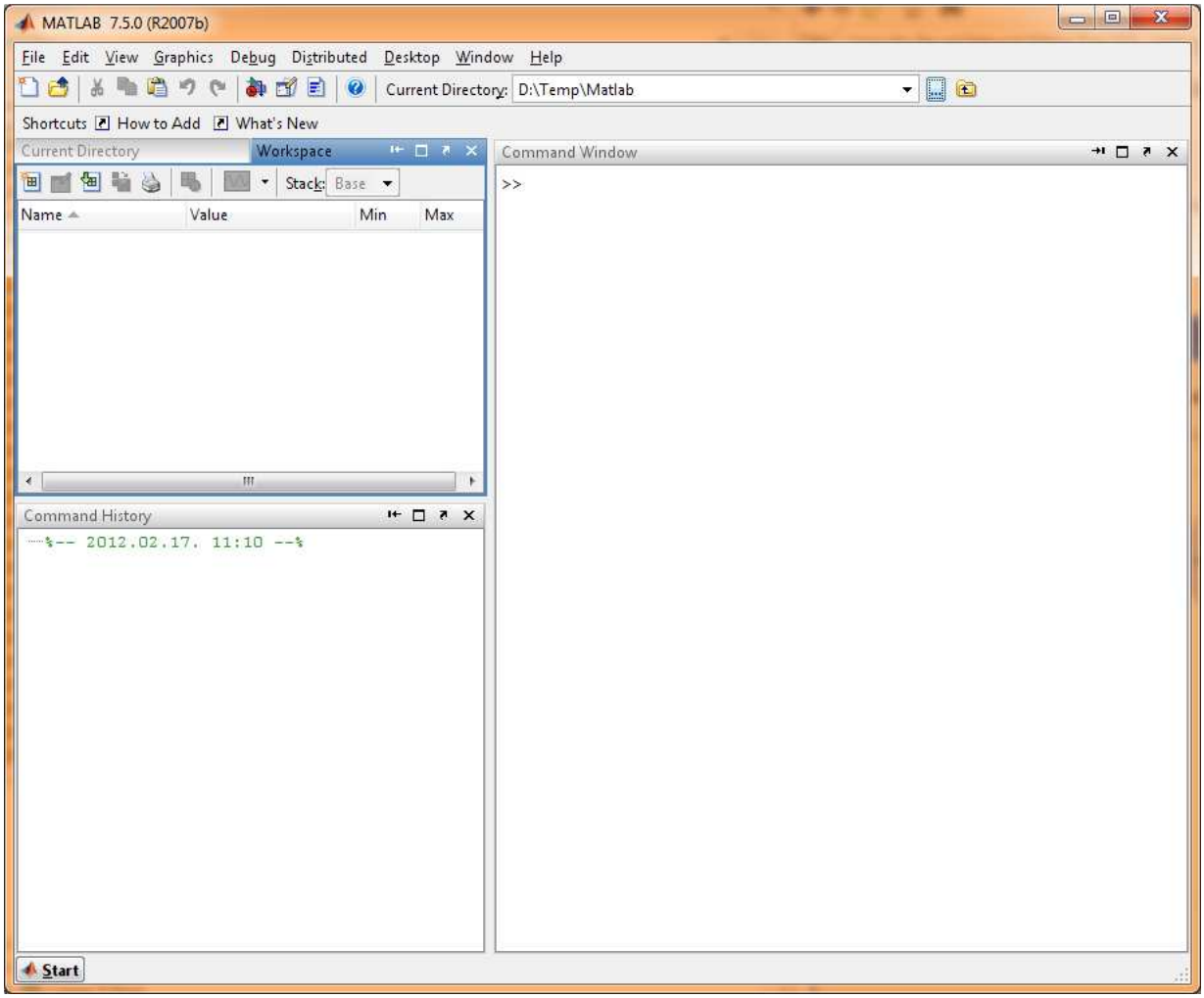
Matlab gyakorlat I.

Alapok, Neural Network toolbox

A házi feladatot a Matlab használatával kell megoldani, így szükséges, hogy mindenki rendelkezzen a minimálisan szükséges tudással, hogy később önállóan is képes legyen a feladatokat megoldani, és a további ismerkedéshez, elmélyüléshez szükséges rálátást megszerezze. Ez a célja ennek a gyakorlatnak.

