

Sok esetben szükség van arra, hogy a textúrák jellemzésére forgatás-invariáns LBP-t használjunk.



7. ábra forgatás-invariáns LBP

Az ábrán jól megfigyelhető a kép forgatásának hatása az LBP-operátorra. A kör mentén elhelyezkedő szomszédos pontok is elfordulnak, így megváltozik a középső pixelhez tartozó LBP-érték.

Forgatás-invariáns leképezéssel az LBP-operátor (részben) forgatás-invariánssá tehető. A leképezés az összes LBP bináris kód legkisebb értékükbe való körkörös bitenkénti jobbra forgatását jelenti:

$$LBP_{P,R}^{ri} = \min_i ROR(LBP_{P,R}, i) \quad (1.12)$$

Így például a 10000110, 00110100, and 00001101 bináris kódokhoz a leképezés után ugyanaz, a 00001101 kód fog tartozni.

Ez az operátor csak a $\frac{a \cdot 360^\circ}{P}$ szöggel ($a=1,2,\dots,P-1$) való elfordulásokra lesz invariáns, de osztályozási feladatok eredményei azt mutatják, hogy az operátor más szögelfordulások esetén is nagy robosztusságot mutat.

1.2.3. Local Ternary Pattern (LTP)

Az LBP kibővítésének tekinthető. Az LBP-nél a küszöb a középső pixel értékénél volt, ezért érzékeny a zajokra, ennek csökkentésére vezették be az LTP-t.

Itt szintén egy pixel értékét hasonlítja össze a szomszédos pixelekkal, de bevezetnek egy küszöbértéket (k).

$$T = \begin{cases} 1 & T \geq (c + k) \\ 0 & T < (c + k) \text{ és } T > (c - k) \\ -1 & T \leq (c - k) \end{cases} \quad (1.13)$$

Ahol c a középső pixel értéke, T a szomszédos pixel értéke

Az így előálló ternáris kódot, 2 bináris kódra osztják, az első tartalmazza a pozitív értékeket, a második a negatív értékeket. A pozitív és a negatív értékeket tartalmazó kódokból külön-külön hisztogramot számolunk, így az LTP mérete az LBP méretének 2-szerese lesz.

Az ábrán láthatunk egy példát az LTP számítására 8 szomszédos pixel esetén.

94	38	54	1	-1	0
23	50	78	-1		1
47	66	12	0	1	-1

Ternary Pattern (k=5): 1(-1)01(-1)10(-1)
Part1=10010100, Part2=01001001

8. ábra LTP számítása

2. Klasszifikációs módszerek

A klasszifikációs módszerek osztályozási problémák megoldására szolgálnak. Ebben a fejezetben ismertetem a leggyakrabban használt klasszifikációs módszereket.

Legfontosabb klasszifikátorcsaládok:

- legközelebbi szomszéd módszer
- neurális hálók
- döntési szabályok, döntési fák
- Naiv-Bayes módszerek
- Szupport vektor gépek
- Random forest

2.1. Szupport Vektor Gép [3], [4], [5]

Az SVM (Support Vector Machine) Vladimir N. Vapnik és Corinna Cortes többféle gépi tanulási feladatra alkalmazható modellje (1995).

Alapváltozata lineáris kétosztályos szeparálásra képes, de kiterjeszthető

- Egy- vagy többosztályos szeparálásra
- Nemlineáris szeparálásra
- Nemlineáris regressziós feladatokra

SVM lineárisan szeparálható adatokra:

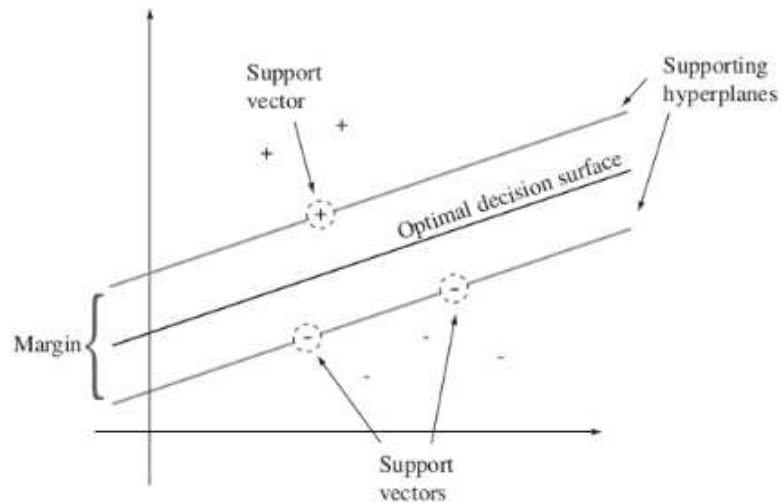
Legyen 2 különböző osztály: $y \in \{+1, -1\}$ (2.1)

A tanulóadatbázis álljon l darab tanítópontból (n dimenziós vektorok): $(\vec{x}, y)$ ahol $\vec{x} \in R^n$, $y \in \{+1, -1\}$ (2.2)

A tanítóhalmaz: $D: \{(\vec{x}_i, y) \mid \vec{x}_i \in R^n, y \in \{+1, -1\}\}$ (2.3)

A szeparáló hipersíkot úgy kell megválasztani, hogy minél messzebb legyen a 2 osztály pontjaitól, és pontosan középen legyen a 2 osztály pontjai között.

Margó-nak (margin) nevezzük a szeparáló hipersíkkal párhuzamos síkok által meghatározott térrészt, amely nem tartalmaz tanítópontokat. A hipersíkhöz legközelebb eső tanítópontok a szupport vektorok.

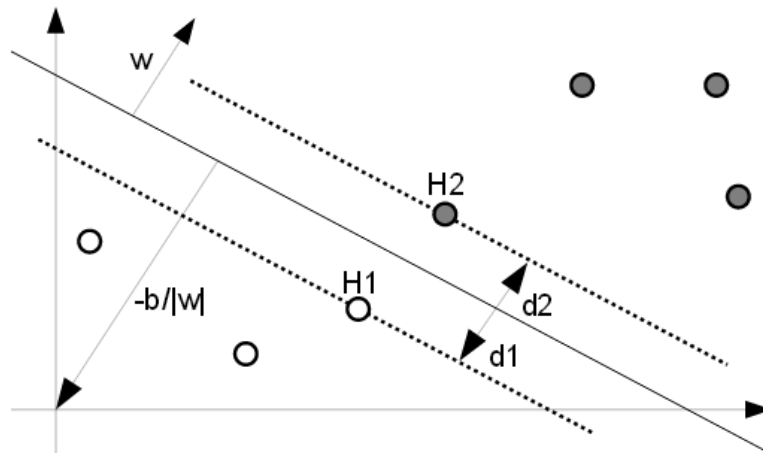


9. ábra szupport vektor gépek

Amennyiben a tanuló adatbázis lineárisan szeparálható a hipersík az alábbi egyenlettel fejezhető ki: $\vec{w} \cdot \vec{x} + b = 0$ (2.4)

ahol w a hipersíkra merőleges vektor (normálvektor)

$\frac{b}{|\vec{w}|}$ a hipersík távolsága az origó-tól



10. ábra elválasztó hipersík

A feladatunk az elválasztó hipersíkot úgy beállítani (w és b értékének megfelelő meghatározása), hogy a margó maximális legyen. A tanításra vonatkozó korlátok az alábbiak szerint alakulnak:

$$\vec{w} \cdot \vec{x}_i \geq 1 + b \text{ minden } (\vec{x}_i, y_i) \in D - re, \text{ ahol } y_i = +1 \quad (2.5)$$

$$\vec{w} \cdot \vec{x}_i \leq 1 - b \text{ minden } (\vec{x}_i, y_i) \in D - re, \text{ ahol } y_i = -1 \quad (2.6)$$

Ami $\vec{w} \cdot (y_i \vec{x}_i) \geq 1 - y_i b$ minden $(\vec{x}_i, y_i) \in D - re$ alakban írható fel a 2 korlátot egyesítve.

A margó maximalizálendő hossza $\frac{2}{|\vec{w}|}$, tehát a w vektor hosszától függ. Ez a feladat a négyzetgyökös kifejezés miatt bonyolult, de szerencsére a w vektor hossza ekvivalens $\frac{1}{2}|\vec{w}|^2$ kifejezéssel. Az optimalizálási feladat tehát a $\frac{1}{2}|\vec{w}|^2$ minimalizálását jelenti, amely egy kvadratikus programozási feladat. Kvadratikus programozási feladatnak nevezzük az olyan optimalizálási feladatot, amelyben a feltételek lineárisak, a célfüggvény pedig egy kvadratikus függvény. A kvadratikus programozás hátránya a nagy számításigény, mivel az összes tanulópont korlátot jelent. Egy optimalizálási feladathoz hozzárendelhető egy vele szoros kapcsolatban levő másik optimalizálási feladat. Az eredeti feladatot primál feladatnak nevezik, a primál feladathoz kapcsolódó optimalizálási feladatot duál feladatnak nevezik. Duál feladatot többféleképpen lehet definiálni, de szupport vektor gépek esetén a Lagrange duál feladatot szokták használni.

2.1.1. Lagrange duál feladat megoldása (általános eset) [5]

Legyen a primál feladat az alábbi:

$$\begin{aligned} f(x) &\rightarrow \min \\ g_i(x) &\leq 0 \quad i=1,2,\dots,m \\ h_i(x) &= 0 \quad i=1,2,\dots,k \\ x &\in X \end{aligned} \quad (2.7)$$

ahol $f(x)$ a primál feladat célfüggvénye

$g_i(x), h_i(x)$ feltételek

A primál feladathoz tartozó Lagrange duál feladat az alábbiak szerint írható fel:

$$\begin{aligned} \Theta(u_1, \dots, u_m, v_1, \dots, v_k) &\rightarrow \max \quad (2.8) \\ u_i &\geq 0 \quad i=1, \dots, m \end{aligned}$$

$$\text{ahol } \Theta(u_1, \dots, u_m, v_1, \dots, v_k) = \inf\{L(x, u, v) : x \in X\} = \inf\left\{f(x) + \sum_{i=1}^m u_i g_i(x) + \sum_{i=1}^k v_i h_i(x) : x \in X\right\}$$

(2.9)

$L(x, u, v)$ a Lagrange-függvény, az egyes feltételekhez rendelt $u_1, \dots, u_m, v_1, \dots, v_k$ együtthatókat Lagrange szorzóknak nevezzük.

A Lagrange duál feladat célfüggvénye tehát a Lagrange függvénynek minden Lagrange szorzóhoz tartozó infimuma az X halmazon. A Lagrange duál feladat feltételi halmaza

egyszerű, hiszen csak az egyenlőtlenségekhez tartozó Lagrange szorzókra ír elő nemnegativitást.

2.1.2. A lineáris SVM Lagrange duál feladatának megoldása

Az előző részben felírtam hogyan kell meghatározni egy primál feladathoz tartozó Lagrange duál feladat megoldását általános esetben.

Alkalmazzuk az ott felírt összefüggéseket az SVM-hez tartozó kvadratikusan programozási feladat Lagrange duál feladatának meghatározására és megoldására!

Az eredeti kvadratikusan programozási feladat:

$$\min_{\vec{w}, b} [\Phi(\vec{w}, b)] = \min_{\vec{w}, b} \frac{1}{2} |\vec{w}|^2 \quad (2.10)$$

$$y_i \cdot (\vec{w} \cdot \vec{x}_i - b) - 1 \geq 0 \text{ a feltételek}$$

Az ehhez tartozó Lagrange duál feladat így írható fel:

$$\max_{\alpha \geq 0} [\Theta(\vec{\alpha}, \vec{w}, b)] = \max_{\alpha \geq 0} \min_{\vec{w}, b} \left\{ \frac{1}{2} |\vec{w}|^2 - \sum_{i=1}^n \alpha_i [y_i (\vec{w} \cdot \vec{x}_i - b) - 1] \right\} \quad (2.11)$$

ahol $\alpha_i \geq 0$ az egyes feltételekhez rendelt Lagrange szorzók

Az ellentétes irányú szélsőértékek miatt ez szemléletesen azt jelenti, hogy $\Theta(\vec{\alpha}, \vec{w}, b)$ nyeregpontját keressük. Mivel $\Phi(\vec{w}, b)$ konvex pontosan 1 ilyen nyeregpont van.

