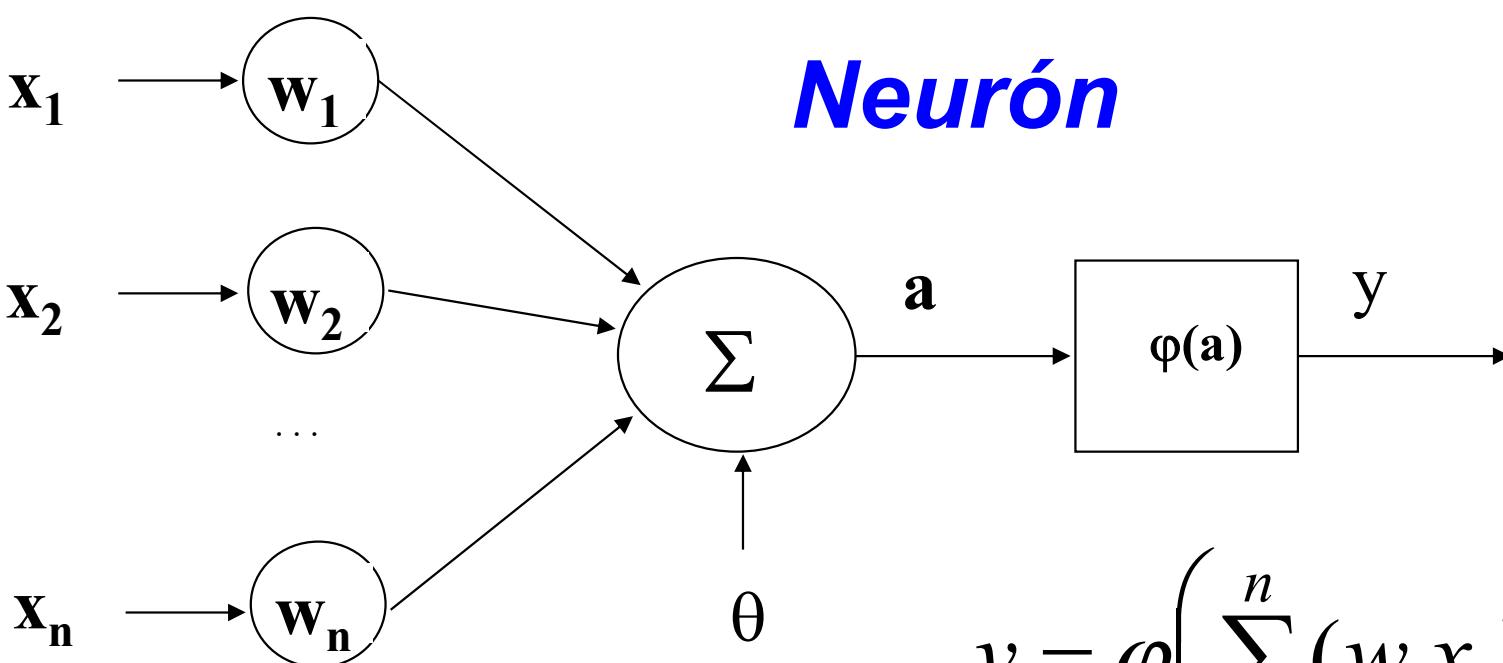
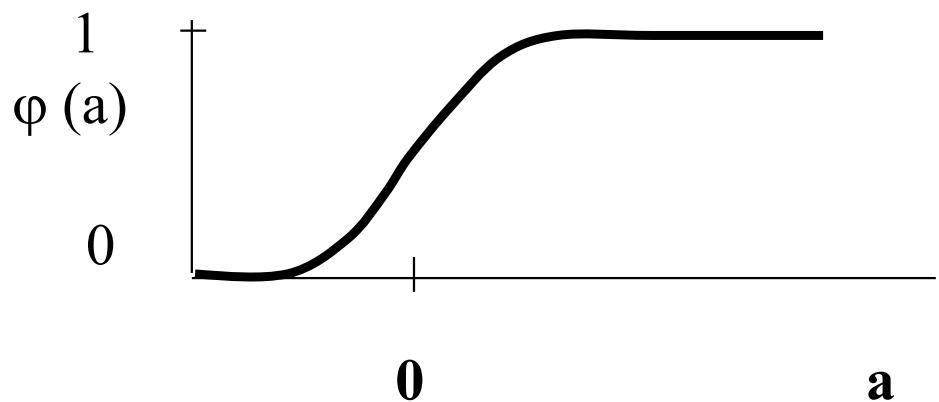


*Viacvrstvové  
perceptrónové siete  
(Multilayer Perceptron networks -  
*MLP*)*



