1 Neurónové siete

Pri práci s neurónovými sieťami sa budeme držať konvencie použitej v MATLABe, špeciálne v toolboxe pre neurónové siete:

Vstupné vektory sú vždy stĺpcové.

Aby sa dobre počítal potenciál neurónu, tak váhy synapsií *vstupujúcich* do neurónu sú uložené ako riadkový vektor.

1.1 Perceptrón

Perceptrón s n vstupmi, vstupnými váhami $w_1, w_2, \ldots, w_n$ a prahom ϑ budeme reprezentovať ako rozšírený váhový vektor $w = (w_1, w_2, \ldots, w_n, w_{n+1})$, kde $w_{n+1} = -\vartheta$. Pri výpočtoch so vstupným vektorom $\vec{x} = (x_1, \ldots, x_n)^T$ budeme používať rozšírený vstupný vektor $(x_1, \ldots, x_n, 1)^T$.

Naprogramujte nasledujúce funkcie:

a) `p=perc_create(n)` vytvorí perceptrón s `n` vstupmi. Funkcia vráti rozšírený (`n` + 1)-rozmerný váhový vektor, ktorého zložky sú náhodne inicializované.

b) `c=perc_recall(p,x)` vráti (riadkový) vektor `c` výstupov perceptrónu reprezentovaného rozšíreným váhovým vektorom `p` s $n + 1$ zložkami pre k vstupných n-rozmerných vektorov. Vstupy sú uložené ako stĺpce matice `x`. Na výpočet výstupu perceptrónu použite skokovú funkciu `hardlim(y)`, ktorá pre `y` ≤ 0 vracia 0 a pre `y` > 0 vracia 1.

c) `pn=perc_update(p,x,c,l)` vráti perceptrón `pn`, ktorý vznikne úpravou perceptrónu `p` vzhľadom ku k trénovacím vzorom reprezentovaným maticou `x` s k stĺpcami. Požadované výstupy `c` sú reprezentované ako riadkový vektor s k zložkami, ktoré majú hodnoty z množiny $\{0, 1\}$. Posledný parameter `l` je učiaca konštanta. Táto funkcia implementuje jednu epochu učenia perceptrónu `p` na množine vstupov `x`.

d) `e=perc_err(p,x,c)` vráti chybu perceptrónu `p` na k vstupných vzoroch uložených ako stĺpce matice `x`. Požadované výstupy `c` sú reprezentované ako riadkový vektor s k zložkami, ktoré majú hodnoty z množiny $\{0, 1\}$.

e) `pn=perc_learn(p,x,c,l,iter)` vráti perceptrón `pn`, ktorý vznikne po `iter` epochách učenia perceptrónu `p`. V jednej epoche sú perceptrónu postupne predkladané trénovacie vzory uložené ako stĺpce matice `x`, `c` je riadkový vektor požadovaných výstupov pre vzory `x`, `l` je učiaca konštanta.

Pozor funkcia `hardlim` z Matlabu sa nezhoduje so skokovou funkciou z prednášky!

Úloha: Máme dané dva pozitívne vzory $[1, 1]$,$[1, 3]$ a dva negatívne vzory $[2, 2]$, $[3, 1]$. Perceptrón má počiatočné váhy 1,1 a prah 0 a bude učený s učiacou konštantou 0.2 .

1. Aké výstupy dáva tento perceptrón na trénovacích vzoroch?
2. Ukážte ako sa mení výstup perceptrónu v prvých dvoch epochách učenia na daných trénovacích vzoroch.

3. Koľko epoch je potrebných na to, aby sa daný perceptrón naučil správne separovať pozitívne a negatívne vzory? Aké bude výsledný rozšírený váhový vektor naučeného perceptrónu.

1.2 Algoritmus zpätného šírenia – Back-Propagation

Je daná vrstevnatá neurónová sieť s topológiou [2,2,1], tj. sieť má 2 vstupné neuróny, 2 skryté neuróny a jeden výstupný neurón. Synaptické váhy medzi vstupnou a skrytou vrstvou neurónov sú dané maticou:

```
w_i_h =

   1.40000   0.40000
  -2.00000   0.80000
```

`w_i_h(i,j)` je váha spoja zo vstupu `j` do skrytého neurónu `i`. Tj. riadok váhovej matice zodpovedá vstupným váham jedného neurónu!