Bizonyítható, hogy a $(\vec{w}^*, b^*, \vec{\alpha}^*)$ megoldás, akkor és csak akkor lesz az eredeti feladatnak is szélsőértéke, ha teljesülnek a Karush-Kuhn-Tucker feltételek:

$$\begin{aligned} \frac{\partial \Theta}{\partial \vec{w}}(\vec{w}^*, b^*, \vec{\alpha}^*) &= \vec{0} \\ \frac{\partial \Theta}{\partial b}(\vec{w}^*, b^*, \vec{\alpha}^*) &= 0 \\ \alpha_i^* (y_i (\vec{w}^* \cdot \vec{x}_i - b^*) - 1) &= 0 \\ y_i (\vec{w}^* \cdot \vec{x}_i - b^*) - 1 &\geq 0 \\ \alpha_i^* &\geq 0 \end{aligned} \quad (2.12)$$

Ha minden feltétel teljesül, akkor

$$\Theta(\vec{\alpha}^*, \vec{w}^*, b^*) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \quad (2.13)$$

A Karush-Kuhn-Tucker feltételek felhasználásával végül a maximális margó megtalálásának duális alakja:

$$\max_{\vec{\alpha}} [\Theta(\vec{\alpha}^*, \vec{w}^*, b^*)] = \max_{\vec{\alpha}} \left(\sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \right) \quad (2.14)$$

$$\begin{aligned} \sum_{i=1}^l \alpha_i y_i &= 0 \\ \alpha_i &\geq 0 \end{aligned} \quad \text{feltételek}$$

Az egyik Karush-Kuhn-Tucker feltétel szerint:

$$\alpha_i^* (y_i (\vec{w}^* \cdot x_i - b^*) - 1) = 0 \quad (2.15)$$

Ebből az következik, hogy:

- $\alpha_i^* = 0$ vagy
- $y_i (\vec{w}^* \cdot x_i - b^*) - 1 = 0$.

Ha $\alpha_j > 0$ egy $(\vec{x}_j, y_j) \in D$ tanulópontra, akkor az a tanulópont rajta van az egyik szupport-síkon.

Látható, hogy a duális megoldásban a hipersíkhöz legközelebb eső tanulópontokat határozzuk meg (szupport vektorok), amelyek a szupport-síkokat korlátozzák. Ezzel szemben a primál feladat megoldása a szupport-síkokat határozza meg.

Ha klasszifikátort arra szeretnék használni, hogy egy pontot a 2 osztály valamelyikébe besoroljon, akkor az alábbi döntési függvénnel határozhatjuk meg a pont osztályát:

$$f(\vec{x}) = \text{sgn}(\vec{w}^* \cdot \vec{x} - b^*) = \text{sgn} \left(\sum_{i=1}^l \alpha_i^* y_i \vec{x}_i \cdot \vec{x} - \sum_{i=1}^l \alpha_i^* y_i \vec{x}_i \cdot \vec{x}_{sv+} + 1 \right) \quad (2.16)$$

ahol $\vec{x}_{sv+}$ egy tetszőleges szupport vektor a +1 osztályból

2.1.3. Puha margójú SVM (Soft margin SVM) [3]

Az előző részben ismertetett SVM megoldása optimális esetben érvényes, amikor az adatok lineárisan szeparálhatóak, a valóságban viszont az adataink zajosak lehetnek, és így az adatainkat nem lehet tökéletesen szétválasztani egy hipersíkkal. Általában vagy csak olyan lineáris szeparálás lehetséges, ahol a két osztály között csak nagyon kis biztonsági sáv alakítható ki, vagy lineáris szeparálás hibamentesen nem is lehetséges. A kis biztonsági sávot növelhetnénk, ha nem ragaszkodnánk ahhoz, hogy minden tanítópont a sávon kívül helyezkedjen el. Ha a pontok többsége a sávon kívül lesz, de néhány mintapontnál megengedjük, hogy a sávon belülré kerüljenek (sőt esetleg azt is megengedjük, hogy egyes mintapontok az elválasztó hipersík „rossz” oldalára kerüljenek), akkor ezen az áron a sáv mérete esetleg jelentősen növelhető.

A probléma megoldására puha margókat (soft margin) alkalmazunk. Az osztályozási feladatra vonatkozó feltételeket gyengítjük kiegészítő (slack) változók bevezetésével:

$$y_i(\vec{w} \cdot \vec{x}_i - b) > 1 - \varepsilon_i \text{ ahol } i = 1, \dots, l \text{ és } \varepsilon_i > 0 \quad (2.17)$$

Egy ε_i kiegészítő változó az elválasztó hipersík hibáját adja meg az x_i vektor esetén:

- Ha $0 < \varepsilon_i < 1$, akkor az x_i vektor a margón belülre esik, de a hipersík megfelelő oldalára
- Ha $\varepsilon_i > 1$, akkor az x_i vektor a hipersík nem megfelelő oldalára esik

A szupport vektorok azok az x_i vektor, amelyekre teljesül:

$$y_i(\vec{w} \cdot \vec{x}_i - b) = 1 - \varepsilon_i \text{ ahol } i = 1, \dots, l \text{ és } \varepsilon_i > 0 \quad (2.18)$$

Az optimalizálási feladat a következő lesz a kiegészítő változók bevezetésével:

$$\min_{\vec{w}, \varepsilon, b} \left\{ \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^l \varepsilon_i \right\} \quad (2.19)$$

$$y_i(\vec{w} \cdot \vec{x}_i - b) > 1 - \varepsilon_i \text{ feltételek, ahol } i = 1, \dots, l \text{ és } \varepsilon_i > 0$$

A célfüggvényben szereplő C egy pozitív értékű paraméter, amit a felhasználó határoz meg a megoldás során, regularizációs paraméternek nevezzük.

A maximális margó megtalálásának duális alakja a Karush-Kuhn-Tucker feltételek felhasználásával határozható meg. Látható, hogy a képlet megegyezik a lineáris SVM-nél kapott összefüggéssel, nem tartalmazza a kiegészítő változókat, csak a feltételek változtak meg:

$$\max_{\vec{\alpha}} [\Theta(\vec{\alpha}^*, \vec{w}^*, b^*)] = \max_{\vec{\alpha}} \left(\sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \right) \quad (2.20)$$

$$\sum_{i=1}^l \alpha_i y_i = 0 \quad \text{feltételek}$$

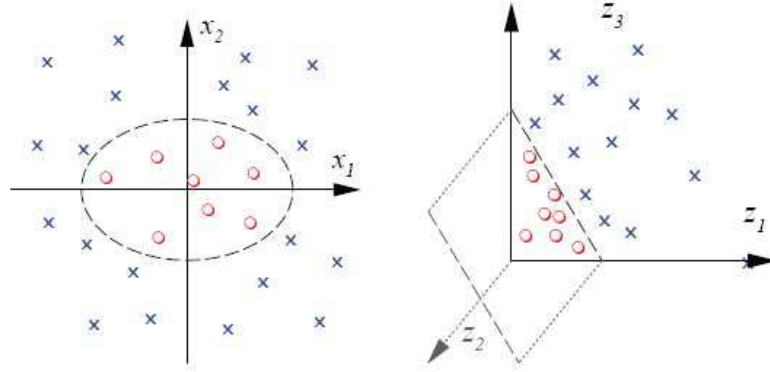
$$0 \leq \alpha_i \leq C$$

2.1.4. Nemlineáris SVM [3]

Abban az esetben, ha a tanulóponatok nem szeparálhatóak lineárisan az eredeti tanítóponokat magasabb dimenzióba transzformáljuk remélve, hogy ott már lineárisan szeparálhatóak lesznek.

A kernel trükk lényege, hogy olyan problémáknál, melyeknél a változók skaláris szorzata szerepel csak, a skaláris szorzatot lecserélhetjük a nagyobb dimenziós térben vett skaláris

szorzatra. Ennek a kiszámításához nincs szükség a nagyobb dimenziós térbe képező függvényre ($\varphi(\vec{x})$ -re).



11. ábra kernel trükk alkalmazása

Kernel függvénynek nevezünk egy olyan függvényt, amely vektorok egy transzformált skaláris szorzatát az eredeti adattérben számítja ki.

$$K(\vec{x}, \vec{y}) = \varphi(\vec{x}) \cdot \varphi(\vec{y}) \quad (2.21)$$

Egy Kernel függvény szimmetrikus a változóiban: $K(\vec{x}, \vec{y}) = K(\vec{y}, \vec{x})$ (2.22)

A lineáris SVM-nél felírt döntési függvény az alábbi alakra változik Kernel függvény $K(\vec{x}_i, \vec{x}) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x})$ használatával:

$$f(\vec{x}) = \text{sgn}(\vec{w}^* \cdot \varphi(\vec{x}) - b^*) = \text{sgn}\left(\sum_{i=1}^l \alpha_i^* y_i \varphi(\vec{x}_i) \cdot \varphi(\vec{x}) - b^*\right) = \text{sgn}\left(\sum_{i=1}^l \alpha_i^* y_i K(\vec{x}_i, \vec{x}) - b^*\right) \quad (2.23)$$

A szükséges és elégséges feltételeket ahhoz, hogy egy függvény Kernel függvény legyen a Mercer-tétel fogalmazza meg:

Egy változóiban szimmetrikus folytonos $K(\vec{x}, \vec{y})$ függvény akkor és csak akkor fejezhető ki $K(\vec{x}, \vec{y}) = \varphi(\vec{x}) \cdot \varphi(\vec{y})$ alakban, ha minden olyan $g(x)$ függvény esetén, amelyre $\int g(x)^2 dx$ véges, teljesül $\int \int K(x, y) g(x) g(y) dx dy > 0$.

A leggyakrabban használt Kernel függvények:

- Polinomiális (homogén): $K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)^d \quad (2.24)$

- Polinomiális (inhomogén): $K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d \quad (2.25)$

- Gauss (RBF): $K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma |\vec{x}_i - \vec{x}_j|^2) \quad (2.26)$

ahol $\gamma > 0$

- Hiperbolikus tangens:

$$K(\vec{x}_i, \vec{x}_j) = \tanh(\kappa \vec{x}_i \cdot \vec{x}_j + c)^d \quad (2.27)$$

ahol $\kappa > 0$ és $c < 0$

2.1.5. Több osztályú SVM

Az SVM-et eredetileg bináris osztályozási feladatokra dolgozták ki, de 2-nél több osztályos problémák megoldására is alkalmassá tehető.

2 alap stratégiát különböztetünk meg a több osztályból álló problémák SVM-mel történő megoldására:

- a többosztályos osztályozási feladat megfogalmazása feltételes szélsőérték-számítási feladatként
- a többosztályos osztályozási feladat felbontása több bináris osztályozási feladattá

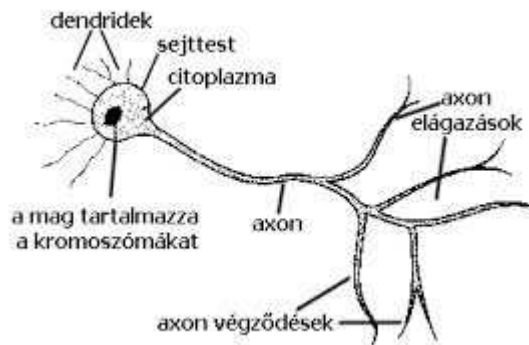
A többosztályos feladat több bináris feladattá történő felbontása esetén egy vektor osztályát az egyes bináris SVM-ek előrejelzéseinek összegzése alapján határozzák meg. Ehhez többségi szavazást szokás alkalmazni, a vektor abba az osztályba fog tartozni, amelyik a legtöbb szavazatot kapta.

A gyakorlatban 2 féle megoldás terjedt el:

- „egy az egy ellen” (one against one) : Az összes különböző osztálypárhoz 1-1 bináris osztályozót készítünk, így $k > 2$ osztály esetén $\frac{k(k-1)}{2}$ darab osztályozónk lesz. Minden osztályozó csak az adott 2 osztályhoz tartozó vektorokat használja fel.
- „egy a többi ellen” (one against others): olyan bináris osztályozók alkotása, amelyek 1 osztályt tekintenek + osztályként, az összes többit – osztályként, így összesen k darab bináris osztályozónk lesz.

2.2. Neurális hálózatok [6], [7]

A mesterséges neurális hálózatok működésének megértéséhez szükséges megismerni az agyban található neurális hálózatokban végbemenő folyamatokat, hiszen a mesterséges neurális hálózatok ezek mintájára épülnek.