$$y = \varphi\left(\sum_{i=1}^n (w_i x_i) - \theta\right) = \varphi(a)$$



$$\varphi(a) = \frac{1}{1 + e^{-\beta a}}$$

Sigmoida

**$x_i$  - vstupy neurónu**

**$\theta$  - prah (citlivost) neurónu**

**$\varphi$  - aktivačná funkcia neurónu**

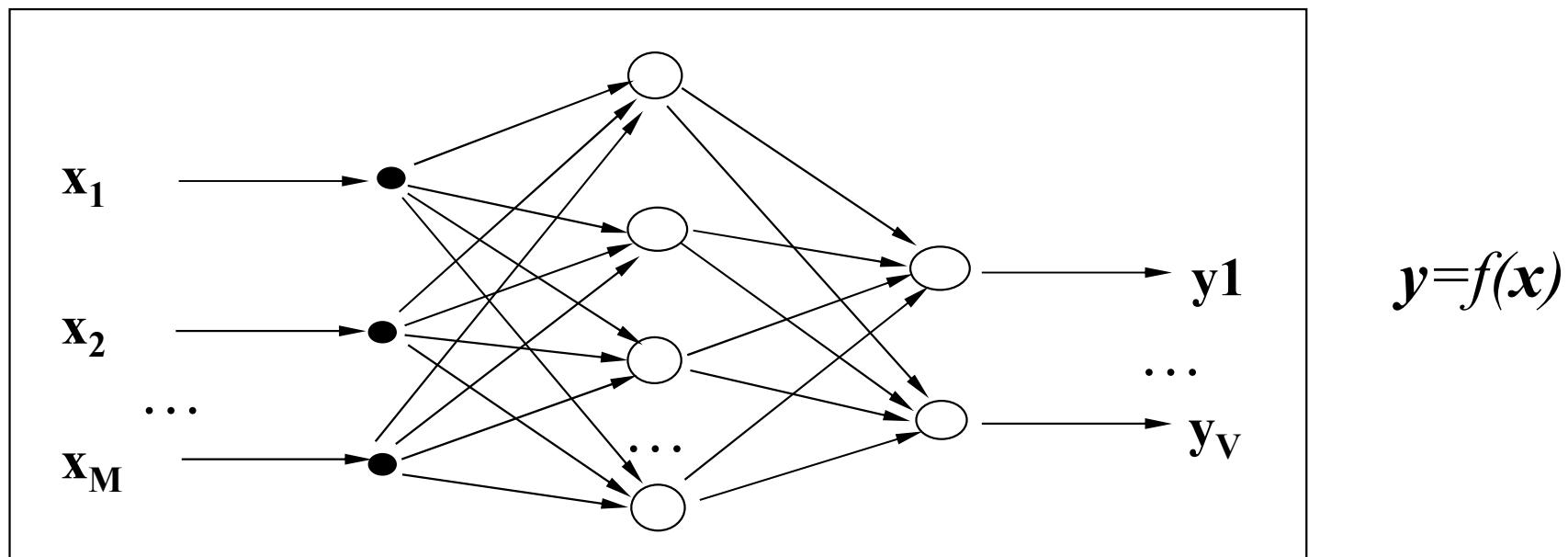
**$w_i$  - váhy synaptických spojení**

**$a$  - vnútorná aktivita neurónu**

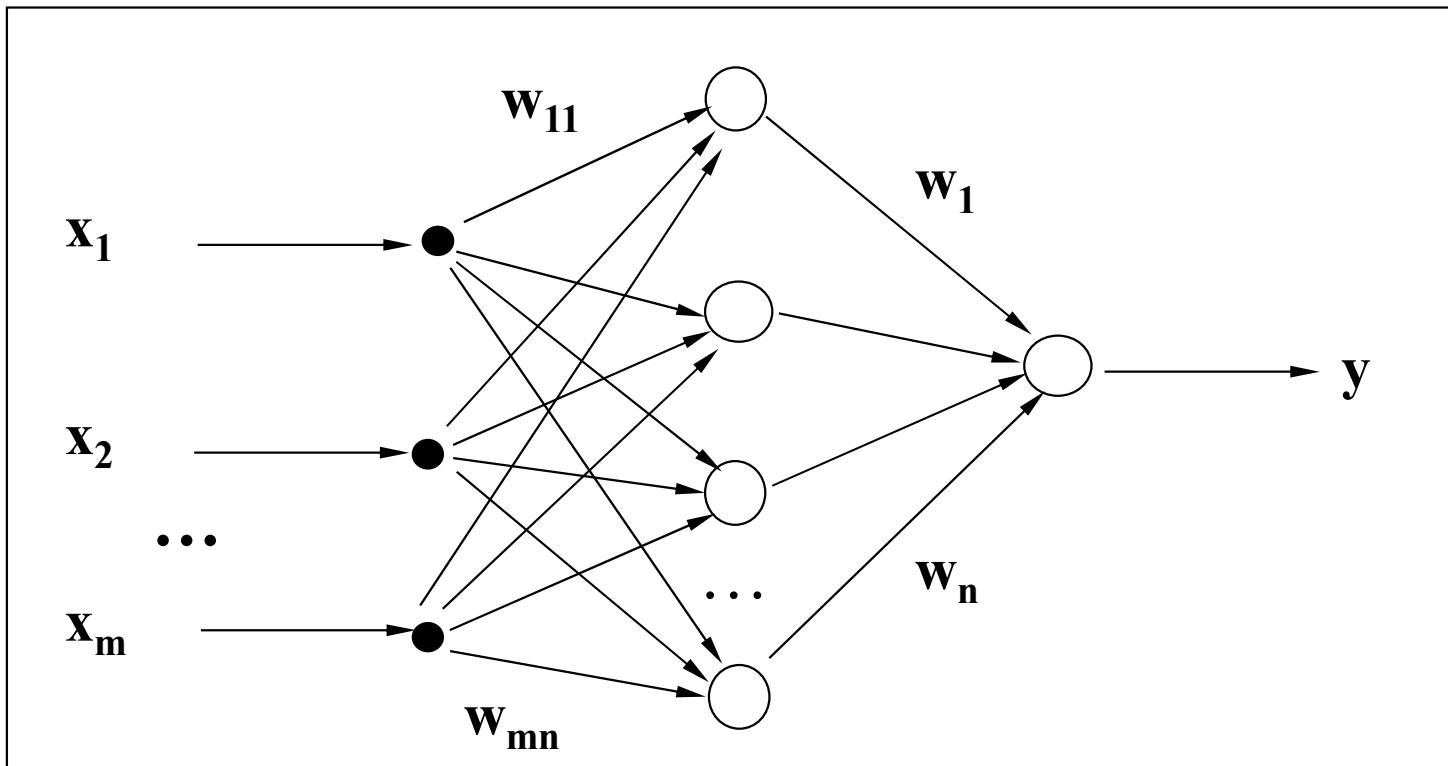
**$y$  - výstup neurónu**

# *Siet'*

- Obsahuje aspoň jednu skrytú vrstvu neurónov.
- Neuróny väčšinou obsahujú spojitú nelineárnu aktivačnú funkciu (obyčajne sigmoidu alebo hyperbolický tangens).
- 3-vrstvové siete sú schopné approximovať ľubovoľnú spojitú nelineárnu funkciu (transformáciu).
- 4-vrstvové siete sú schopné approximovať ľubovoľnú nespojité nelineárnu funkciu (transformáciu).