1. A Matlab kezelőfelületének áttekintése



4 fő részből tevődik össze az alkalmazás:

- **Current Directory:** Ebben az ablakban láthatjuk az aktuális munka könyvtár tartalmát. Bármilyen adatot, scriptet, függvényt szeretnénk elérni a Matlab 2 különböző helyen fogja keresni. Az egyik a Path-ban lévő könyvtárak, a másik pedig az aktuális könyvtár. Ha ezek

egyikében sincs a keresett elem, akkor a Matlab nem lesz képes azt megtalálni, kivéve, ha teljes elérési útvonalat specifikálunk. Ez utóbbi csak adatot tartalmazó állományok esetében igaz.

- **Workspace:** A workspace tartalmazza a közvetlen számításokhoz használható változókat. Ezen változók nevét, típusát, méretét és egyéb általunk specifikálható tulajdonságot is képes mutatni.
- **Command History:** A korábbi parancsokat tartalmazza.
- **Command Window:** Itt jelennek meg a futtatások eredményei, de mi is adhatunk meg itt közvetlenül utasításokat, melyeket a Matlab végrehajt.

2. Hasznos alap funkciók a Matlabban

2.1. Általános eszközök

Először is tudnunk kell, hogy a Matlab interpreter nyelv, vagyis az általunk írt kód nem fordítódik le semmilyen gépi kóddá. Ezért van lehetőségünk közvetlenül a Command Window-n keresztül utasítások végrehajtására.

Mint minden programozási nyelv esetén, itt is érdemes már az elején használni a kommenteket.

```
% egy soros komment a %-jellel

%{
    Több soros komment pedig
    a %{ nyitó és %} záró
    jelekkel tehető. Mindkét
    jelnek a sor elején kell
    szerepelnie
%}
```

Ha egy függvénnyel kapcsolatban szeretnénk segítséget, akkor a Command Window **help** parancsát tudjuk használni, amennyiben ismerjük a kérdéses függvény pontos nevét. Ha nem ismerjük a függvény pontos nevét, csak a működését ismerjük és tudunk néhány specifikus kifejezést, akkor a **lookfor** paranccsal próbálhatjuk megtalálni a függvényt. Erre mutat be két példát az alábbi kódrészlet.

```
help sin
lookfor 'neural network'
```

Azért, hogy a Command Window-ban megjelenő rengeteg eredmény ne zavarjon minket össze, hasznos, ha néha letöröljük a tartalmát. Erre szolgál a

```
clc;
```

parancs. Az utasítások végén általában ; áll. Ez nem szükségszerű, ugyanis, ha nem tesszük ki a pontosvesszőt, akkor az utasítás ugyanúgy hajtódik végre, de az eredményt a Matlab kiírja a Command Window-ban. Ez persze a **clc** esetén mindegy, de nem az, ha az alábbi utasítást hajtjuk végre:

```
a = zeros(15000,1)
```

```
%vagy
a = zeros(15000,1);
```

A fenti utasítással egy csupa 0-ból álló 15000 hosszúságú oszlopvektort hozunk létre.

2.2. Vektorizálás

Ezzel eljutottunk a Matlab egyik jellegzetességéhez. A program alapvetően vektor és mátrix műveletekre optimalizált, vagyis minden esetben célszerű a problémánkat vektorok és mátrixok segítségével megfogalmazni, és az egyéb programnyelvekből ismert ciklusokat, `for` és `while`, mellőzni. Természetesen bizonyos esetekben ezek használata elkerülhetetlen, de ilyenkor is célszerű a lehető legrövidebb iteráció számra törekedni, mivel jelentős futási időbeli különbségek lehetnek. Ha az alábbi kódrészletet lefuttatjuk a futási időben akár 40x-es eltérés is lehet!

```
clc;

a = rand(150000,1);
b = rand(150000,1);

tic
s = 0;
for i = 1 : length(a)
    s = s + a(i) * b(i);
end
toc
disp(s);

tic
s = a' * b;
toc
disp(s);
```

Több dolgot is érdemes megfigyelni a fenti kódrészletben. Az első a két vektor létrehozása. A rand egy [0,1] paraméterű egyenletes eloszlású véletlen szám generátor, ahol az első argumentum a sorok, a második az oszlopok számát adja. Ha csak egy argumentumot adnánk a függvénynek, akkor négyzetes mátrixot generálna.

A tic - toc egy praktikus időmérő, alapszintű teljesítmény analízishez.