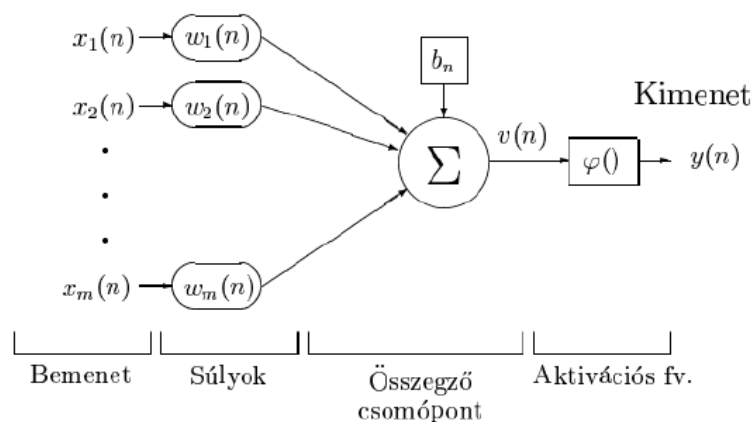
12. ábra Neuron felépítése

Neuronnak nevezzük az idegsejt és nyúlványainak együttesét. A neuronok az idegrendszer legkisebb elemét jelentik. A neuronok ingerérzékeny sejtek, amelyeknek az ingerfelvétel és az idegi ingerek vezetése a feladatuk.

A dendritek azok a nyúlványok, amelyek az információ felvételére és a sejttest irányába történő vezetéséért felelősek. A dendritek az agyi neuronok bemenetei, amelyek más neuronokhoz csatlakoznak. Az információ-feldolgozást a neuronok sejtmagja végzi. Az axon egy cső alakú nyúlvány, amely az ingerületeket a sejttől távolodó irányba vezeti. Az axon végződése több másik neuron dendritjeihez csatlakozhatnak.

2.2.1. Perceptron [6], [7]

A neurális hálózat elemi egységekből, neuronokból épülnek fel, melyek valamilyen lokális feldolgozást hajtanak végre. A neuronok valamilyen struktúra szerint összekapcsolódva hálózatot alkotnak. A neurális hálózat tanulása minták alapján megy végbe, a megszerzett tudás a neuronok közötti összeköttetésekben, súlyokban tárolódik.



13. ábra Perceptron felépítése

A mesterséges neuron az agyi neuron másolata, egy több bemenetű, egy kimenetű elem, ahol a bemenetek súlyozva egy összegző bemenetére kerülnek.

A Rosenblatt vagy egyszerű perceptron egy olyan hálózat, amely a megfelelő beállítás illetve tanítás után egy lineáris szeparálási probléma megoldására alkalmas.

A neuron a biológiai neuronhoz hasonlóan reagál a feldolgozott inputokra, a súlyozott összeg egy aktivációs függvényen megy keresztül. A bemeneti súlyozott összeget ingernek (excitation) is szokták nevezni.

A bemeneti súlyozott összeg M db bemenet esetén:

$$v(n) = \sum_{i=1}^M w_i x_i + b_n \quad (2.28)$$

A súlyozott összeghez hozzáadunk egy b eltolási (bias) értéket. Az eltolási érték nélkül az osztályokat elválasztó hipersík az origón minden esetben áthaladna. Az eltolási érték lehetővé teszi, hogy a hipersík b -vel való eltolásával olyan adathalmazok osztályozását is meg tudjuk oldani, amit egy origón áthaladó hipersíkkal nem lehetett volna. Általában a $b(n)$ torzítást is súlyként szoktuk értelmezni és $w(n)$ -ként hivatkozunk rá. A torzításhoz tartozó bemeneti értéket pedig $x(n) = 1$ -nek vesszük.

Az aktivációs függvény kimenetén megjelenő érték továbbítódik a hozzá kapcsolódó neuronokhoz. A kimeneti jelet válasznak (activation) is szokták nevezni.

Az aktivációs függvénynek az esetek többségében nemlineáris függvényt kell választani. Lineáris aktivációs függvény használatával a neuron egy lineáris függvényt valósítana meg, amely sok esetben azt eredményezné, hogy a hálózat hibája nem csökkenne le kellőképpen.

A leggyakrabban alkalmazott aktivációs függvények az alábbiak:

- Egységugrás függvény:
$$y(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad (2.29)$$

- Előjelfüggvény:
$$y(x) = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases} \quad (2.30)$$

- Sigmoid függvény
$$y(x) = \frac{1}{1 + e^{-x}} \quad (2.31)$$

Az egyes rétegekhez tartozó neuronok a bemenettel vagy az előző réteg neuronjainak kimenetével illetve a következő réteg bemenetével vannak összekötve. A legtöbb hálózat esetében a rétegek teljesen össze vannak kötve, tehát a réteg minden kimenete össze van kapcsolva a következő réteg összes bemenetével.

Az egyes rétegeket feladatuk alapján különböztetjük meg:

- Bemeneti réteg: Ebben a rétegben azok a neuronok találhatók, amelyek a bemeneti jelek továbbítását végzik a további rétegek felé.
- Rejtett réteg: A tulajdonképpeni információfeldolgozást ebben a rétegben találhatók neuronok végzik. A rejtett rétegek száma több is lehet.
- Kimeneti réteg: Ebben a rétegben található neuronok továbbítják az információt a külvilág felé.

2.2.2. A perceptron tanulása [6], [7]

A tanulási folyamat tanulóadatokkal történik. A tanulóadatoknál a bemenetekhez tartozó kimeneti érték ismert. A bemeneti értékeket a perceptron bemeneteire adjuk, és kiszámoljuk a neuron kimenetét. A neuron kimeneti értékét összehasonlítjuk az elvárt kimeneti értékkel. A két érték közötti különbséget a hálózat hibájának nevezzük. A tanulás során az a célunk, hogy a hálózat hibája lehetőleg minél kisebb értékre csökkenjen. Az ideális az lenne, ha a hiba 0-ra csökkenne, de általában ez nem valósítható meg.

Jelölje az $x(n)$ bemenethez tartozó ideális kimenetet $d(n)$, illetve $y(n)$ jelentse a hálózat által az $x(n)$ bemenetre adott kimenetét, ekkor a neurális hálózat négyzetes hibája a következőképpen számítható ki:

$$\varepsilon = (d(n) - y(n))^2 \quad (2.32)$$

A tanulás szabályai:

- ha az adott bemenet mellett számított kimenet jó ($y_i = d_i$), akkor a súlytényezőket nem változtatjuk;
- ha $y_i = 0$, de $d_i = 1$, akkor növeljük az "aktív" inputok (melyek zérusnál nagyobbak) súlytényezőit;
- ha $y_i = 1$, de $d_i = 0$, akkor csökkentjük az "aktív" inputok súlytényezőit.

A fenti növelések és csökkentések mértéke sokféle lehet, tipikusan:

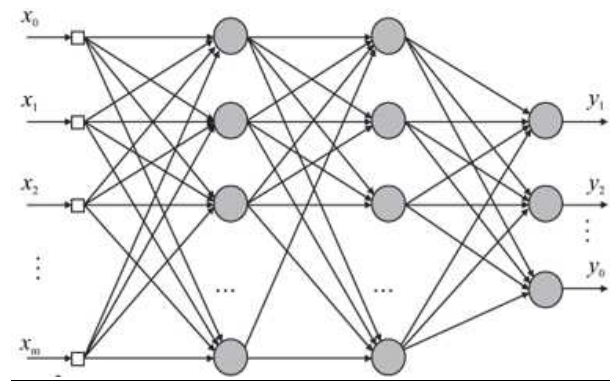
- rögzített növekmény
- a $(d_i - y_i)$ –vel arányos növekmény
- az előző kettő kombinációja

2.2.4. Többrétegű perceptron [6], [7]

Egy egyszerű, lineárisan szeparálható feladat egyetlen perceptronnal is megoldható, viszont bonyolultabb, lineárisan nem szeparálható feladatok megoldásához többrétegű perceptron

hálózat szükséges. A többrétegű perceptron hálózat áll egy bemeneti rétegből, rejtett réteg(ek)ből és egy kimeneti rétegből.

A rejtett réteg és az abban elhelyezkedő perceptronok száma lényegesen befolyásolhatja, általában javíthatja, a hálózat approximációs képességét, hiszen egy bonyolultabb függvénykapcsolatot reprezentál a bemenet és a kimenet között. Viszont egy nagyobb hálózat tanítása hosszabb időt vesz igénybe.



14. ábra többrétegű előrecsatolt hálózat felépítése

2.2.5. Backpropagation algoritmus [6], [7]

Elsőrendű gradiens módszer, amely a hiba elsőrendű deriváltjait használja iteratív módon a hálózat súlyainak beállítására. A módszer hátránya, hogy könnyen beragadhat egy lokális minimumba.

A hálózat által meghatározott kimenet és az elvárt kimenet közötti különbség az alábbi összefüggéssel írható fel, amit a neurális hálózat hibájának nevezünk:

$$E = \frac{1}{2} \sum_{i=1}^m (d_i(n) - y_i(n))^2 \quad (2.33)$$

A backpropagation algoritmust használjuk arra, hogy a hiba minimumát megtaláljuk.

Ebben az összefüggésben d_i a i . mintához tartozó ideális kimenet, y_i az i . mintára számított kimenet, m pedig a minták darabszáma.

A súlyok meghatározására szolgáló iterációs eljárás két fázisból áll:

- 1) propagáció: az aktuális súlyokkal a bemenő jelekből a kimeneti jelek kiszámítása.
- 2) backpropagáció: a bemeneti jelekhez tartozó tanulóhalmazbeli kimeneti jelek elő írt értékei és a valóságos kimenetek alapján a súlyok módosítása, a kimeneti rétegtől a bemeneti réteg felé, azaz visszafelé haladva.

Leggyakrabban szigmoid aktivációs függvényt szoktak alkalmazni, mivel ez a függvény deriválható.

- a) minden kimeneti neuronra kiszámítjuk a módosult súlyokat a gradiensek meghatározása után szigmoid aktivációs függvény használata esetén:

$$\begin{aligned}\frac{\partial E}{\partial w_{j,k}} &= y_j \delta_k \\ \delta_k &= y_k(1-y_k)(d_k - y_k) \\ w_{l,k}^* &= w_{l,k} + \eta \delta_k y_l\end{aligned}\quad (2.34)$$

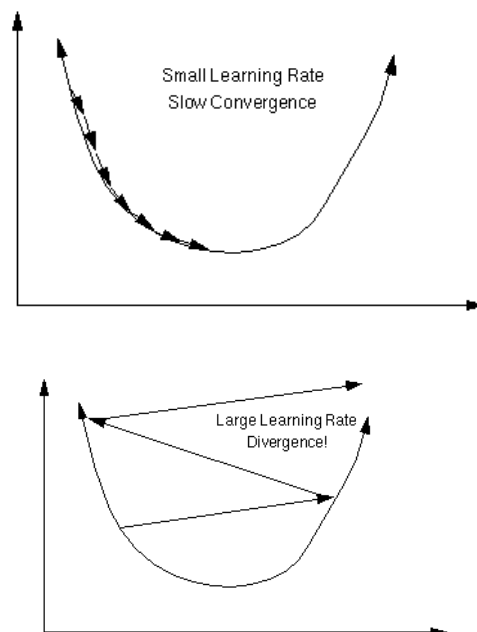
$\eta > 0$ tanulási paraméter

- b) a rejtett rétegekben a következő réteg értékeire támaszkodva számítjuk a módosult súlyokat:

$$\begin{aligned}\frac{\partial E}{\partial w_{i,j}} &= y_i \delta_j \\ \delta_j &= y_j(1-y_j) \sum_k \delta_k w_{j,k} \\ w_{l,j}^* &= w_{l,j} + \eta \delta_j y_l\end{aligned}\quad (2.35)$$

$\eta > 0$ tanulási ráta

A tanulási ráta megfelelő beállítása nagyon fontos, értékét a programozónak kell meghatároznia. Ha túl kicsinek választjuk, akkor az algoritmus lassan fog konvergálni a hiba minimuma felé. A túl nagy tanulási ráta hatására az algoritmus instabillá válhat, oszcillálhat.



15. ábra Tanulási ráta hatása

2.2.6. Backpropagation algoritmus momentummal [6], [7]

A backpropagation algoritmus momentummal alkalmas arra, hogy elkerüljük a hiba lokális minimumba történő beragadását. Ennek elkerülésére meg kell növelni a tanulási ráta értékét, viszont az instabil viselkedést okozhat. Az algoritmus előnye, hogy nagyobb tanulási rátát használhatunk a stabilitás megtartása mellett mint a backpropagation algoritmusnál. Momentum értékének 0 és 1 közötti értéket kell választani.