- Parametrizácia takýchto sietí sa realizuje trénovaním neurónovej siete algoritmom "spätného šírenia chyby,, (učenie s učiteľom).



**Trénovanie MLP :**  
**Algoritmus spätného šírenia chyby**  
**(„Back- Propagation“)**



$$w_{ij}=? ; w_j=?$$

**Globálna chyba siete  $E \rightarrow \min$**

$$E = \sum_{p=1}^N \varepsilon = \sum_{p=1}^N \frac{1}{2} \sum_{k=1}^V e_k^2$$

N - počet vzoriek trénovacej množiny, V - počet neurónov vo výstupnej vrstve

$\alpha$  – krok učenia

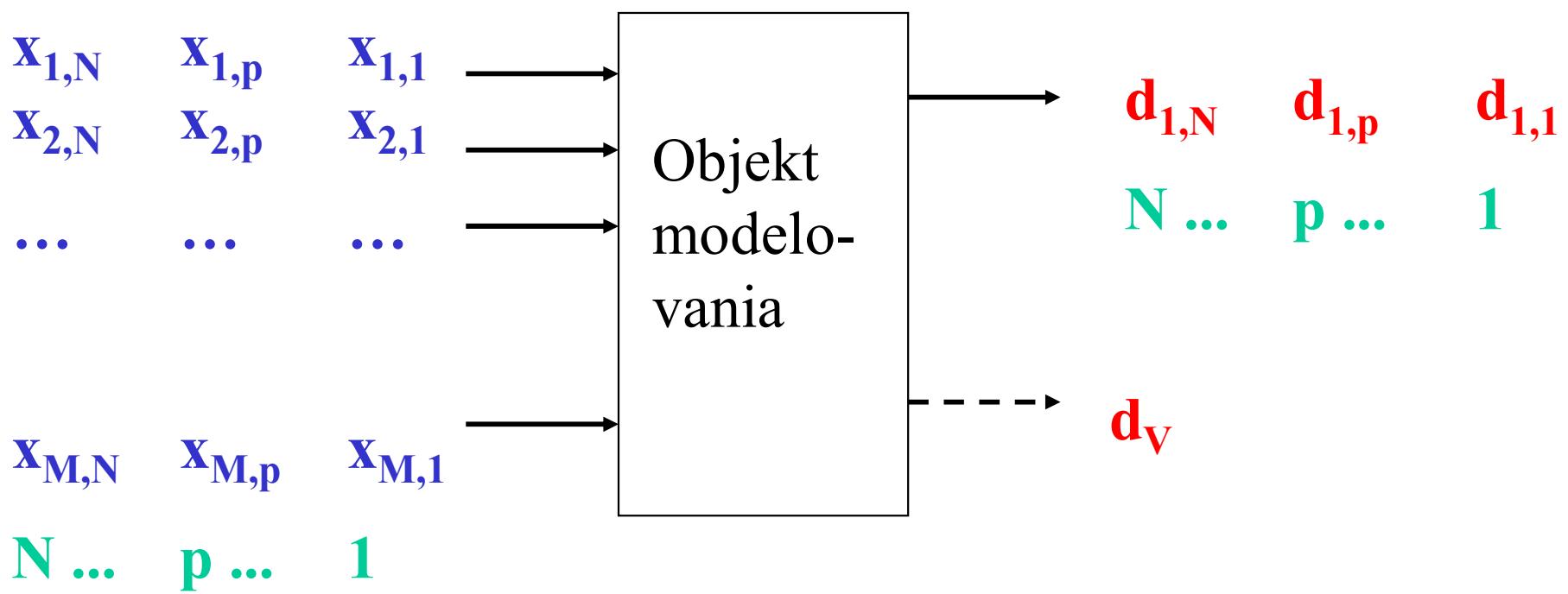
$$e_k = d_k - y_k$$

$$\Delta w = \alpha \frac{\delta E}{\delta w_{ij}}$$

Výpočet váh w sa uskutočňuje iteračným algoritmom, kde sa postupne počítajú korekcie každej váhy  $\Delta w$  ;  $w(t)=w(t-1)+\Delta w$