A kód bemutatja a **`for`** ciklus használatát is. A szokásostól eltérően itt nem { és } jelzik a ciklust, hanem a `for` kulcsszótól tart az **`end`** kulcsszóig. A : operátorral szekvenciákat generálhatunk. Jelen esetben 1 és 100000 között 1-es lépésközzel. Ha a lépésközt is szeretnénk változtatni, akkor azt az alábbiak szerint tehetjük meg:

```
x = -20:.1:20;
disp(x(1:10));
```

Ekkor -20 és +20 között generálunk 0.1 lépésközzel számokat. Érdemes megjegyezni, hogy a szélső értékeket a sorozat tartalmazza, vagyis a fenti utasítás eredményeként előálló tömbben a -20 és a +20 is

szerepelni fog, és ezt a tömböt a Matlab **sorvektor**ként értelmezi. Ha ezzel a módszerrel nem tudunk megfelelő számsorozatot létrehozni, akkor a **`linspace`** és a **`logspace`** függvények használata javasolt. Nézzünk egy példát a logspace használatára:

```
lx = logspace(-2, 2, 20);
disp(lx);
```

Amennyiben nem sorrendezett tartományt szeretnénk generálni, hasznos parancs a **randperm**. Generáljunk vele egy 4x4-es mátrixot egy véletlen permutációval. Ehhez szükség lesz a **reshape** parancsra, mellyel vektorból építhetünk mátrixot.

```
p = randperm(16);
disp(p);
p = reshape(p, 4, 4);
disp(p);
```

2.3. Indexelés

A következő fontos rész, amiről szót kell ejteni az indexelés. A Matlab itt is eltérő megoldást alkalmaz, ugyanis 1-től indul az index *n*-ig, ha a vektor *n* hosszú. Ezen kívül az indexekre a () segítségével lehet hivatkozni. Mátrixok esetén is, hasonló módon lehetséges az elemekhez közvetlenül hozzáférni, pl. az `m` mátrix 2. sorának 3. oszlopában lévő elem az `m(2,3)` utasítással kérdezhető le. Ezen kívül a Matlab ismeri a logikai indexelést is, vagyis nem szükséges nekünk megmondani, hogy mely sorszámú elemeken szeretnénk műveletet végezni, elég ha specifikáljuk azt a kritériumot, amely alapján szeretnénk a számokat válogatni. Például, ha a fenti példában az `a` vektor 0.5-nél nagyobb értékeit, -0.5 és 0 közé szeretnék vinni, akkor a követező utasítást használhatjuk:

```
a(a > 0.5) = a(a > 0.5) - 1;
```

A **`find`** parancs használható arra, hogy lekérjük a logikai indexelés során kiválasztásra kerülő elemek tényleges indexeit.

Még egy Matlab jellegzetességre kell odafigyelni. A vektor és mátrix műveleteket a Matlab dimenzió helyesen végzi. Vagyis egy (*n*×*m*)-es mátrix csak egy (*m*×*p*)-s mátrixszal szorozható össze, és az eredmény egy (*n*×*p*)-s mátrix lesz. Vagyis a fenti példában *a* és *b* nem szorozható össze közvetlenül, mivel mind a kettő 100000×1 dimenziójú. Ezért *a* transzponálására van szükség. Erre szolgál az aposztróf (’). Sokszor azonban szükség lehet arra, hogy tagonként végezzük el a műveletet. Hogy ekkor se kelljen ciklusokat használni, a Matlabban a . operátor használatos. Vagyis, ha a fenti példában `a.*b`-t írunk, akkor az eredmény egy 100000×1-es vektor lesz, melynek *i*-edik eleme `a(i)*b(i)` értékét tartalmazza.

2.4. Ábrázolás

A Matlabban számos beépített eszköz található adataink hatékony megjelenítésére. A legalapvetőbb eszköz a **plot**. Nézzük meg az előzőekben létrehozott logaritmikus tartományunkat.

```
plot(lx, 1, 'kx');
```

A 'k' a fekete színt jelöli, az 'x' pedig az ábrázolás stílusát adja meg. Jelenítsünk meg egy szinusz görbét:

```
y = sin(x)./x; %a y vektor elemei legyenek az x vektor elemeinek szinusza,
%elemenként elosztva (./) az x vektor elemeivel
```

```
%Megj.: egyéb esetben lehet használni a sinc függvényt is közvetlenül
plot(x,y); %grafikonon ábrázoljuk x vektor függvényében y vektort
```

Ha egy meglévő ábrára szeretnénk utólag rajzolni, pl. a hold paranccsal tudjuk megtenni:

```
hold on;
plot(x(y > 0), y(y > 0), 'r.');
hold off;
```

A fenti kódban példa látható az előzőekben megismert logikai indexelésre is: a szinuszgörbének csak a pozitív tartományát érintettük.