A módosult súlyok az alábbi összefüggésekkel számíthatóak:

a) kimeneti neuronok esetén:

$$\begin{aligned} w_{l,k}^*(t+1) &= w_{l,k}(t) + \mu(w_{l,k}(t) - w_{l,k}(t-1)) + \eta \delta_k y_l \\ \delta_k &= y_k(1 - y_k)(d_k - y_k) \end{aligned} \quad (2.36)$$

μ a momentum értéke

b) rejtett réteg neuronjai esetén:

$$\begin{aligned} w_{l,k}^*(t+1) &= w_{l,k}(t) + \mu(w_{l,k}(t) - w_{l,k}(t-1)) + \eta \delta_k y_l \\ \delta_j &= y_j(1 - y_j) \sum_k \delta_k w_{j,k} \end{aligned} \quad (2.37)$$

μ a momentum értéke

2.2.7. Backpropagation algoritmus adaptív tanulási paraméterrel [7]

Az eredeti backpropagation algoritmus során a tanulási ráta értéke a számítás során a programozó által meghatározott érték. A backpropagation algoritmus teljesítményét nagyban befolyásolja a tanulási ráta értéke.

Az algoritmus teljesítményét javítani lehet adaptív tanulási ráta alkalmazásával, mivel az optimális tanulási ráta a tanulási folyamat során változik.

Az algoritmus célja, hogy a lehető legnagyobb legyen a tanulási ráta a tanulás során a stabilitás megtartása mellett.

A tanulási ráta módosítása az alábbi folyamat szerint történik:

- Ha az új hiba egy előre definiált aránnyal meghaladja az utolsó iterációban számolt hibát (általában 1,04):
 - A súlyokat nem módosítjuk
 - A tanulási rátát csökkentjük meghatározott részére csökkentjük (általában 0,7)
- Ha az új hiba kisebb mint az utolsó iterációban számolt hiba

- o A súlyokat módosítjuk
 - o A tanulási rátát növelni fogjuk (általában 1,05 szörösére)
- Az új hiba nem kisebb mint a régi és nem haladja meg a meghatározott aránnyal
 - o A súlyokat módosítjuk
 - o A tanulási ráta változatlan marad

A leggyorsabb tanuló algoritmusok a Levenberg-Marquardt algoritmus illetve a kvázi Newton-módszer, viszont nagy hálózatok esetén (több 100 súly) kevésbé hatékonyak, mert nagy számítási időt és memóriát igényelnek.

Nagy hálózatok esetén a konjugált gradiens módszert vagy a resilient backpropagation algoritmust célszerű használni.

Túltanulás: A modell túl jól illeszkedik a tanuló példákra, ennek következtében nem tud jól általánosítani, azaz új, a tesztelés során nem tanult példák esetén nem működik jól a modell. Ha a neurális hálózat paramétereinek száma sokkal kisebb mint a tanuló adatok száma, akkor elég kicsi az esély a túltanulásra. A neurális hálózat alul illetve túltanulásának elkerülését tehát alapvetően a megfelelően megválasztott rejtett rétegbeli csomópontok számával biztosíthatjuk.

A túltanulás akkor jelentkezhet, ha korlátozott száma tanuló adat áll a rendelkezésünkre. Több módszer ismert ilyen esetekben is a túltanulás elkerülésére.

Korai leállás módszere (early stopping):

A módszer alkalmazása esetén a rendelkezésre álló adatokat 2 részre osztjuk, egy tanuló halmazra és egy validációs halmazra. A tanuló halmazon addig történik a tanítás, míg a validációs halmazon is csökken a hiba. Ha a tanuló halmazon a hálózat hibáját nagyon lecsökkentjük, a hálózat elkezd túltanulni a tanuló adatokat, és a validációs hiba növekedni fog.

Regularizáció:

A regulációs módszer lényege, hogy a hálózat hibájának csökkentése mellett minimalizálni próbáljuk a hálózat által megvalósított függvény gradiensét is. Ez az ötlet arra a tapasztalatra támaszkodik, hogy a túltanulás esetén a hálózat a mérési hibákat is megtanulva, nagy lokális gradienseket produkál az adatpontokban (hullámozás). Ezért azután a minimalizálandó hibafüggvényt kiegészítjük egy a gradiensek nagyságával arányos ún. regularizációs taggal. Ennek eredményeként a súlyok kisebbek lesznek, és csökken a túltanulás valószínűsége és a hálózat válasza simább lesz.

2.2.8. Newton-módszer [6], [7]

A Newton-módszer a másodrendű gradiens módszerek csoportjába tartozik, a hibafüggvény másodrendű deriváltjait használja fel a hálózat súlyainak meghatározására. A Newton-módszer négyzetes függvények minimumát 1 lépésben találja meg. Általában gyorsabb konvergenciát mutatnak mint a backpropagation algoritmus, de hátránya a nagyobb számítási igény (Hesse-mátrix számítása miatt), valamint előfordulhat oszcilláció vagy divergencia. Ezzel szemben a backpropagation algoritmus a tanulási ráta megfelelő választása esetén garantáltan konvergálni fog.

A Newton-módszer azt feltételezi, hogy a gradiens vektor komponensei a súlyok értékétől függenek, illetve a súlyok lineárisan függetlenek.

Tekintsük a hibafüggvény másodfokú Taylor-polinomját a w_k pontban!

$$q(w) = E(w_k) + \nabla E(w_k)(w - w_k) + \frac{1}{2}(w - w_k)H(w_k)(w - w_k) \quad (2.38)$$

$$H = \begin{pmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_N^2} \end{pmatrix} \quad (2.39)$$

A H a Hesse-mátrixot jelöli, amely az E hibafüggvény másodrendű deriváltjait tartalmazza.

Határozzuk meg $q(w)$ minimumát! A minimum szükséges feltétele:

$$\nabla q(w) = 0 \quad (2.40)$$

Elvégezve a gradiens-képzést az alábbi összefüggést kapjuk:

$$\begin{aligned} \nabla E(w_k) + H(w_k)(w - w_k) &= 0 \\ w_{k+1} &= w_k - H(w_k)^{-1} \nabla E(w_k) \end{aligned} \quad (2.41)$$

A Hesse-mátrix kiszámítása bonyolult, hosszadalmas, ezért olyan módszereket dolgoztak ki, amelyek a Hesse-mátrixok közelítik.

Ezeket a módszereket kvázi Newton-módszereknek nevezik. A kvázi Newton-módszereknél a másodrendű deriváltakat nem számítjuk ki, hanem a közelítésüket számoljuk ki egy egyszerű képlettel.

2.2.9. BFGS (Broyden-Fletcher-Goldfarb-Shanno) eljárás [6], [7]

Vezessük be az y_k jelölést a gradiensek különbségére, az s_k jelölést pedig a w súlyvektorok különbségére!

$$\begin{aligned} s_k &= w_{k+1} - w_k \\ y_k &= \nabla E(w_{k+1}) - \nabla E(w_k) \end{aligned} \quad (2.42)$$

Induló lépésként kiindulunk egy tetszőleges $w_k \in R^n$ vektorból és egy szimmetrikus pozitív definit B_k mátrixból.

A k . lépés az alábbiak szerint alakul:

1. Megoldjuk a $B_k s_k = -\nabla E(w_k)$ lineáris egyenletrendszert az s_k különbségvektorra.
2. Kiszámoljuk a w_{k+1} vektor értékét a $w_{k+1} = w_k + s_k$ összefüggés alapján.
3. Az alábbi formulákkal kiszámoljuk az y_k vektort és a Hesse mátrixot közelítő következő B_{k+1} mátrix értékét:

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{(B_k s_k)(B_k s_k)^T}{(B_k s_k)^T s_k} \quad (2.43)$$

2.2.10. Levenberg-Marquardt algoritmus [6], [7]

A Levenberg-Marquardt algoritmus az egyik leggyakrabban használt, leghatékonyabb tanuló algoritmus, amely sokkal hatékonyabb mint a backpropagation algoritmus. A módszer kiválóan alkalmas neurális hálózatok súlyainak beállítására. Az eljárás ötvözi a backpropagation eljárás és a Gauss-Newton módszer előnyeit: a backpropagation algoritmus stabilitását illetve a Gauss-Newton módszer gyors konvergenciáját.

Ezt az iterációs eljárást a legkisebb négyzetes függvényközelítési probléma megoldására használják, így használható nemlineáris függvények négyzetösszegének minimalizálására is.

$$E(w) = \frac{1}{2} \sum_{i=1}^m (f_i(w))^2 \rightarrow \min \quad (2.44)$$

A Levenberg-Marquardt algoritmus a Hesse mátrix közelítésére az alábbi összefüggést adja, így biztosítva a Hesse mátrix invertálhatóságát:

$$H \approx J^T J + \mu I \quad (2.45)$$

ahol J a Jacobi-mátrix, I egységmátrix

A súlyok új értéke az alábbi összefüggéssel számítható:

$$\begin{aligned} \underline{w}_{k+1} &= \underline{w}_k - (J_k^T J_k + \mu I)^{-1} J_k \underline{e}_k \\ \underline{e}_k &= \underline{y}_k - \underline{t}_k \end{aligned} \quad (2.46)$$

ahol $\underline{e}_k$ a hibavektor értéke a k. lépésben

A μ ún. regularizációs paraméter a keresés irányát és a módosítás nagyságát egyaránt befolyásolja. Ha a regularizációs paraméter kicsi értékű (közel 0), akkor az algoritmus a Gauss-Newton módszerhez konvergál, ha az értéke a végtelenbe tart, akkor pedig a legmeredekebb lejtő típusú megközelítést adja.

Az algoritmus k. iterációs lépése az alábbiak szerint alakul:

1. Adott $\underline{w}_k$ súlyok valamint a μ_k regularizációs paraméter

Inicializálásként tetszőleges pozitív μ értéket választhatunk

2. J_k és $\underline{e}_k$ meghatározása

3. $\underline{s}_k = -(J_k^T J_k + \mu I)^{-1} J_k \underline{e}_k$ kiszámítása

4. r_k az ún megbízhatósági régió számítása $r_k = \frac{E(\underline{w}_k) - E(\underline{w}_k - \underline{s}_k)}{E(\underline{w}_k) - \frac{1}{2} \|J_k \underline{s}_k + \underline{e}_k\|^2}$ (2.47)

5. a μ paraméter értékét az r_k értékétől függően állítjuk

o Ha $r_k < 0.25$ akkor $\mu_{k+1} = 4\mu_k$

o Ha $r_k > 0.75$ akkor $\mu_{k+1} = \frac{1}{2} \mu_k$

o Egyébként $\mu_{k+1} = \mu_k$

6. Ha $r_k \leq 0$ akkor $\underline{w}_{k+1} = \underline{w}_k$, különben $\underline{w}_{k+1} = \underline{w}_k + \underline{s}_k$

2.2.11. Konjugált gradiens módszer [6], [7]

A konjugált gradiens módszer egy iterációs módszer, amely alkalmazható többrétegű perceptron hálózatok tanító algoritmusaként.

Az átlagos négyzetes hiba $\underline{w}_k$ körüli Taylor-sora:

$$E(\underline{w}_k + \Delta \underline{w}_k) = E(\underline{w}_k) + g(\underline{w}_k)^T \Delta \underline{w}_k + \frac{1}{2} \Delta \underline{w}_k^T H(\underline{w}_k) \Delta \underline{w}_k + \dots \quad (2.48)$$

ahol $g(\underline{w})$ a gradiens vektor, $H(\underline{w})$ pedig a Hesse mátrix

A konjugált gradiens módszer alkalmazható, hiszen a fenti Taylor-sor alapján közelítőleg egy

$$f(x) = \frac{1}{2} x^T A x - b^T x + c \quad (2.49)$$

alakú kvadratikus képletet kell minimalizálni. Először kiszámítjuk a gradienst a backpropagation eljárással, majd a konjugált gradiens módszer segítségével úgy módosítjuk a súlyokat, mintha a közelítő kvadratikus forma minimumát keresnénk.