Predpokladajme trénovaciu množinu (vstupno/výstupných) dát  
 $D=\{x_{pq}, d_{pr}\}$ ;

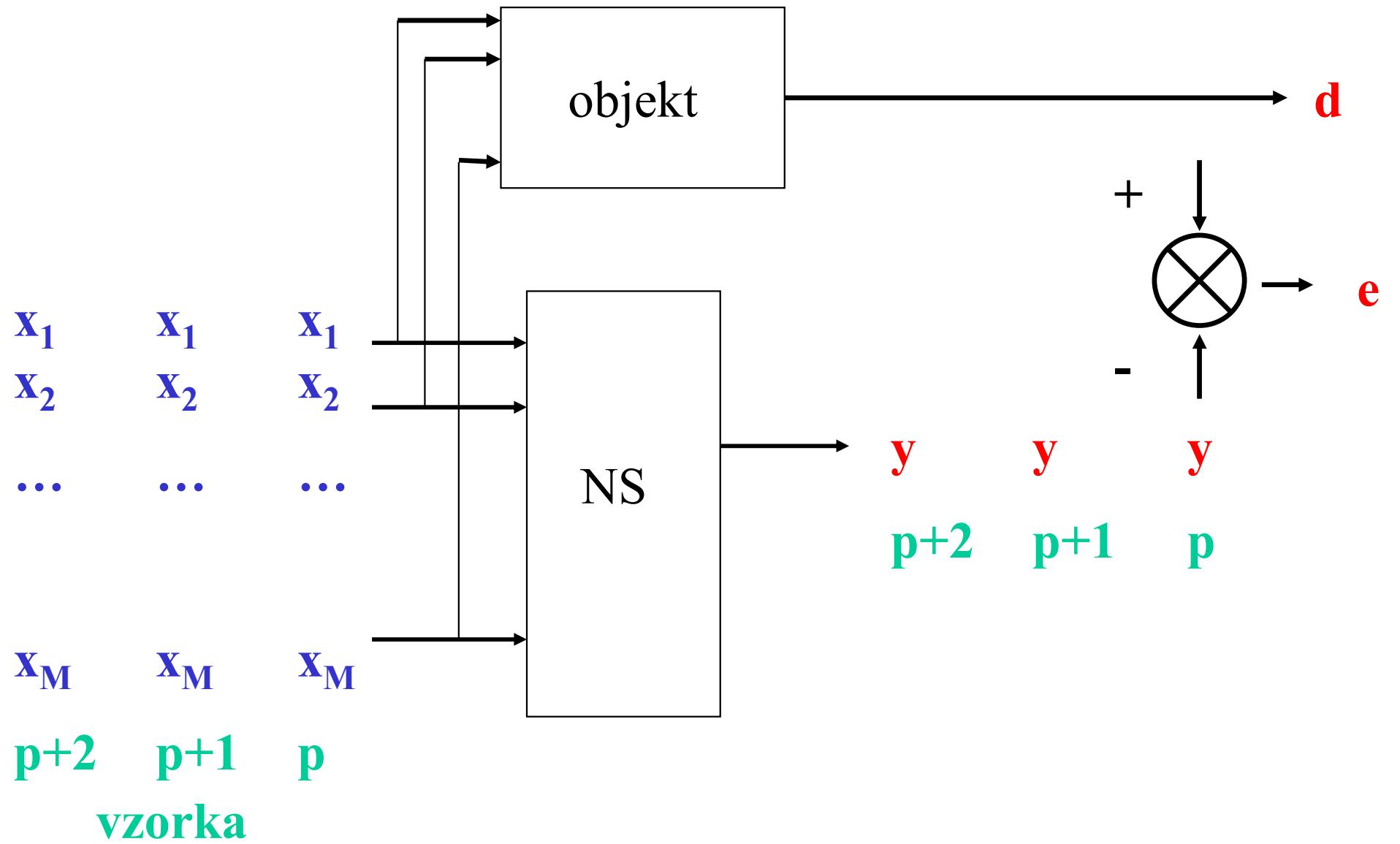
$p=1, \dots, N$  je počet vzoriek  
 $q=1, \dots, M$  je počet vstupov siete  
 $r=1, \dots, V$  je počet výstupov siete