Ha kétdimenziós függvényekkel szeretnénk dolgozni, egy hasznos eszköz a **meshgrid**, mellyel két- vagy háromdimenziós koordinátákat generálhatunk. Nézzünk először egy kisebb tartományt:

```
[X, Y] = meshgrid(1:3, 1:5);
disp(X);
disp(Y);
```

Vegyünk most egy kicsivel bonyolultabb példát. A következő kód egy 0.5 magas, 30 sugarú hengert generál (50, 50) középponttal.

```
[X, Y] = meshgrid(1:100, 1:100);
disc = ((X - 50).^2 + (Y - 50).^2 < 30 * 30) * 0.5;
mesh(X, Y, disc);
axis([0 100 0 100 -0.5 5]);
```

A **mesh** kétdimenziós függvények színes ábrázolására praktikus, az **axis** a tengelyek tartományát állítja be.

2.5. Adatszerkezetek

A **struct** segítségével más nyelvekhez hasonlóan hierarchikus adatszerkezetek hozhatók létre. A létrehozásuk igen rugalmas, a mátrixokhoz hasonló, mint azt a következő példa is szemlélteti:

```
akarmi.egyik = 1;
disp(akarmi);
akarmi.masik = 2;
disp(akarmi);
akarmi(2).masik = 'abc';
disp(akarmi(1));
disp(akarmi(2));
```

A **cell** segítségével heterogén, vagy különböző méretű elemeket kezelhetünk együtt a mátrixokhoz hasonlóan. Szövegfüzérek kezelésekor gyakran használják a beépített függvények. Egy egyszerű példa egy cell felépítésére:

```
a = ['alma', 'korte'];
disp(a);
a = {'alma', 'korte'};
disp(a);
a{2, 1} = [1 2 3];
disp(a);
```

3. A Neural Network toolbox

3.1. Regresszió

Először egy egyszerű esetet nézünk meg. Első lépésként generáljunk mintát a tanításhoz az előző részben ismertetett módon.

```
clear all;
x = -20:0.1:20;
d = sin(x)./x;
plot(x,d);
```

Tanítsuk most a generált mintákat a hálónak. Egy dologra kell figyelni. Az NN toolbox bemeneti mintaként egy $n \times p$ méretű mátrixot vár, míg a várt kimenet $m \times p$ méretű, ahol p a tanító minták száma, vagyis az egyes oszlopokba kerül egy elem. Itt n a bemeneti vektor dimenziója, m pedig a háló által visszaadott vektor dimenziója.

```
net = newff(x,d,15); %MLP létrehozása 1 rejtett réteggel, benne 15 neuronnal.
%Minden egyéb paraméter alapértelmezett.
net = train(net,x,d); %futtassuk le a tényleges tanítási algoritmust
```

A Matlab ekkor több transzformációt elvégez, elfedve a részleteket a felhasználó elől. Először is normálja a bemeneteket, majd felbontja a mintákat 3 részre. Tanító halmazra, teszt halmazra és validációs halmazra. Az első halmazt használja a súlyok értékének meghatározására, a másodikat a túltanulás elkerülésére és a harmadikat pedig a végső eredményül kapott háló minősítésére. Mivel minden beállítást az alapértelmezett értéken hagytunk ezért a tanításhoz az ún. Levenberg-Marquardt módszert használta az MLP.

A megtanított hálót ellenőrizzük le. Először csak egy pontban, majd az összes tanítómintában, végül pedig a [-20,20] halmazon véletlenszerűen választott 100 mintában.

```
y0 = sim(net,0) %számoljuk ki a háló válaszát a 0 helyen.
y = sim(net,x); %számoljuk ki a háló válaszát az x vektor összes elemére
x_test = (40 * rand(1,100))-20; %generáljunk 100 véletlen elemet
y_test = sim(net, x_test); %számljuk ki a háló válaszát ezekben a pontokban
figure; %nyissunk meg egy új grafikus ablakot
plot(x,d,'b',x, y,'r',... %kékkel ábrázoljuk a tanítómintákat, pirossal a háló
válaszát
    x_test,y_test,'g*'); %zöld csillag jelzi a véletlen minták által felvett
értéket
```

Megj.: A … segítségével lehet egy esetlegesen hosszú utasítást több sorba tördelni.