Az iterációs eljárás az alábbiak szerint írható le:

- 1) Inicializálás: Az első iterációban a minimum-keresés a legmeredekebb lejtő irányában (negatív gradiens) kezdődik.

$$\underline{p}_0 = -\underline{g}_0 \quad (2.50)$$

- 2) k. lépés:

- a. Keressük az új súlyokat

$$\underline{w}_k = \underline{w}_{k+1} + \eta_k \underline{p}_{k-1} \quad (2.51)$$

alakban. Az η_k úgy határozzuk meg, hogy $E(\underline{w}_k)$ minimális legyen a keresési irány mentén. Ez egy egyenes mentén való keresés, ami numerikusan megoldható.

- b. Ezután a $\underline{w}_k$ pontban megkeressük a hibafüggvény gradiensét.

Ha $|\underline{g}_k| \leq \delta |\underline{g}_0|$, ahol $\delta > 0$ előre megadott kis érték, akkor megállunk.

Ellenkező esetben folytatjuk a következő lépéssel, viszont a következő lépés nem a $-\underline{g}_k$ irányban történik, hanem a konjugált gradiens módszer szerint meghatározott irányban. A keresési irány akkor lesz konjugált, ha merőleges a korábbi gradiens-változásokra.

$$\underline{p}_k = -\underline{g}_k + \beta_k \underline{p}_{k-1} \quad (2.52)$$

A β_k számítása a Fletcher-Reeves-formula szerint:

$$\beta_k = \frac{\underline{g}_k^T \underline{g}_k}{\underline{g}_{k-1}^T \underline{g}_{k-1}} \quad (2.53)$$

A β_k számítása a Polak-Ribière-formula szerint:

$$\beta_k = \frac{\Delta \underline{g}_{k-1}^T + \underline{g}_k}{\underline{g}_{k-1}^T \underline{g}_{k-1}} \quad (2.54)$$

3. A feladat gyakorlati megvalósítása [8]

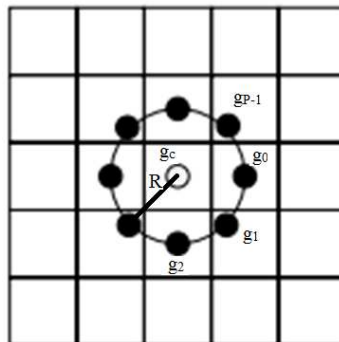
A talaj felismerő rendszer programjait Matlab-ban készítettem el. Ebben a fejezetben ismertetem a textúrák leírására használt módszereket, valamint az egyes klasszifikációs módszerekkel kapott eredményeket.

3. 1. Talaj Textúrájának leírása

Az egyes talajtípusokról készített képek textúrájának leírására a circular LBP-operátort (LBP(8,1)) valamint ennek egy továbbfejlesztett változatát az uniform LBP-t használtam.

3.1.1. Circular LBP(8,1)

Az LBP(8,1) esetén 8 környező pont intenzitásértékét, amelyek egy kör mentén helyezkednek el (kör sugara egy pixeltávolság) hasonlítjuk össze a középső pixel értékével, viszont mivel nem minden pont esik a pixel középpontjába, ezért ezen pontok intenzitásértékének (g_1, g_3, g_5, g_7) meghatározására bilineáris interpolációt alkalmazunk.



16. ábra LBP(8,1) pixeleinek elhelyezkedése

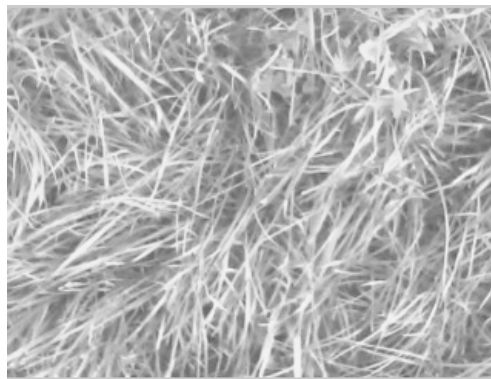
A LBP(8,1) kiszámításának a lépései az alábbiak:

- 1) A talajról készült kép beolvasása: A képek betöltésére a Matlab-ban az `imread(...)` függvény szolgál. A függvény visszatérési értéke egy képmátrix, amelynek mérete a kép típusától függ. Szürkeárnyaltos képek esetében egy $M \times N$ -es mátrixot kapunk., ahol M a vízszintes sorokban található pixelek számát, N a függőleges oszlopokban található képpontok számát jelöli. Színes képek esetében, ahol három színcsatorna van (piros, zöld, kék), ott egy $M \times N \times 3$ -as képmátrixot kapunk eredményül.



17. ábra Beolvasott színes kép

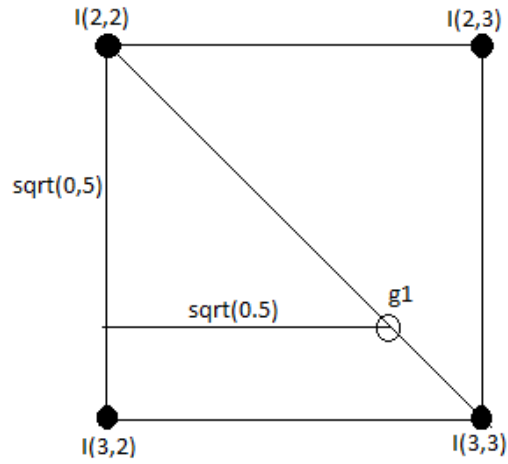
- 2) Szürkeárnyaltos kép előállítása: Ha a beolvasott kép színes volt, akkor átalakítjuk szürkeárnyaltossá (rgb2gray(...)) utasítás), mivel az LBP-operátort szürkeárnyaltos képeken fogjuk alkalmazni. A szürkeárnyaltos képeken 256 szürkeárnyalat szerepelhet, mivel 8-as bitmélységgel rendelkeznek.



18. ábra Szürkeárnyaltos kép

- 3) Az egyes pixelekhez tartozó LBP-értékek meghatározása: Az egyes pixelekhez tartozó eredményeket egy mátrixban tároljuk. Ha a szürkeárnyaltos képünk MxN-es méretű, akkor az eredmények tárolására szolgáló mátrixunk (M-2)x(N-2) méretű lesz, hiszen a 2-2 szélső sorban illetve oszlopban levő pixelekhez nem tudunk LBP-értéket számítani, mivel nem tudunk körük egy 1 pixeltávolságnyi sugarú kört rajzolni. A páros sorszámú (g_0, g_2, g_4, g_6) pixelek értékei könnyen meghatározhatóak, hiszen pixelközéppontra esnek. A páratlan sorszámú (g_1, g_3, g_5, g_7) pixelek értékének meghatározása bilineáris interpolációval történik. Például a 2. sor 2. pixeléhez tartozó g_1 értékének meghatározása az alábbiak szerint történik:

$$g_1 = I(2,2) + \sqrt{0,5}[I(2,3) - I(2,2)] + \sqrt{0,5}[I(3,2) - I(2,2)] + 0,5[I(2,2) - I(2,3) - I(3,2) + I(3,3)]$$

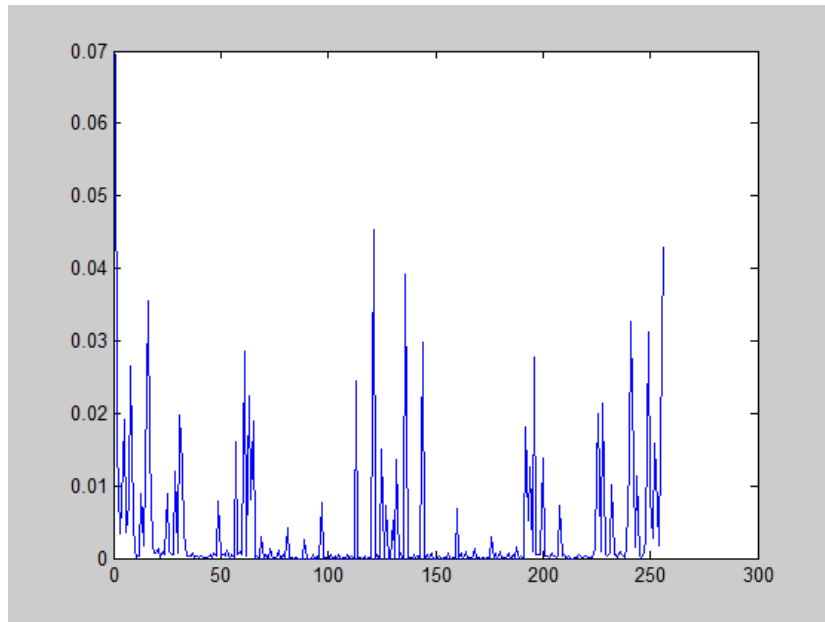


19. ábra Az $I(2,2)$ középső pixelhez tartozó g_1 értékének meghatározása

A 8 környező pont értékének kiszámítása után, összehasonlítjuk az értéküket a középső pont intenzitásával, és kiszámoljuk az adott ponthoz tartozó LBP-értéket:

$$LBP_{8,1}(x_c, y_c) = \sum_{p=0}^7 s(g_p - g_c) 2^p \quad (3.1)$$

Az LBP-mátrixot sorvektorra (256 elemű, mivel az LBP 8 bites, ezért 256 különböző értéket vehet fel) rendeztem a reshape(...) utasítással, majd a kapott eredményt normalizált LBP-hisztogramon ábrázoltam.



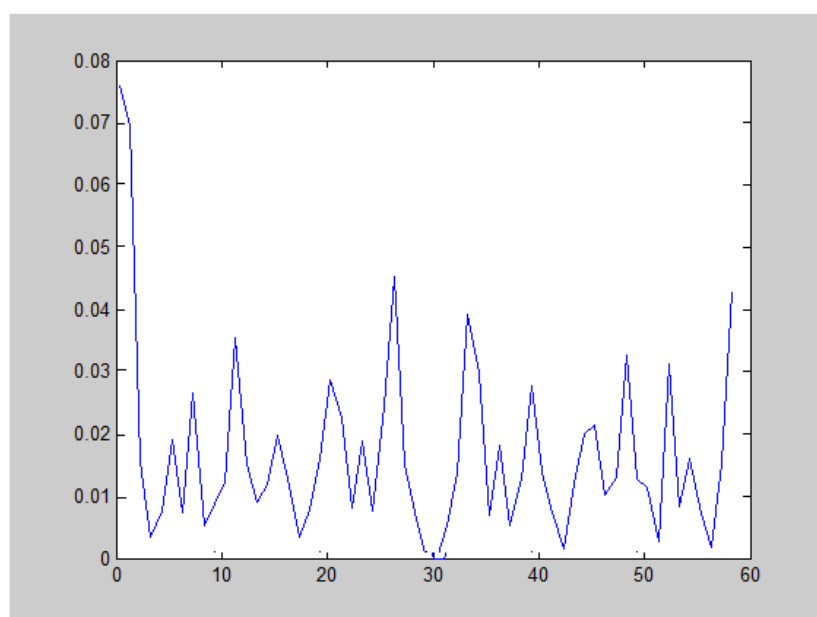
20. ábra Normalizált LBP(8,1)-hisztogram

3.1.2. Uniform LBP

Ojala 2002-ben megállapította, hogy LBP(8,1) esetén (8 a szomszédos pontok száma, a kör sugara 1 pixel hosszúságú) az előálló LBP-értékek közel 90 %-a uniform, így nem veszünk sok információt azzal, ha a nem uniform mintákat egyetlen címkével illetjük. Egy LBP-bitminta akkor uniform, ha a bitmintán körkörösén végigmenve legfeljebb 2 db 0-1 vagy 1-0 átmenetet kapunk. Pl: 00000000 (0 átmenet) 01110000 (2 átmenet) and 11001111 (2 átmenet) uniformok, de pl. 11001001 (4 átmenet) and 01010010 (6 átmenet) nem azok.

LBP(8,1) esetén a 256 lehetséges bitmintából 58 lesz uniform, így a textúra jellemzése uniform LBP esetén 59 elemű vektorral történik, szemben az LBP(8,1) esetén alkalmazott 256 elemű vektorral.

Az uniform LBP meghatározásához a már kiszámított LBP(8,1) vektorból indultam ki. Mind a 256 bitmintára meghatároztam az 0-1 és 1-0 átmenetek összegét. A 2-nél több átmenetű bitmintához tartozó minták számát összegeztem, és az 59 elemű vektor első elemében eltároltam, tehát az 59 elemű vektor első eleme tartalmazza a nem uniform mintákat. Az utána következő 58 elem tartalmazza az uniform mintákat. A vektor 2. eleme tartalmazza a 00000000 bitsorozathoz tartozó minták számát, 3. eleme 00000001 bitsorozathoz tartozó minták számát, egészen az utolsó elemig, ami 11111111 bitsorozathoz tartozó minták számát mutatja. A kapott eredményt normalizáltam, és hisztogramon ábrázoltam. Ojala megállapításához hasonlóan az eredményeimnél szintén csupán 10 % körüli az uniform minták száma.