Synaptické váhy medzi skrytou a výstupnou vrstvou neurónov sú dané maticou:

```
w_h_o =

   2.30000 -1.00000
```

`w_h_o(1,i)` je váha spoja zo skrytého neurónu `i` do výstupného neurónu. Prahy skrytých neurónov sú v matici:

```
b_h =

   0.00000
  -0.50000
```

a prah výstupného neurónu je:

```
b_o = 0.40000
```

Teda váhy zo vstupnej vrstvy do skrytej vrstvy s pridaným jednotkovým neurónom pre prah sú:

```
>> w_i_hb=[w_i_h b_h]
w_i_hb =

   1.40000   0.40000   0.00000
  -2.00000   0.80000  -0.50000
```

Váhy zo skrytej vrstvy do výstupnej vrstvy s pridaným jednotkovým neurónom pre prah sú:

```
>> w_h_ob=[w_h_o  b_o]
w_h_ob =

   2.30000  -1.00000   0.40000
```

Úlohy:

1. Spočítajte výstup siete pre vzory `p1=[-1; 1]` a `p2=[1; -1]`. Pri výpočte v MATLABe môžete využiť funkciu `logsig(x)`, ktorá počíta (logickú) sigmoidu

$$\texttt{logsig}(x) = \frac{1}{1+e^{-x}} \quad .$$

   Jedná sa teda o sigmoidu so strmosťou 1.

2. Vzor `p1=[-1; 1]` je trénovací vzor, pre ktorý je požadovaný výstup 0.9. Aká je chyba siete pre tento vzor? Ako sa zmenia váhy siete pri vykonaní jedného kroku algoritmu spätného šírenia s parametrom učenia $\alpha = 0.2$?

1.3 Učenie vrstevnatej siete algoritmom spätného šírenia v MATLABe

Pozrite si stránky s nápovedou cez Help → Neural Network Toolbox → Backpropagation. Použitie vrstevnatých neurónových sietí sa skladá zo 4 krokov

1. Zostavenie trénovacích dát – funkcie pre predspracovanie (normalizácia dát), ... Vstupy i požadované výstupy sú stĺpcové vektory:

```
p = [-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
      -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 1 1 1 1 1 1 1 1 1  1]
t = [10   9  8  7  6  5  4  3  2  1 0 1 2 3 4 5 6 7 8 9 10];
```

2. Vytvorenie siete: príkaz `newff` má nasledujúce parametre:

   (a) Matica trénovacích vzorov.

   (b) Matica požadovaných výstupov.

   (c) riadkový vektor $[i_1, i_2, \ldots, i_m]$ popisujúci topológiu siete – i_j je počet neurónov v j-tej vrstve. Vstupná vrstva je 0-tá a jej veľkosť sa určí z prvého parametra. i_m je počet neurónov v poslednej skrytej vrstve. Veľkosť výstupnej vrstvy sa určí z druhého parametra.

   (d) Zoznam mien prenosových funkcií použitých v jednotlivých vrstvách (skrytých a výstupnej).

   (e) Meno trénovacieho algoritmu.

```
net=newff(p,t,[5],{'logsig','purelin'},'trainlm');
```

   Funkcia `newff` má v skutočnosti ešte viacej parametrov – viz nápoveda k MATLABu. Tiež je možné vynechať implicitné hodnoty parametrov – napríklad volanie `newff(p,t,5)` je ekvivalentné volaniu `newff(p,t,5,{'tansig','purelin'},'trainlm`

   Potom je možné nastaviť parametre siete, alebo nechať implicitné nastavenie:

```
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
```

Pri vytvorení je sieť automaticky inicializovaná. Už hotovú sieť je možné znova inicializovať volaním:

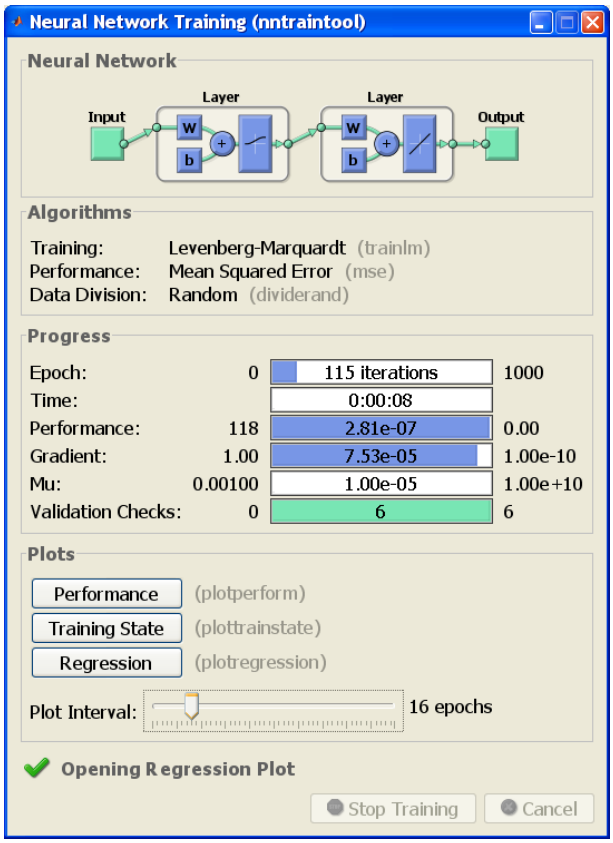
```
net=init(net)
```

Toto sa používa pri opakovanom učení siete.

3. Trénovanie siete.

```
[net,tr]=train(net,p,t);
```

Pred učením siete sa množina vzorov rozdelí na trénovaciu množinu, validačnú množinu a testovaciu množinu (implicitne v pomere 60%:20%:20%). Potom sa spustí učenie siete. Automaticky sa otvorí okno, kde sa zobrazuje priebeh učenia:



Učenie končí pri splnení niektorej z nasledujúcich podmienok:

- Bol dosiahnutý maximálny počet epoch – implicitne 1000. Dá sa nastaviť pomocou

```
net.trainParam.epochs = 300
```

- Bola dosiahnutá chyba menšia než `net.trainParam.goal` – implicitne 0.
- Gradient klesol pod `net.trainParam.min_grad` – implicitne 1^{-10}.
- Chyba na validačnej množine vzrástla v `net.trainParam.max_fail` po sebe idúcich epochách (implicitne 6 krát).
- Čas učenia prekročil nastavenú dobu `net.TrainParam.time` – implicitne táto doba nie je obmedzená a tento parameter má hodnotu `Inf`.
- Niektoré algoritmy (napr. Levenberg-Marquardtov) majú ďalšie parametre, ktorých hraničná hodnota sa dá sledovať a použiť ako stop-kritérium.

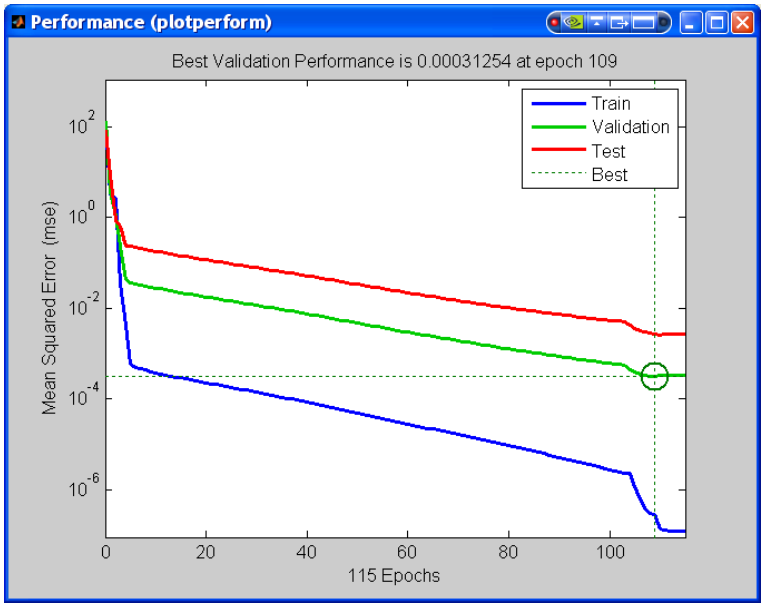
Pri vyššie uvedenom spôsobe volania funkcie `train` je v druhom parametre `tr` vrátený výsledok učenia, z ktorého sa dá prečítať mnoho informácií o použitom algoritme, priebehu učenia a dosiahnutých výsledkoch:

```
>> tr
tr =
        trainFcn: 'trainlm'
      trainParam: [1x1 struct]
      performFcn: 'mse'
    performParam: [1x1 struct]
       divideFcn: 'dividerand'
     divideParam: [1x1 struct]
        trainInd: [3 4 6 8 11 13 14 16 17 18 19 20 21]
          valInd: [2 9 12 15]
         testInd: [1 5 7 10]
            stop: 'Validation stop.'
      num_epochs: 115
      best_epoch: 109
            goal: 0
          states: {1x8 cell}
           epoch: [1x116 double]
            time: [1x116 double]
            perf: [1x116 double]
           vperf: [1x116 double]
           tperf: [1x116 double]
              mu: [1x116 double]
        gradient: [1x116 double]
        val_fail: [1x116 double]
```