Číslo vzorky      vstupy      (požadované) výstupy

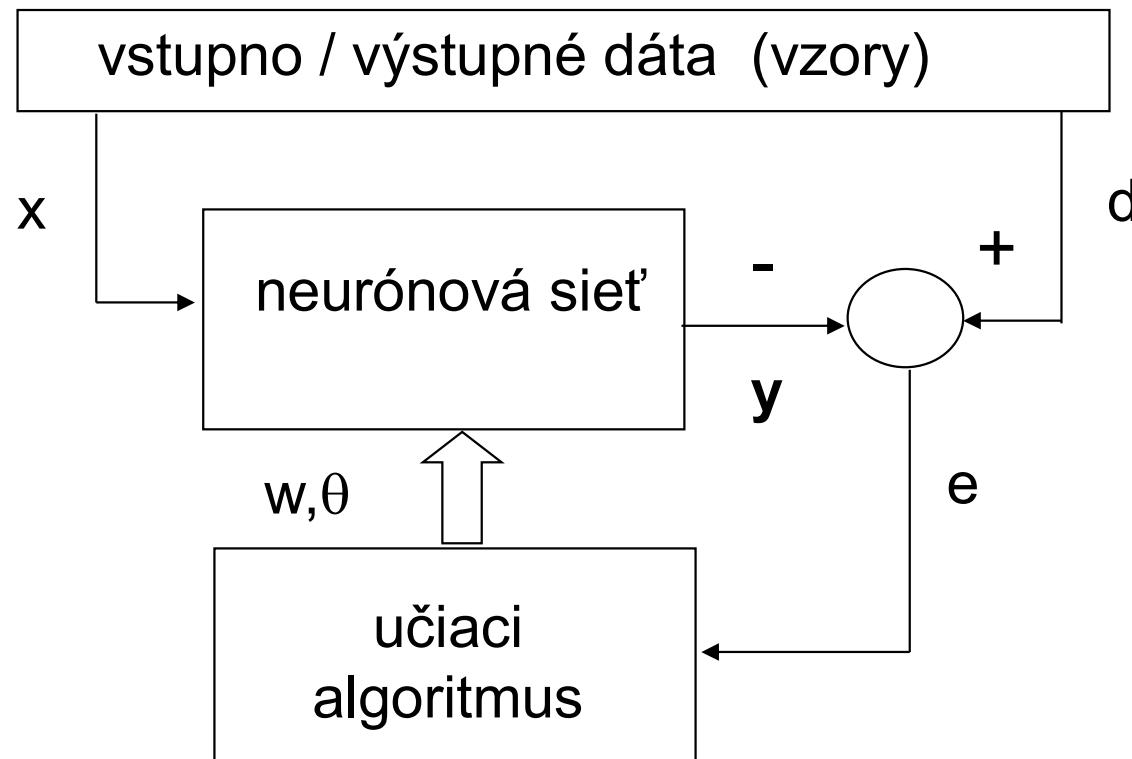
**Trénovanie (parametrizácia) neurónovej siete prebieha vo viacerých cykloch („vlnách“) - tzv. EPOCHÁCH.**

**V každej epoche sa postupne cez siet' prešíri celá postupnosť trénovacích vzoriek  $x_p$  (postupnosť vstupných vektorov), porovnáva sa so skutočnými výstupmi a na základe odchýliek sa upravujú parametre siete).**