21. ábra Normalizált uniform LBP-hisztogram

3. 2. Klasszifikációs eljárások

A tesztadatok osztályozására 2 klasszifikációs eljárást próbáltam ki, az SVM-mel illetve előrecsatolt neurális hálózattal történő osztályozást. Ebben a fejezetben áttekintem ezen módszerek paramétereinek beállítását, valamint az osztályozás során kapott eredményeket.

3.2.1. Tanító pontok előállítása

Mindhárom osztály (fű, homok, aszfalt) 25-25 db tanító képet rendeltem. Mindegyik képhez kiszámoltam a 256 elemű LBP(8,1) vektorokat illetve 59 elemű uniform LBP vektorokat, és egy-egy mátrixban eltároltam, így egy 75x256 valamint egy 75x59 méretű mátrixot kaptam, amiket elmentettem egy-egy xls kiterjesztésű fájlba.

3.2.2. Osztályozás többosztályú SVM-mel

A tesztképek osztályba sorolására többosztályú SVM módszert kell alkalmazni, mivel 3 osztályunk van. A fejezetben ismertetett módszerek közül az „egy a többi ellen” módszert alkalmaztam.

Mindhárom osztályhoz tartozik egy-egy bináris osztályozó, amely a tesztképet az adott osztályba sorolja be vagy az összes többi osztályba. (Például a fűhöz tartozó osztályozó esetén a fű alkotja az egyik osztályt, a másik osztályt az aszfalt és a homok együttesen alkotják.) A tesztadat abba az osztályba fog tartozni, amelyik a legtöbb szavazatok kapta. A módszer hátránya, hogy abban az esetben, ha több osztály is azonos számú szavazatot kapott, akkor nem tudjuk a tesztadatot osztályba sorolni.

A tesztképek osztályozásához készítettem egy grafikus felhasználói felületet (GUI) a Matlab-ban. A 22. ábrán láthatjuk a program kinézetét.

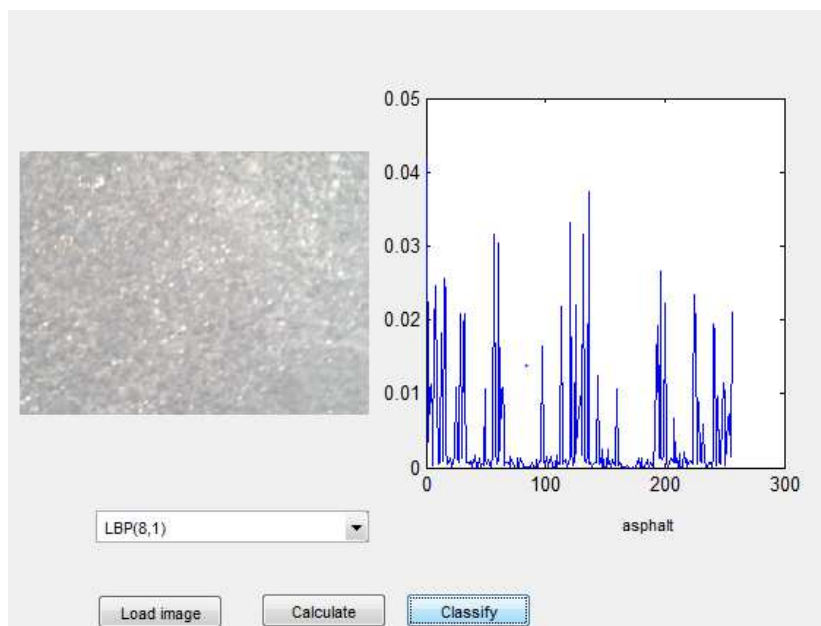
A 'Load image' nyomógomb megnyomásával egy, a számítógépünkön mentett tesztképet tudunk beolvasni, amely a jobb oldali mezőben meg fog jelenni.

A legördülő menüből ki tudjuk választani, hogy melyik LBP-operátort használjuk majd az osztályozásra. A 2 választási lehetőségünk az LBP(8,1) valamint az uniform LBP.

A 'Calculate' nyomógomb megnyomásával a program kiszámítja a beolvasott képhez tartozó LBP-vektor értékét, és a bal oldali mezőben normalizált hisztogramon ábrázolja.

A 'Classify' nyomógommbal végezhetjük el a tesztképünk osztályozását. Az `xlsread(...)` utasítással beolvassuk az adott LBP-operátorhoz tartozó 75 elemű tanító adathalmazt, amit korábban egy xls fájlban tároltunk el. A tesztkép osztályozásához szükség van a tesztminták osztályainak beolvasására is, amit egy 75 elemű oszlopvektorban tároltam. Az osztályozásnál 4 eset lehetséges (fű, homok, aszfalt, szavazás döntetlen eredménye miatt nem eldönthető).

Az osztályozással kapott eredményt a normalizált hisztogram alatt elhelyezkedő statikus szövegdoboz jeleníti meg.



22. ábra Az SVM osztályozó program kinézete

75 tesztmintán (25 fű, 25 homok, 25 aszfalt) megvizsgáltam az osztályozó pontosságát. A kapott eredményeket a 2. táblázatban rögzítettem. Az eredményeket összehasonlítva látható, hogy puha margójú SVM alkalmazásával nagyobb pontosságot lehet elérni. A puha margójú SVM abban különbözik az eredeti SVM-től, hogy az osztályozás feltételeit kiegészítő változókkal gyengítjük, így megengedjük, hogy néhány tanítópont a margón belülre kerüljön. A felhasználó által beállítandó C regularizációs paraméterrel a kiegészítő változók súlyát adjuk meg. Az osztályozás során több regularizációs paramétert is kipróbáltam, és a regularizációs paraméter 0,5 értékénél kaptam a legnagyobb pontosságot.

Osztályozás pontossága	Többosztályú SVM	Többosztályú puha margójú SVM C=0,05	Többosztályú puha margójú SVM C=0,5
Fű	96%	96%	96%
Homok	80%	84%	88%
Aszfalt	88%	88%	92%

2. táblázat Osztályozás pontossága többosztályú SVM esetén

3.2.3. Osztályozás előrecsatolt neurális hálózattal

A tesztképek osztályba sorolása előrecsatolt neurális hálózattal is megvalósítható. Uniform LBP használata esetén a bemeneti vektor 59 elemű lesz, a kimeneti vektor pedig 3 elemű, hiszen 3 osztályunk van.

A tanítási halmazunk 75 mintából áll (3.2.1. fejezet).

Hálózat tanítása

A hálózat tanítása történhet egyenként (szekvenciális tanítás) vagy epochonként (kötegelt (batch) tanítás). Soros tanítás esetén a súlymódosításokat tanító pontonként, kötegelt tanítás esetén a teljes tanító készlet (epoch) felhasználása után végezzük. Az első esetben tehát a súlymódosítások száma megegyezik a tanító pontok számának és a tanítási ciklusok számának szorzatával, míg a batch eljárásnál csak ciklusonként korrigálunk, ahol egy ciklus a teljes tanító készlet egyszeri felhasználását jelenti. A szekvenciális tanítás előnye, hogy kevesebb tárolási helyet igényel, valamint könnyű implementálni.

A feladat során a szekvenciális tanítást használtam, a hálózat hibájának számítására az átlagos négyzetes hibát választottam.

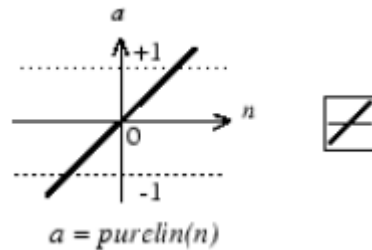
Aktivációs függvény kiválasztása

A neurális hálózat teljesítőképessége szempontjából fontos, hogy megfelelő aktivációs függvényt válasszunk ki. Általában gyorsabb a tanulás, ha az aktivációs függvény antiszimmetrikus. ($\varphi(-n) = -\varphi(n)$)

A Matlab programban az alábbi aktivációs függvények közül lehet választani:

- Lineáris aktivációs függvény

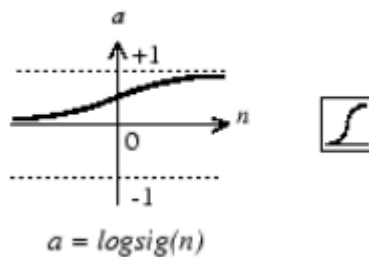
$$\varphi(n) = n \quad (3.2)$$



23. ábra Lineáris aktivációs függvény

- Logisztikus aktivációs függvény

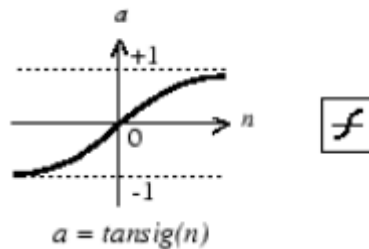
$$\varphi(n) = \frac{2}{1 + e^{-n}} \quad (3.3)$$



24. ábra Logisztikus aktivációs függvény

- Tangens hiperbolikus aktivációs függvény

$$\varphi(n) = \frac{2}{1 + e^{-2n}} - 1 \quad (3.4)$$



25. ábra Tangens hiperbolikus aktivációs függvény

A 3. táblázatban a különböző aktivációs függvényekkel kapott eredményeket ismertetem. Azt vizsgáltam meg, hogy 0,1 értékű tanulási paraméter mellett 1000 epoch tanulási ciklus alatt

milyen értékűre csökken le az átlagos négyzetes hiba backpropagation tanulási algoritmus alkalmazásával. Az átlagos négyzetes hiba meghatározásánál 10 futtatás eredményének átlagát vettem. Az eredményekből jól látszik, hogy a leggyorsabb tanítást a rejtett réteg(ek)ben tangens hiperbolikus aktivációs függvény alkalmazásával lehet elérni. A kimeneti réteg(ek)ben lineáris illetve tangens hiperbolikus aktivációs függvény is jó megoldásnak mutatkozik.

	Rejtett réteg	Kimeneti réteg	Rejtett réteg	Kimeneti réteg	Rejtett réteg	Kimeneti réteg	Rejtett réteg	Kimeneti réteg	Rejtett réteg	Kimeneti réteg	Rejtett réteg	Kimeneti réteg
rejtett réteg neuronjainak száma	Logisztikus	Lineáris	Logisztikus	Logisztikus	Logisztikus	tangens hip.	tangens hip.	Lineáris	Tangens hip.	Logisztikus	Tangens hip.	Tangens hip.
1	0,1919		0,2731		0,2492		0,1514		0,277		0,21	
3	0,1716		0,2597		0,1844		0,0823		0,2434		0,1001	
5	0,16		0,2567		0,1518		0,073		0,2442		0,0672	
7	0,1477		0,249		0,1225		0,0678		0,2409		0,063	
10	0,1278		0,248		0,1189		0,0635		0,2432		0,0604	
5+5	0,1538		0,2561		0,1453		0,0513		0,2425		0,0558	
10+10	0,1242		0,2495		0,1239		0,0452		0,2387		0,0353	

3. táblázat a hálózat átlagos négyzetes hibájának értékei különböző aktivációs függvényekkel 1000 epoch tanulási ciklus után (tanulási paraméter 0,1)

A neurális hálózat ideális méretének meghatározása

A neurális hálózat ideális méretének meghatározására még nem dolgoztak ki általános módszert, annak meghatározása próbálgatásos úton lehetséges. Általánosan elmondható, hogy a több réteg alkalmazása könnyítheti a feladat megoldását, illetve rétegenként kevesebb neuron is elegendő lehet. Kétféle módszer terjedt el a hálózat méretének meghatározására. Az első szerint egy nagyobb hálózattól kiindulva próbáljuk meg csökkenteni annak méretét, a másik szerint egy kisebb hálózat bővítésével próbálunk eljutni a megfelelő megoldásig.