Érdemes lehet eltérő neuron számot beállítani, és emellett is lefuttatni az eljárást, hogy lásuk mely neuron szám mellett történik meg, hogy nem képes a háló megtanulni a $\sin(x)/x$ függvényt.

Természetesen az MLP-ről tanultak ismeretében tudjuk, hogy nem csak egy rejtett réteg használatára képes a háló, hanem tetszőleges számú réteg alkalmazható, de a gyakorlati problémák többségénél csak egyet szoktak alkalmazni. Ha azonban több rejtett rétegre van szükségünk, akkor az alábbi egyszerű módosítással tudjuk ezt elérni:

```
new = newff(x,d,[6 5]); %MLP létrehozása 2 rejtett réteggel, benne rendre 6
```

```
%és 5 neuronnal. Minden egyéb paraméter alapértelmezett.
```

Nézzünk most másfajta beállítást. Nem mindig akarjuk azt, hogy az NN toolbox kiválogassa helyettünk, hogy mely minták tartozzanak a tesztekhez és melyek a tanító mintákhoz. Használjunk ehhez egy kicsit bonyolultabb függvényt.

```
d = 1.3*sin(0.4*x)+0.4*sin(0.31*x)-0.6*sin(sqrt(2)*x); %3 szinusz keveréke
P = size(x,2); %az x vektor második dimenzió mentén vett mérete, vagyis az
%oszlopok száma ami nem más mint a tanítópontok száma
perm = randperm(P); %vesszük 1-től az x hosszáig lévő egész számoknak egy
%véletlen permutációját
trainlen = round(P*0.8); %a leendő tanító mintahalmazunk hossza, az összes
%minta 80%-a
trainx = x(perm(1:trainlen)); %a perm vektor elejéről veszünk trainlen
%indexet, és vesszük az ehhez tartozó x értékeket
traind = d(perm(1:trainlen));
testx = x(perm(trainlen+1:end)); %vesszük az x vektor maradék elemeit
testd = d(perm(trainlen+1:end));
```

A most előállított tanító és teszt mintákat felhasználva az előbb is bemutatott tanítást csináljuk végig. Ebben az esetben használjuk a gradiens alapú tanítást. Ahhoz, hogy ezt specifikálni tudjuk, először meg kell adni a hálónak azt, hogy az egyes rétegek milyen nemlinearitásokat tartalmaznak. Ez egy n+1 elemű string vektor, ahol n a rejtett rétegek száma. Azért a +1, mert most a kimeneti réteghez tartozó nemlinearitást is meg kell adni. A rejtett rétegben tangens szigmoidot használjunk, míg a kimeneti rétegben egyszerű lineáris transzfer függvényt. Ezzel biztosítani tudjuk, hogy nem csak a [-1,+1] tartományban lehet a háló által visszaadott érték.

```
%MLP létrehozása, rejtett rétegben szigmoid, kimeneti rétegben lineáris
%transzfer függvény, legegyszerűbb gradiens eljárás a tanításra
net = newff(trainx,traind,35,{'tansig', 'purelin'},'traingd');
net.trainParam.lr = 0.01; %bátorsági faktor (learning rate)
net.trainParam.epochs = 2000; %hányszor iteráljunk végig az összes
tanítómintán
net.divideParam.testRatio = 0;
net.divideParam.valRatio = 0;
net.divideParam.trainRatio = 1; %minden mintát tanításra használunk
net = train(net,trainx,traind); %tényleges tanítás
plot(x,d,'b',testx,sim(net,testx),'rx'); %teszteljük a háló válaszát a
tesztpontokban
hold on;
plot(sort(testx),sim(net,sort(testx)),'r');
hold off;
```

3.2. Osztályozás

Osztályozási feladat során is hasonlóan kell eljárni a korábban látottakhoz. Ebben az esetben is van egy tanítóminta készletünk, de most az elvárt kimenet értéke diszkrét.

```
% A workspace összes változójának törlése
clear all;
% A command window törlése
clc;
% Az esetlegesen nyitva maradt fig-ek bezárása
```

```
close all;

% Osztalyozasi feladat megoldasa
load('puzzle2.mat'); %Az adatokat tartalmazó fájl neve
% Figyeljük meg, hogy a beolvasás után X és y változók
% jelennek meg a workspace-n

y = (y - 1.5) * 2;%transzformáljuk át y értékkészletét az
%{1,2}-ből a {-1,1}-be

figure(1);
subplot(1,2,1);%1 sor és 2 oszlopba kerüljenek a
%plot-ok és ez kerüljün az első helyre
plot(X(y==-1,1),X(y==-1,2),'r.',...
     X(y== 1,1),X(y== 1,2),'b.');%rajzoljuk ki a mintapontokat
```

A háló tanítását most először ismét csak az alapbeállításokkal teszteljük. Mivel a fájlból beolvasott változók soronként tartalmazzák a mintákat, ezért az NN toolboxnak a változók transzponáltját kell átadnunk.