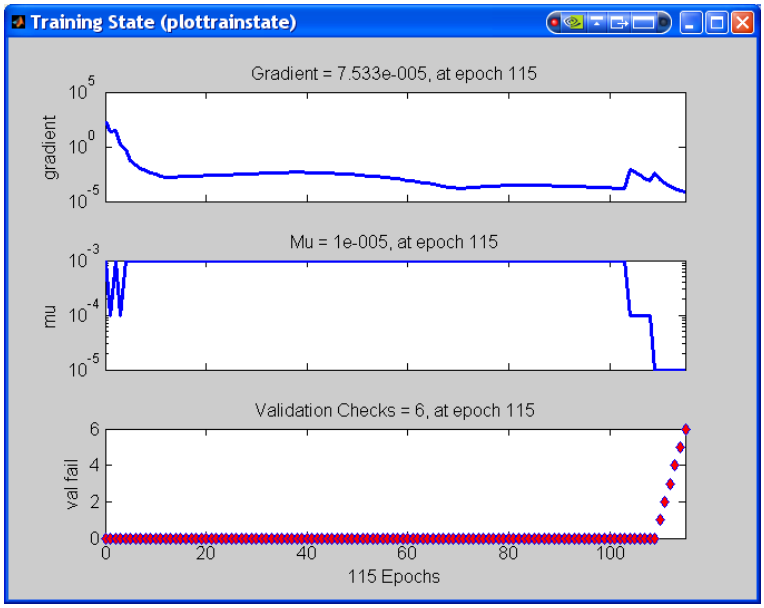
Avšak tento druhý parameter sa pri volaní nemusí uvádzať a je možné používať iba volanie

```
net1=train(net,p,t);
```

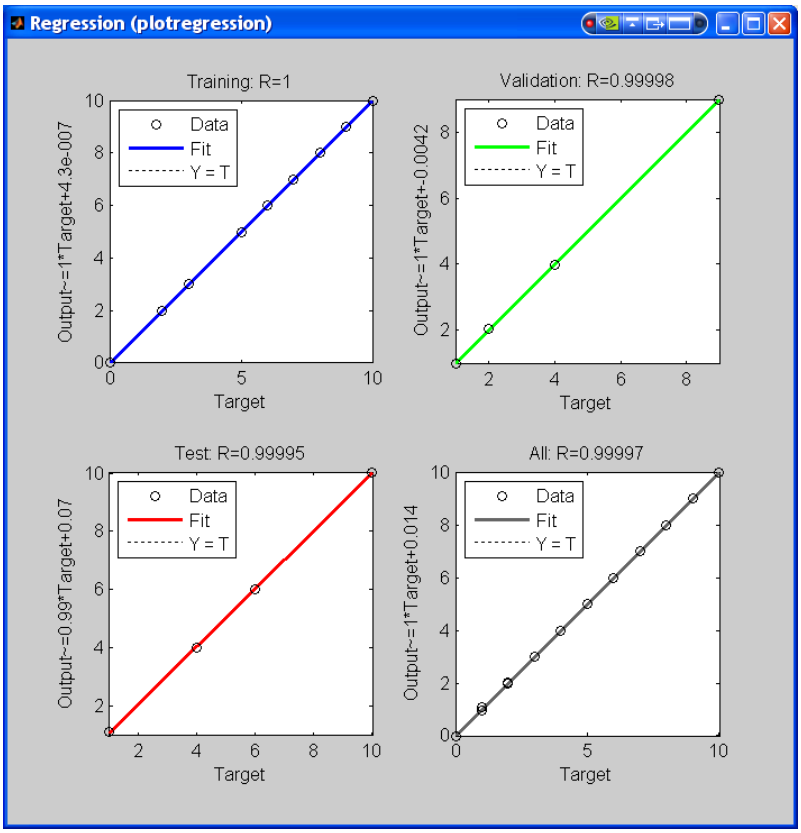
V priebehu alebo po skončení učenia je možné sledovať vývoj chyby siete (tlačítko "Performance"):



a tiež stav učenia (tlačítko "Training State"):



Grafy regresnej analýzy naučenej siete dostaneme stlačením tlačítka "Regression":