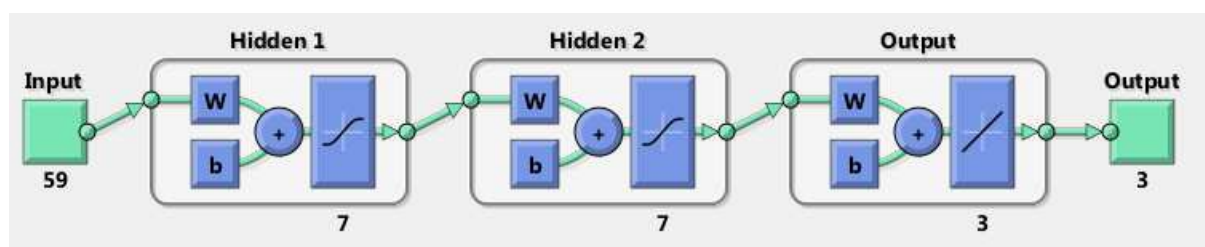
A neurális hálózat rejtett rétegeinek illetve neuronjainak ideális számának meghatározásához a 75 elemű tanító adathalmazt 2 részre osztottam: 80 %-a a hálózat tanítására szolgál, 20 %-a validációs minta. A túltanulás elkerülésére a hálózat tanítása le fog állni, ha a validációs minta hibája 10 egymást követő epochban növekedni fog. Megvizsgáltam, hogy a különböző hálózati struktúrákkal (rejtett rétegek száma, neuronok száma) mekkora értékűre lehet csökkenteni a validációs hibát (4. táblázat). A legkisebb validációs hibát 7-7 neuronból álló 2 rejtett rétegű neurális hálózathoz kaptam.

Rejtett réteg neuronjainak száma	Validációs hiba
1	0,1602
3	0,1216
5	0,0891
7	0,0787
10	0,0601
5+5	0,0510
7+7	0,0480
10+10	0,0514
15+15	0,0642

4. táblázat a validációs hiba értéke az egyes hálózati struktúráknál

A választott neurális hálózat struktúrája

A neurális hálózat 59 bemenettel és 3 kimenettel rendelkezik. 2 rejtett rétegében egyaránt 7-7 neuron található, a kimeneti rétegben pedig 3 neuron. A rejtett réteg neuronjainak tangens hiperbolikus aktivációs függvényük, a kimeneti réteg neuronjainak lineáris aktivációs függvényük van.



26. ábra A választott neurális hálózat struktúrája

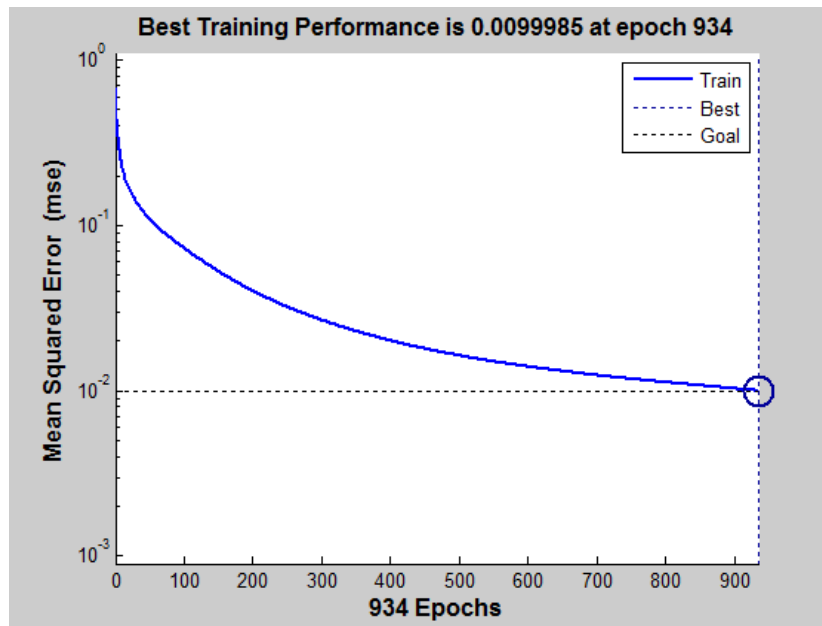
Tanulási algoritmusok összehasonlítása

A 7-7 neuronból álló 2 rejtett rétegű neurális hálózat tanítását többféle tanulási algoritmussal elvégeztem. Megvizsgáltam, hogy a különböző tanulási algoritmusokkal mennyi idő és hány epochszám alatt lehet elérni 0,01 értékű átlagos négyzetes hibát (5. táblázat).

Tanulási algoritmus	epochszám	Tanulási idő
Backpropagation algoritmus tanulási paraméter 0,1	896,6	9,9896
Backpropagation algoritmus tanulási paraméter 0,15	687,2	8,472
Backpropagation algoritmus tanulási paraméter 0,5	219,2	2,4778
Backpropagation algoritmus momentummal tanulási paraméter 1 momentum konstans 0,7	185,2	2,085
Levenberg-Marquardt Algoritmus	7	8,9872
Fletcher-Reeves konjugált gradiens módszer	44	1,9586

5. táblázat a tanulási algoritmusok eredményei

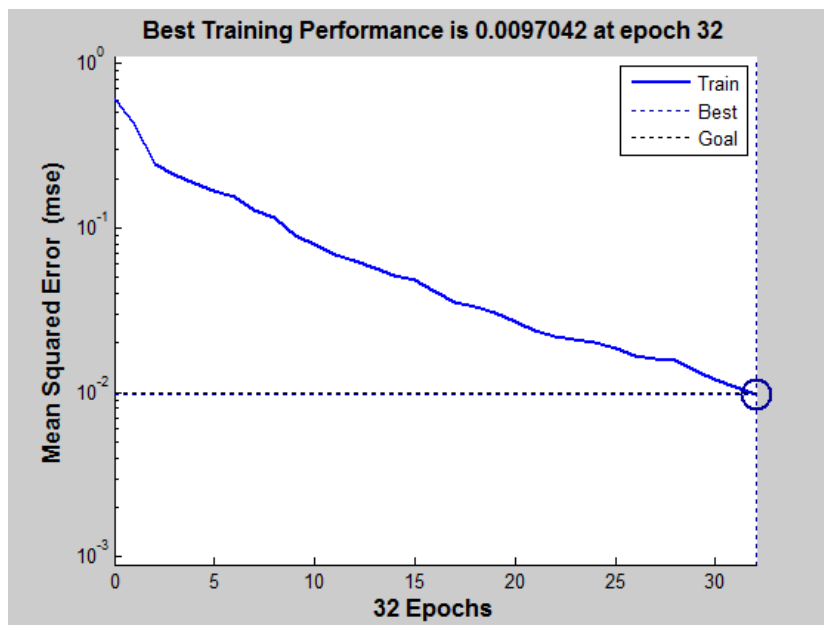
A backpropagation algoritmusnál a tanulási paraméter értéke nagy mértékben befolyásolja az algoritmus sebességét, Ha a tanulási paraméter optimális értékét próbálgatással lehet meghatározni. Ha a tanulási paraméter túl kicsire választjuk, akkor a súlyok kis mértékben fognak változni, lassú lesz a konvergencia. Nagy tanulási paraméter esetén gyorsabb lesz a konvergencia, de instabilitás, oszcillációs léphet fel. Momentum konstans hozzáadásával növelni lehet a tanulás gyorsaságát, mivel a momentum aluláteresztő szűrőként viselkedve csökkenni az oszcillációt, így nagyobb tanulási paraméterekre is stabilitást biztosít. Az eredményekből látható, hogy a backpropagation algoritmus nagy epochszám után éri el a 0,01 értékű átlagos négyzetes hibát, hiszen a hibafüggvény elsőrendű deriváltjait használja a hálózat súlyainak beállítására (27. ábra).



27. ábra Átlagos négyzetes hiba alakulása backpropagation algoritmussal (tanulási paraméter 0,1)

A Levenberg-Marquardt algoritmusnál lehet a legkisebb epochszám alatt elérni a 0,01 értékű átlagos négyzetes hibát. Ez annak köszönhető, hogy ez az algoritmus a másodrendű deriváltakat használja a tanítás során. A hálózat tanulási ideje magas, mivel az algoritmus során számítandó Jacobi-mátrix nagy, 75x500-as méretű. (75 a tanító minták száma, 500 a hálózat súlyainak száma.)

A konjugált gradiens módszer a backpropagation algoritmushoz hasonlóan elsőrendű gradiens módszer, viszont a keresés nem a negatív gradiens irányában történik, hanem a konjugált gradiens módszer szerint meghatározott irányokban. A backpropagation algoritmushoz képest a tanulás gyorsabb lesz, valamint kisebb epochszám alatt éri el a hibafüggvény megfelelő mértékű csökkenését (28. ábra). A vonalmenti minimumkeresésre az aranymetszés keresést választottam. Ez az eljárás egyike a leggyakrabban használtaknak. Előnye, hogy minden lépésben csak egy olyan új pont adódik, ahol ki kell kiszámolni a függvényértéket.



28. ábra Átlagos négyzetes hiba alakulása Fletcher-Reeves konjugált gradiens módszer

A neurális hálózat osztályozásának pontossága

A hálózat tanítására a 75 képből álló tanító mintahalmazt használtam fel. A tanító halmazt 2 részre osztottam, 80%-át a hálózat tanítására, 20 %-át validációra használtam. A hálózat tanítása le fog állni, ha a validációs minta hibája 10 egymást követő epochban növekedni fog. A hálózat tanítására Fletcher-Reeves konjugált gradiens módszert alkalmaztam, mert ez biztosította a leggyorsabb tanulást.

Az osztályozás pontosságát az SVM-nél is használt 75 tesztkép segítségével határoztam meg. A kapott eredményeket a 6. táblázat tartalmazza, amiket 10 tanítás során kapott eredmények átlagaként számítottam.

Osztályozás pontossága	59-7-7-3 struktúrájú neurális hálózat
fű	96,4%
homok	83,6%
aszfalt	87,2%

6. táblázat az osztályozás pontossága 59-7-7-3 struktúrájú neurális hálózattal

Irodalomjegyzék

- [1] Fazekas Gábor, Hajdu András: Képfeldolgozási módszerek egyetemi jegyzet, Debreceni Egyetem, Informatikai Intézet, 2004, 2-4. old.
- [2] M. Pietikäinen, A. Hadid, G. Zhao, T. Ahonen: Computer Vision Using Local Binary Patterns, Springer-Verlag, 2011 3-19. old, 32-33. old.
- [3] SVM https://dms.sztaki.hu/sites/dms.sztaki.hu/files/file/2013/svm_ea.pdf letöltés ideje: 2014.05.12.
- [4] Ispány Márton, Jeszenszky Péter: Tartóvektorgépek
<http://www.inf.unideb.hu/valseg/dolgozok/ispany/DataMine/Alkalmazasok/svm.pdf> letöltés ideje: 2014.05.12.
- [5] Dr. Nagy Tamás: Lagrange dualitás, Miskolci Egyetem, Alkalmazott Matematikai Tanszék <http://www.unimiskolc.hu/~matente/oktatasi%20tananyagok/LAGRANGE%20DUALITAS.pdf> letöltés ideje: 2014.05.13.
- [6] Fazekas István: Neurális hálózatok, Debreceni Egyetem, Informatikai Kar, 2013, 15-44. old.
- [7] M. T. Hagan, H. B. Demuth, M. Beale: Neural Network Design, PWS Publishing Company, 1996 Chapter 2, 4, 11, 12
- [8] M. Demuth, M. Beale: Neural Network Toolbox for Use with Matlab 5.14-5.30

