

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 Algoritmus spätného 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

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

Jedná sa teda o sigmoidu so strmostou 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 $\alpha = 0.2$?

1.1 Učenie vrstevnatej siete algoritmom spätného šírenia v MATLABe

Pozrite si stránky s nápovedou cez Help → Neural Network Toolbox → Back-propagation. 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 $[i_1, i_2, \dots, i_m]$ 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:

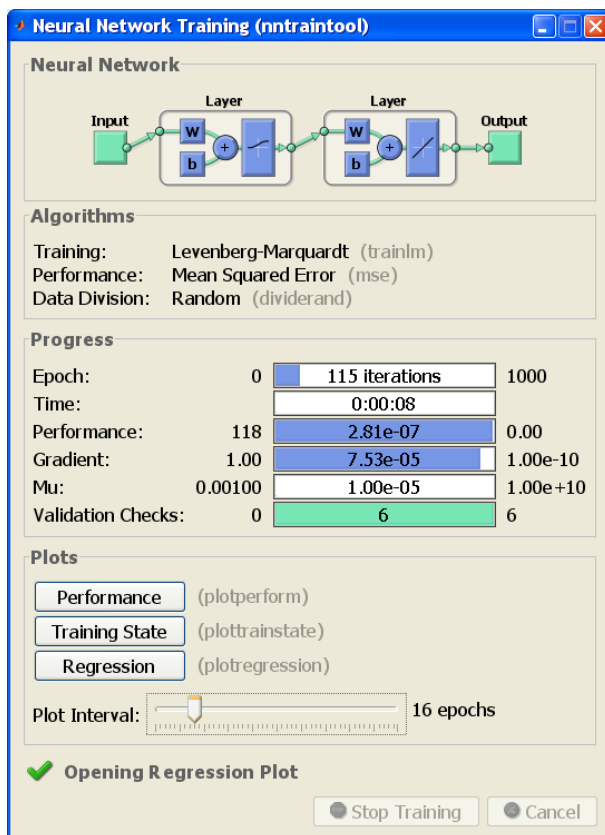
```
net=init(net)
```

Toto sa používa pri opakovanom učení siete.

3. Trénovanie siete.

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

Pred učením siete sa množina vzorov rozdelí na tréningovú 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.

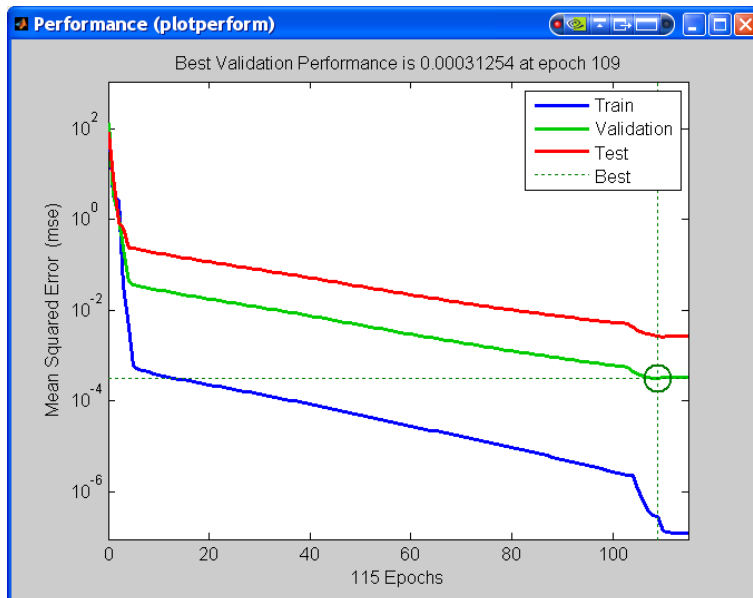
Pri vyššie uvedenom spôsobe volania funkcie `train` je v druhom parametre `tr` vrátený výsledok učenia, z ktorého sa dá prečítať mnoho informácií o použítom algoritme, priebehu učenia a dosiahnutých výsledkoch:

```
>> 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]
```