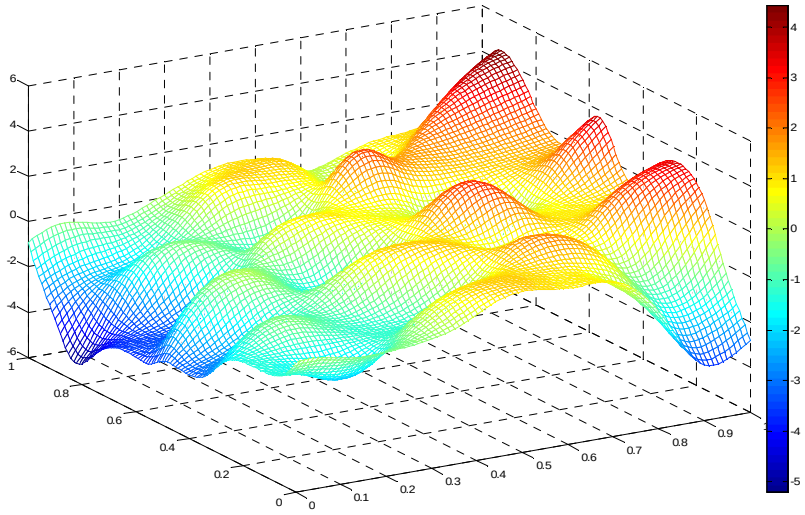
```
% Transzponálni kell X-t és y-t is
net = newff(X',y', 15);

net = train(net, X', y');

r = sim(net, X');%az eredmények lekérése
```

Tudjuk, hogy az MLP által megvalósított leképezés folytonos, vagyis nem várhatjuk, hogy az egyes minták esetében, az általunk elvárt {-1,1} értékek valamelyikét adja vissza pontosan. Érdekes lehet éppen ezért magát az MLP által megtanult felületet kirajzoltatni. Ezt végzi el az alábbi néhány sor. A kód megértésével nem feltétlenül szükséges most bajlódni.

```
figure(2);
[mx, my] = meshgrid(0:0.01:1,0:0.01:1);
test = sim(net,[mx(:)';my(:)']);
test2 = reshape(test,101,101);
mesh(mx,my,test2);
```



Egyértelmű az ábra alapján, hogy lesz olyan pont a térben, ahol se -1 se +1 nem lesz a háló kimenetének értéke és nagyjából 0 lesz. Ilyenkor két lehetőségünk van.

- Az előjelek szerint soroljuk osztályba a mintapontokat
- Csak egy bizonyos threshold-ot meghaladó helyeken végzünk osztályba sorolást, a többi helyen visszautasítjuk a döntést

Most a második esetet nézzük. Vagyis határozzuk meg az egyes osztályokba sorolt tanítómintákat, majd ábrázoljuk őket.

```
t = 0.3;%csak azokat a pontokat tekintjük, amelyekre elég
%biztos eredményt ad a háló

r = max(-1,min(1,round(0.5/t * r)));
%legyen minden pont a {-1,0,1} valamelyike
%-1 és 1 az osztályokat jelentik, míg 0 a döntés visszautasítását

figure(1);
subplot(1,2,2);
plot(X(r==-1,1),X(r==-1,2),'r.',...%rajzoljuk ki az eredményt
     X(r== 1,1),X(r== 1,2),'b.',...%piros az egyik, kék a másik
     X(r== 0,1),X(r== 0,2),'k.');%osztályt jelenti, fekete azokat
 %ahol nincs döntés
```

Kérdés miért kell max(-1,min(1,…) az alábbi kódrészletben? Ugyanezen ok miatt van a fenti ábrán a felület [-5,+5] tartományban. Hogyan lehetne ezt kivédeni? Alakítsa át úgy a kódot, hogy ez igaz legyen.