4. Vybavovanie siete – simulácia siete na nových dátach. Funkcia `sim(net1,p)`. Napríklad

```
>> p = [1;2];
>> a = sim(net1,p)
a =
    0.1735
```

2 Transformácie dát

Normalizácia hodnôt vektorov môže výrazne urýchliť učenie siete. V balíku na neurónové siete sú pre takého transformáie pripravené funkcie. Naviac mnoho modelov neurónových sietí v Toolboxe robí automatické predspracovanie vstupov. Napr. `newff` najprv nahradí chýbajúce hodnoty vstupov (`NaN`), vynechajú sa konštantné riadky a urobí sa min-max normalizácia na interval $< -1, 1 >$. Je to v parametroch siete

```
net.inputs{1}.processFcns =
      {'fixunknowns','removeconstantrows','mapminmax'}
net.outputs{2}.processFcns =
      {'removeconstantrows','mapminmax'})
```

2.1 Min-max škálovanie na interval $< -1, 1 >$

```
[pn,ps] = mapminmax(p);
[tn,ts] = mapminmax(t);
net = train(net,pn,tn);
```

`pn`, resp. `tn`, sú transformované vstupné vzory, resp. požadované výstupy. `ps`, resp. `ts`, sú parametre príslušných transformácií, ktoré sa využijú pri spätnej transformácii. Keď takto naučenú sieť aplikujeme, tak jej výstup musíme patrične transformovať

```
an = sim(net,pn);
a = mapminmax('reverse',an,ts);
```

Ak chceme použiť tú istú transformáciu na nové dáta

```
pnewn = mapminmax('apply',pnew,ps);
anewn = sim(net,pnewn);
anew = mapminmax('reverse',anewn,ts);
```

2.2 Škálovanie podľa priemeru a štandardnej odchýlky

Funkcie

```
[pn,ps] = mapstd(p);
[tn,ts] = mapstd(t);
```

```
an = sim(net,pn);
a = mapstd('reverse',an,ts);

pnewn = mapstd('apply',pnew,ps);
anewn = sim(net,pnewn);
anew = mapstd('reverse',anewn,ts);
```

2.3 Principal Component Analysis

Najprv je treba dáta normalizovať tak, aby mali strednú hodnotu 0 a rozptyl 1. Potom môžeme urobiť PCA analýzu.

```
[pn,ps1] = mapstd(p);
[ptrans,ps2] = processpca(pn,0.02);
```

Pri vyššie uvedenej transformácii sa vynechajú zložky, ktoré prispievajú menej než 2% k celkovému rozptylu. Ak chceme túto transformáciu použiť na nových údajoch

```
pnewn = mapstd('apply',pnew,ps1);
pnewtrans = processpca('apply',pnewn,ps2);
a = sim(net,pnewtrans);
```

Úloha: Súbor `dataset1.mat` obsahuje dve matice. Matica `p` sú vzory a matica (riadkový vektor) `t` sú požadované výstupy. (Do MATLABu sa načítajú príkazom `load dataset1`.)

1. Vyberte z tejto trénovacej množiny polovicu vzorov a použite ich ako trénovaciu množinu pre naučenie vrstevnatej neurónovej siete.

2. Pomocou PCA-predspracovania transformujte vstupné 3-rozmerné vektory na 2-rozmerné vektory a porovnajte rýchlosť učenia a presnosť naučenia príslušných sietí s rovnakými parametrami. Môžete sa inšpirovať demo príkladom `demobp1.m`. Aby ste dosiahli redukciu na 2 dimenzie je treba oproti demo-príkladu zľaviť na presnosti!

2.4 Hopfieldove siete

Váhy Hopfieldovej siete sa nastavujú podľa Hebbovského pravidla:

$$W_{ij} = \sum_{p, i \neq j} x_i^p x_j^p = \sum_p \left[(\vec{x}^p)^T \vec{x}^p - I \right], \quad \text{kde } I \text{ je jednotková matica.}$$

Predpokladáme, že zložky vektorov sú z množiny $\{-1, 1\}$.
Úloha: Spočítajte váhy Hopfieldovej siete siete, ktorá sa má naučiť vzory:

```
B=[  1 -1 -1 ;
    -1  1 -1 ;
     1  1 -1 ;
    -1 -1 -1
  ]
```

Aký vzor vybaví táto sieť pri vstupe `[1; -1; -1; -1]`:

(i) pri synchrónnej adaptácii;

(ii) pri asynchrónnej adaptácii?

Spočítajte energetickú funkciu siete pri jednotlivých krokoch vybavovania.