Uniform LBP kódja:

```
I=imread('E:/kep1.jpg');
m=size(I,1);
n=size(I,2);
I2= double(rgb2gray(I));
C=C+1;

for i=2:m-1
    for j=2:n-1
        J0=I2(i,j);

        %gc
        d0=I2(i,j+1)>J0;          %g0
        dx1=I2(i,j+1)-I2(i,j);
        dy1=I2(i+1,j)-I2(i,j);
        dxy1=I2(i,j)-I2(i,j+1)-I2(i+1,j)+I2(i+1,j+1);
        d1=(J0+0.5*sqrt(2)*dx1+0.5*sqrt(2)*dy1+0.5*dxy1)>J0;
        %g1
        d2=I2(i+1,j)>J0;          %g2
        dx3=I2(i,j)-I2(i,j-1);
        dy3=I2(i+1,j-1)-I2(i,j-1);
        dxy3=I2(i,j-1)-I2(i,j)-I2(i+1,j-1)+I2(i+1,j);
        d3=(I2(i,j-1)+(1-0.5*sqrt(2))*dx3+0.5*sqrt(2)*dy3+(1-
0.5*sqrt(2))*0.5*sqrt(2)*dxy3)>J0;          %g3
        d4=I2(i,j-1)>J0;
        %g4

        dx5=I2(i-1,j)-I2(i-1,j-1);
        dy5=I2(i,j-1)-I2(i-1,j-1);
        dxy5=I2(i-1,j-1)-I2(i-1,j)-I2(i,j-1)+I2(i,j);
        d5=(I2(i-1,j-1)+(1-0.5*sqrt(2))*dx5+(1-0.5*sqrt(2))*dy5+(1-0.5*sqrt(2))*(1-
0.5*sqrt(2))*dxy5)>J0;          %g5
        d6=I2(i-1,j)>J0;
```

```

dx7=I2(i-1,j+1)-I2(i-1,j);
dy7=I2(i,j)-I2(i-1,j);
dxy7=I2(i-1,j)-I2(i-1,j+1)-I2(i,j)+I2(i,j+1);

%g6
d7=(I2(i-1,j)+0.5*sqrt(2)*dx7+(1-0.5*sqrt(2))*dy7+0.5*sqrt(2)*dxy7)>J0;
%g7
LBP(i-1,j-1)=d0*2^0+d1*2^1+d2*2^2+d3*2^3+d4*2^4+d5*2^5+d6*2^6+d7*2^7;

end
end
LBP=uint8(LBP);
LBPv=reshape(LBP,1,size(LBP,1)*size(LBP,2));
Hist=hist(LBPv,0:255);
imshow(LBP);
plot(Hist);

%table létrehozása
table = zeros(1, 256);

nextLabel = 1;
for k = 1:256
    bits=de2bi(k-1,8);
    changes=sum(abs(diff(bits)));
    if bits(8)-bits(1)~=0
        changes=changes+1;
    end
    if changes <= 2
        table(k) = nextLabel;
        nextLabel = nextLabel + 1;
    else

```

```

        table(k) = 0;

    end

end

%LBP-ból LBP uniform

for i=1:1:256
    x=table(i); if x==0
        LBPu(1)=LBPu(1)+Hist(i);
    else
        LBPu(x+1)=Hist(i);
    end
end
end

```

SVM osztályozó GUI kódja:

```
function varargout = LBPhistogram1(varargin)
%LBPHISTOGRAM1 M-file for LBPhistogram1.fig
%   LBPHISTOGRAM1, by itself, creates a new LBPHISTOGRAM1 or raises
the existing
%   singleton*.
%
%   H = LBPHISTOGRAM1 returns the handle to a new LBPHISTOGRAM1
or the handle to
%   the existing singleton*.
%
%   LBPHISTOGRAM1('Property','Value',...) creates a new
LBPHISTOGRAM1 using the
%   given property value pairs. Unrecognized properties are passed via
%   varargin to LBPhistogram1_OpeningFcn. This calling syntax produces a
%   warning when there is an existing singleton*.
%
%   LBPHISTOGRAM1('CALLBACK') and
LBPHISTOGRAM1('CALLBACK',hObject,...) call the
%   local function named CALLBACK in LBPHISTOGRAM1.M with the
given input
%   arguments.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help LBPhistogram1

% Last Modified by GUIDE v2.5 02-Nov-2014 09:37:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @LBPhistogram1_OpeningFcn, ...
    'gui_OutputFcn', @LBPhistogram1_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
```

```

    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before LBPhistogram1 is made visible.
function LBPhistogram1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for LBPhistogram1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes LBPhistogram1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = LBPhistogram1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
handles.output = hObject;
[fn pn] = uigetfile('*.jpg','select jpg file');

```

```

complete = strcat(pn,fn);
set(handles.edit1,'string',complete);
I = imread(complete);
axes(handles.axes1);
imshow(I,[]);
guidata(hObject, handles);
set(hObject,'UserData',I);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
h = findobj('Tag','pushbutton1');
I= get(h,'UserData');

m=size(I,1);
n=size(I,2);
I2= double(rgb2gray(I));

switch get(handles.popupmenu1,'Value')
case 1

for i=2:m-1
    for j=2:n-1
        J0=I2(i,j);
        d0=I2(i,j+1)>J0; %g0 %gc
        dx1=I2(i,j+1)-I2(i,j);
        dy1=I2(i+1,j)-I2(i,j);
        dxy1=I2(i,j)-I2(i,j+1)-I2(i+1,j)+I2(i+1,j+1);
        d1=(J0+0.5*sqrt(2)*dx1+0.5*sqrt(2)*dy1+0.5*dxy1)>J0;
    %g1
        d2=I2(i+1,j)>J0; %g2
        dx3=I2(i,j)-I2(i,j-1);
        dy3=I2(i+1,j-1)-I2(i,j-1);
        dxy3=I2(i,j-1)-I2(i,j)-I2(i+1,j-1)+I2(i+1,j);
        d3=(I2(i,j-1)+(1-0.5*sqrt(2))*dx3+0.5*sqrt(2)*dy3+(1-
0.5*sqrt(2))*0.5*sqrt(2)*dxy3)>J0; %g3 %g4
        d4=I2(i,j-1)>J0;
        dx5=I2(i-1,j)-I2(i-1,j-1);
        dy5=I2(i,j-1)-I2(i-1,j-1);
        dxy5=I2(i-1,j-1)-I2(i-1,j)-I2(i,j-1)+I2(i,j);
        d5=(I2(i-1,j-1)+(1-0.5*sqrt(2))*dx5+(1-0.5*sqrt(2))*dy5+(1-0.5*sqrt(2))*
0.5*sqrt(2))*dxy5)>J0; %g5
        d6=I2(i-1,j)>J0;

```

```

    dx7=I2(i-1,j+1)-I2(i-1,j);
    dy7=I2(i,j)-I2(i-1,j);
    dxy7=I2(i-1,j)-I2(i-1,j+1)-I2(i,j)+I2(i,j+1);
    %g6
    d7=(I2(i-1,j)+0.5*sqrt(2)*dx7+(1-0.5*sqrt(2))*dy7+0.5*sqrt(2)*dxy7)>J0;
    %g7
    LBP(i-1,j-
1)=d0*2^0+d1*2^1+d2*2^2+d3*2^3+d4*2^4+d5*2^5+d6*2^6+d7*2^7;

    end
end
LBP=uint8(LBP);
LBPv=reshape(LBP,1,size(LBP,1)*size(LBP,2));
Hist=hist(LBPv,0:255);
Hist=Hist./sum(Hist);
axes(handles.axes2);
plot(Hist);
set(hObject,'UserData',Hist);

case 2

for i=2:m-1
    for j=2:n-1
        J0=I2(i,j);
        d0=I2(i,j+1)>J0;
        dx1=I2(i,j+1)-I2(i,j);
        dy1=I2(i+1,j)-I2(i,j);
        dxy1=I2(i,j)-I2(i,j+1)-I2(i+1,j)+I2(i+1,j+1);
        d1=(J0+0.5*sqrt(2)*dx1+0.5*sqrt(2)*dy1+0.5*dxy1)>J0;
        %g1
        d2=I2(i+1,j)>J0;
        dx3=I2(i,j)-I2(i,j-1);
        dy3=I2(i+1,j-1)-I2(i,j-1);
        dxy3=I2(i,j-1)-I2(i,j)-I2(i+1,j-1)+I2(i+1,j);
        d3=(I2(i,j-1)+(1-0.5*sqrt(2))*dx3+0.5*sqrt(2)*dy3+(1-
0.5*sqrt(2))*0.5*sqrt(2)*dxy3)>J0;
        d4=I2(i,j-1)>J0;
        dx5=I2(i-1,j)-I2(i-1,j-1);
        dy5=I2(i,j-1)-I2(i-1,j-1);
        dxy5=I2(i-1,j-1)-I2(i-1,j)-I2(i,j-1)+I2(i,j);
        d5=(I2(i-1,j-1)+(1-0.5*sqrt(2))*dx5+(1-0.5*sqrt(2))*dy5+(1-0.5*sqrt(2))*
0.5*sqrt(2))*dxy5)>J0;
        d6=I2(i-1,j)>J0;

```

```

dx7=I2(i-1,j+1)-I2(i-1,j);
dy7=I2(i,j)-I2(i-1,j);
dxy7=I2(i-1,j)-I2(i-1,j+1)-I2(i,j)+I2(i,j+1);
%g6
d7=(I2(i-1,j)+0.5*sqrt(2)*dx7+(1-0.5*sqrt(2))*dy7+0.5*sqrt(2)*dxy7)>J0;
%g7
LBP(i-1,j-
1)=d0*2^0+d1*2^1+d2*2^2+d3*2^3+d4*2^4+d5*2^5+d6*2^6+d7*2^7;

end
end
LBP=uint8(LBP);
LBPv=reshape(LBP,1,size(LBP,1)*size(LBP,2));
Hist=hist(LBPv,0:255);
%table létrehozása
table = zeros(1, 256);
ch=zeros(1,256);
nextLabel = 1;
for k = 1:256
    bits=de2bi(k-1,8);
    changes=sum(abs(diff(bits)));
    if bits(8)-bits(1)~=0
        changes=changes+1;
    end
    if changes <= 2
        table(k) = nextLabel;
        nextLabel = nextLabel + 1;
        ch(k)=changes;
    else
        table(k) = 0;
        ch(k)=changes;
    end
end
%LBP-ből LBP uniform

for i=1:1:256
x=table(i);
if x==0
LBPu(1)=LBPu(1)+Hist(i);
else
LBPu(x+1)=Hist(i);
end
end
end

```

```

axes(handles.axes2);
LBPu=LBPu./sum(LBPu);
set(hObject,'UserData',LBPu);
plot(LBPu);

```

```

end

```

```

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents
as cell array
%    contents{get(hObject,'Value')} returns selected item from popupmenu1

```

```

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%    str2double(get(hObject,'String')) returns contents of edit1 as a double

```

```

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
t= cputime;

switch get(handles.popupmenu1,'Value')
case 1
    TrainingSet = xlsread('E:/circularLBP256norm.xls');
h = findobj('Tag','pushbutton2');
    Hist= get(h,'UserData');
    TestSet=zeros(1,256);
    TestSet(1,:)=Hist ;
    GroupTrain= xlsread('E:/GroupTrain.xls');
    u=unique(GroupTrain);
    numClasses=length(u);
    result = zeros(length(TestSet(:,1)),1);

%build models
for k=1:numClasses
    %Vectorized statement that binarizes Group
    %where 1 is the current class and 0 is all other classes
    G1vAll=(GroupTrain==u(k));
    models(k) = svmtrain(TrainingSet,G1vAll,'boxconstraint',0.1);
end

x=zeros(length(TestSet(:,1)),numClasses);
y = zeros(length(TestSet(:,1)),1);

```

```

%classify test cases
for j=1:size(TestSet,1)
    for k=1:numClasses

        x(j,k)=svmclassify(models(k),TestSet(j,:));

        if sum(x(j,:))==1
            if x(j,k)==1;
                result(j)=k;

                end
            else

                result(j)=0;

        end

    end

end

disp(Hist(1));
disp(x);

if result==1
    disp('grass');
elseif result==2
    disp('sand');
elseif result==3
    disp('asphalt');
else
    disp('nonclassified');
end

e= cputime-t;
disp(e);

```

```

case 2
TrainingSet = xlsread('E:/uniformLBPnorm.xls');
h = findobj('Tag','pushbutton2');
LBPu= get(h,'UserData');
TestSet=zeros(1,59);
TestSet(1,:)=LBPu;
GroupTrain= xlsread('E:/GroupTrain.xls');
u=unique(GroupTrain);
numClasses=length(u);
result = zeros(length(TestSet(:,1)),1);

%build models
for k=1:numClasses
    %Vectorized statement that binarizes Group
    %where 1 is the current class and 0 is all other classes
    G1vAll=(GroupTrain==u(k));
    models(k) = svmtrain(TrainingSet,G1vAll,'boxconstraint',0.1);
end

x=zeros(length(TestSet(:,1)),numClasses);
y = zeros(length(TestSet(:,1)),1);

%classify test cases
for j=1:size(TestSet,1)
    for k=1:numClasses

        x(j,k)=svmclassify(models(k),TestSet(j,:));

        if sum(x(j,:))==1
            if x(j,k)==1;
                result(j)=k;

            end
        else

            result(j)=0;

        end
    end
end

```

```
end
```

```
end
```

```
disp(TestSet);  
disp(x);
```

```
if result==1  
    disp('grass');  
elseif result==2  
    disp('sand');  
elseif result==3  
    disp('asphalt');  
else  
    disp('nonclassified');  
end
```

```
e= cputime-t;  
disp(e);
```

```
end
```