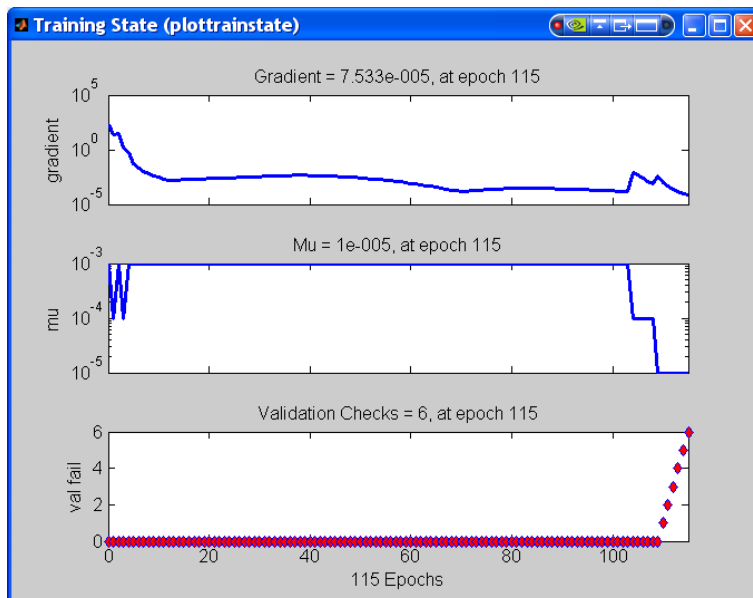
Avšak tento druhý parameter sa pri volaní nemusí uvádzať a je možné používať iba volanie

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

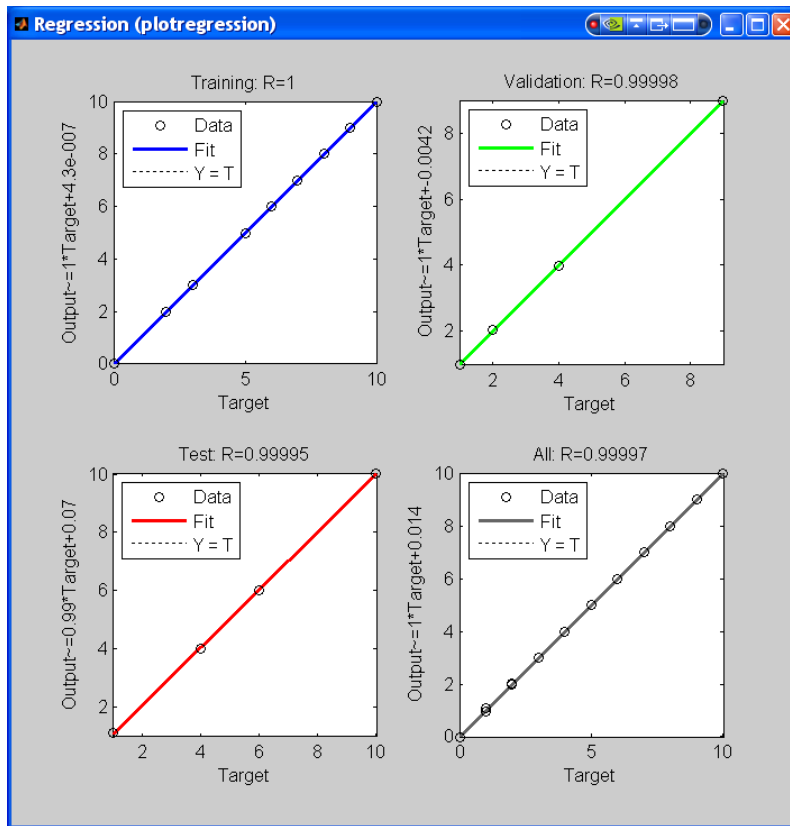
V priebehu alebo po skončení učenia je možné sledovať vývoj chyby siete (tlačítka "Performance"):



a tiež stav učenia (tlačítka "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 takéto transformácie pripravené funkcie. Navyše 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 $\langle -1, 1 \rangle$

```
[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 <http://ksvi.mff.cuni.cz/~mraz/nn/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 `nftool` naučte neurónovú sieť aproximaovať funkciu podľa vzorov **z p** a **t**.
3. Pomocou `nntool` naučte neurónovú sieť s dvomi skrytými vrstvami aproximaovať funkciu podľa vzorov **z p** a **t**. Vyskúšajte rôzne učiace algoritmy `'traingd'`, `'traingdm'`, `'trainbr'`, ...