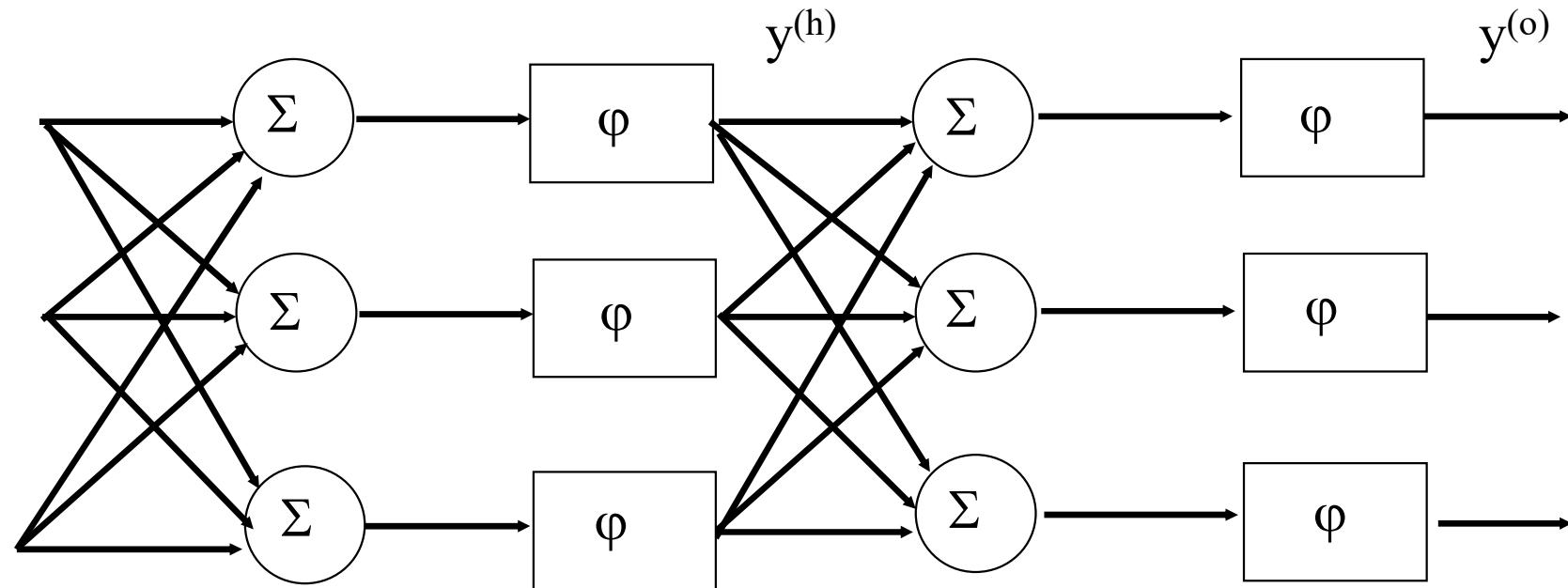


1 epocha:  $p=1,2,\dots,N$

# Algoritmus trénovania UNS



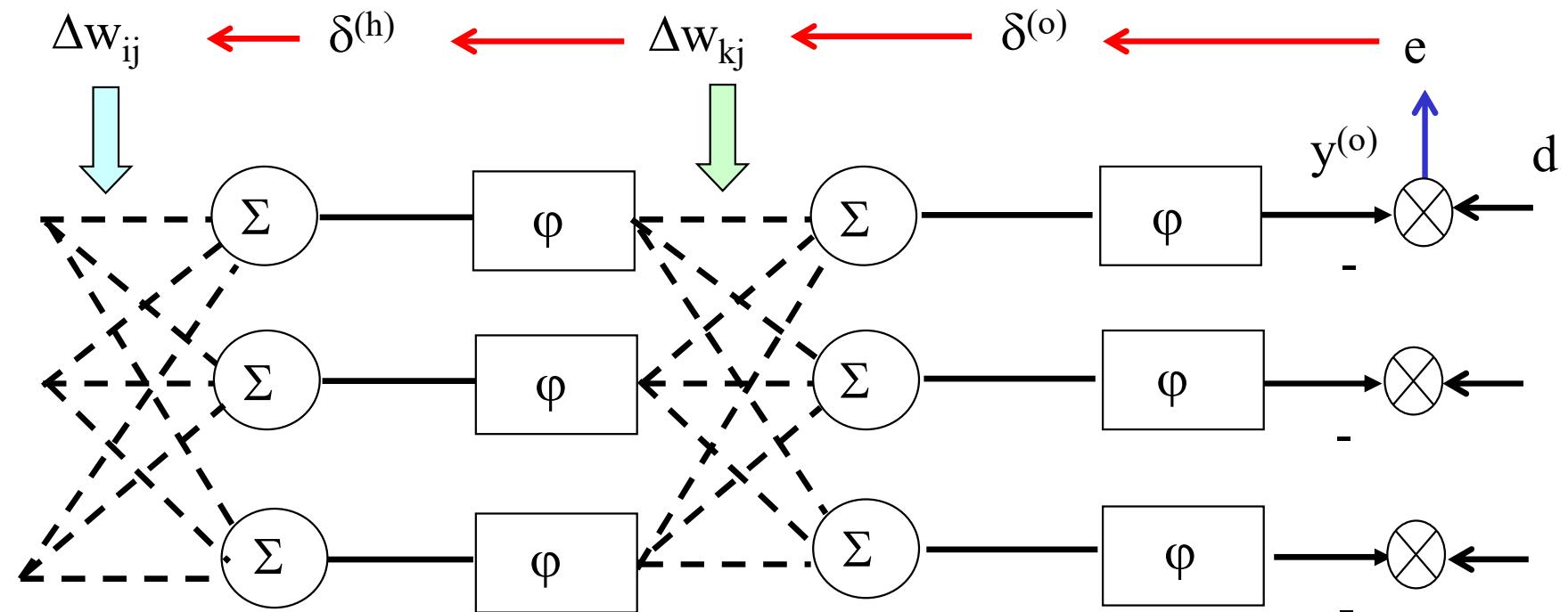
## Dopredná fáza šírenia signálov v neurónovej sieti



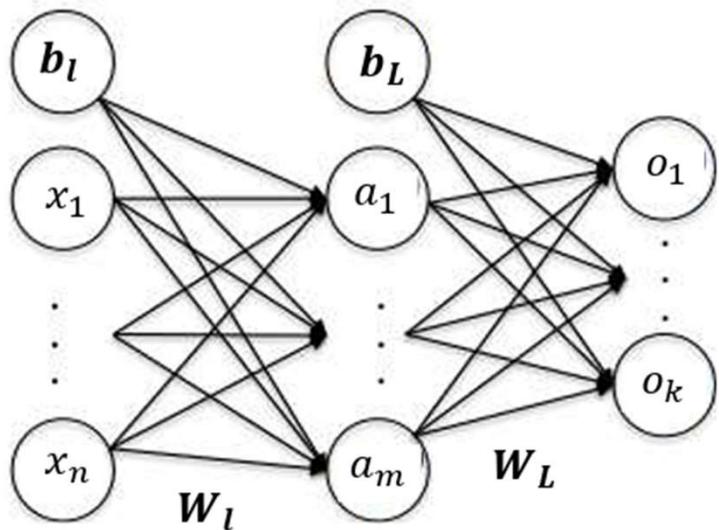
$^{(h)}$  – index neurónov skrytej vrstvy (hidden),

$^{(o)}$  – index neurónov výstupnej vrstvy (output)

## Spätná fáza šírenia signálov v neurónovej sieti



# MLP s jednou skrytou vrstvou



Matice:

$$W_l = \begin{pmatrix} w_{11}^l & \cdots & w_{m1}^l \\ \vdots & \ddots & \vdots \\ w_{1n}^l & \cdots & w_{mn}^l \end{pmatrix} \quad W_L = \begin{pmatrix} w_{21}^L & \cdots & w_{k1}^L \\ \vdots & \ddots & \vdots \\ w_{2m}^L & \cdots & w_{km}^L \end{pmatrix}$$

Vektory:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} \quad o = \begin{pmatrix} o_1 \\ o_2 \\ \vdots \\ o_k \end{pmatrix}$$

Výstup siete (dopredné šírenie):

$$a = \varphi_1 (W_l^T x + b_l)$$

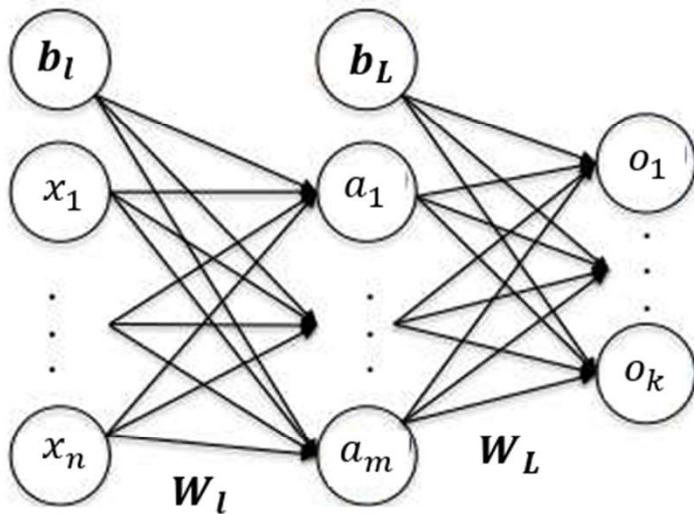
[m x n] x [n x 1] + [m x 1]

$$o = \varphi_2 (W_L^T a + b_L)$$

[k x m] x [m x 1] + [k x 1]

$$b_l = \begin{pmatrix} b_1^l \\ \vdots \\ b_m^l \end{pmatrix} \quad b_L = \begin{pmatrix} b_1^L \\ \vdots \\ b_k^L \end{pmatrix}$$

# MLP s jednou skrytou vrstvou



Vektory:

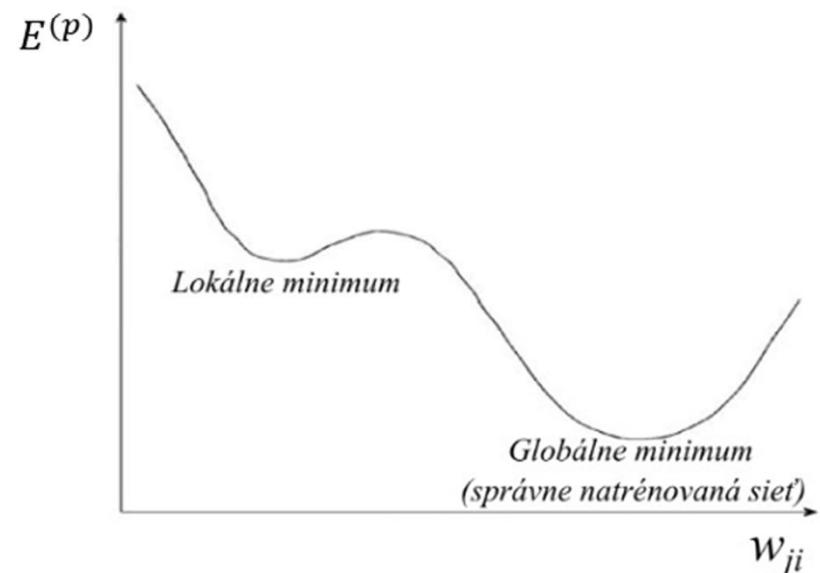
$\mathbf{o}$  – vektor výstupov neurónovej siete

$\mathbf{d}$  – cieľový vektor (ideálny výstup)

Celková chyba siete pre daný tréningový vzor  $p$ , napríklad MSE:

$$E^{(p)} = \frac{1}{2} \sum_{j=1}^k (d_j^{(p)} - o_j^{(p)})^2$$

Cieľom trénoania je minimovať celkovú chybu vzhľadom na jednotlivé parametre siete.



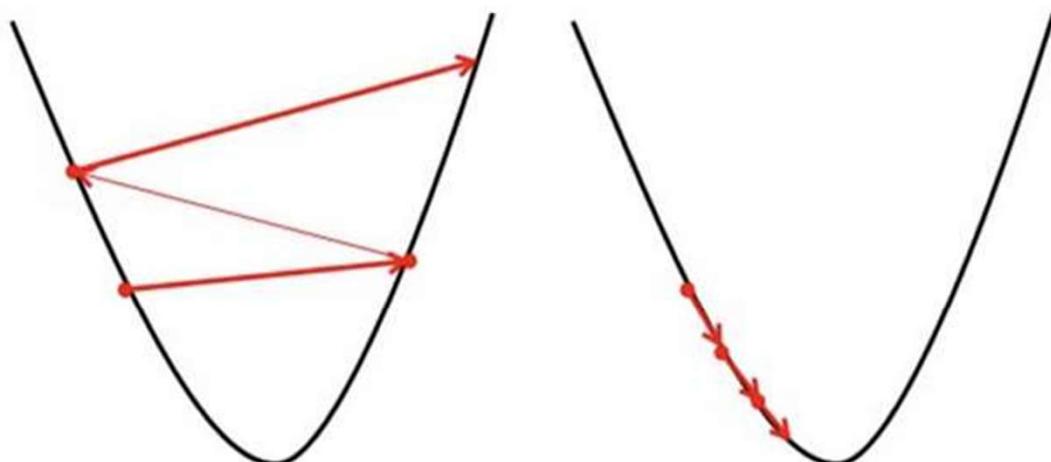
# MLP s jednou skrytou vrstvou

Základnou optimalizačnou metódou je metóda najväčšieho spádu:

$$w_{ji}^{(p)}(k+1) = w_{ji}(k) - \alpha \frac{\partial E^{(p)}}{\partial w_{ji}}(k)$$

Parametre:

$\alpha$  – krok učenia (konšanta)



Reťazové pravidlo:

$$ak z = f(y) \quad a \quad y = g(x)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

V našom prípade:

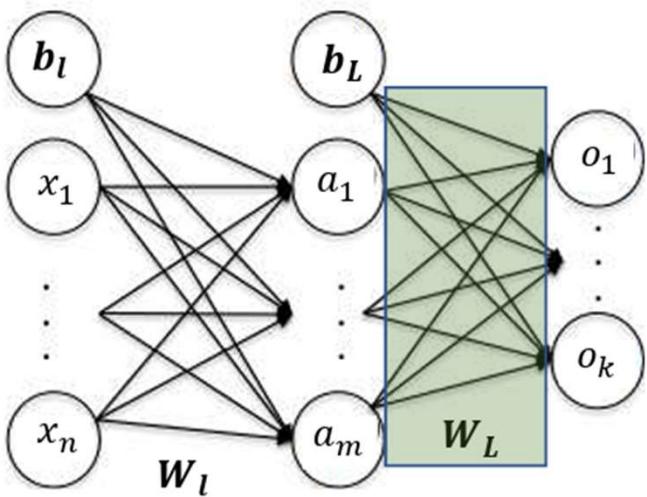
$$E^{(p)} = \frac{1}{2} \sum_{j=1}^k (d_j^{(p)} - o_j^{(p)})^2$$

$$net_j^{(p)} = \sum_{i=1}^n w_{ji} \tilde{a}_i^{(p)} + b_j = \mathbf{w}_j^T \tilde{\mathbf{a}}^{(p)} + b_j$$

Pričom:

$$\tilde{a}_i^{(p)} = \begin{cases} a_i^{(p)} & \text{ak je tento vstup zároveň výstupom z} \\ & \text{predchádzajúceho neurónu} \\ x_i^{(p)} & \text{ak je tento vstup priamym} \\ & \text{vstupom do siete} \end{cases}$$

# MLP s jednou skrytou vrstvou



Opäť uplatníme reťazové pravidlo:

$$\frac{\partial E^{(p)}}{\partial net_j^{L(p)}} = \frac{\partial E^{(p)}}{\partial o_j^{(p)}} \frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} \text{ pretože } E^{(p)} = f(o_j^{(p)}) \text{ a } o_j^{(p)} = g(net_j^{L(p)})$$

$$\begin{aligned} \frac{\partial E^{(p)}}{\partial o_j^{(p)}} &= \frac{\partial}{\partial o_j^{(p)}} \left[ \frac{1}{2} \sum_{j=1}^k (d_j^{(p)} - o_j^{(p)})^2 \right] \\ &= -(d_j^{(p)} - o_j^{(p)}) = -e_j^{(p)} \end{aligned}$$

$$\frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} = \varphi'(net_j^{L(p)}) \quad \text{Podmienka diferencovateľnosti aktivačnej funkcie}$$

$$\frac{\partial net_j^{L(p)}}{\partial w_{ji}^L} = \frac{\partial}{\partial w_{ji}^L} \left[ \sum_{i=1}^m w_{ji}^L a_i^{(p)} + b_j^L \right] = a_i^{(p)}$$

$$w_{ji}^{L(p)}(k+1) = w_{ji}^L(k) + \alpha e_j^{(p)} \varphi'(net_j^{L(p)}) a_i^{(p)}$$

**Úprava váh siete poslednej vrstvy :**

$$w_{ji}^{L(p)}(k+1) = w_{ji}^L(k) - \alpha \frac{\partial E^{(p)}}{\partial w_{ji}^L}(k)$$

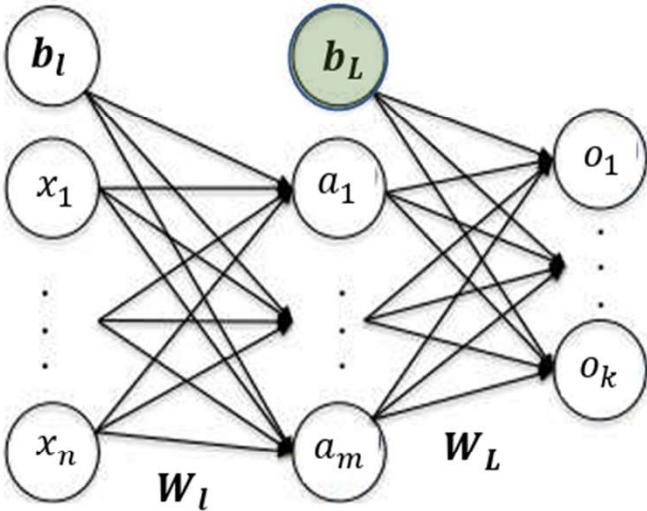
Uplatníme reťazové pravidlo:

$$\frac{\partial E^{(p)}}{\partial w_{ji}^L} = \frac{\partial E^{(p)}}{\partial net_j^{L(p)}} \frac{\partial net_j^{L(p)}}{\partial w_{ji}^L} \quad \text{pretože } E^{(p)} = f(net_j^{L(p)}) \text{ a } net_j^{L(p)} = g(w_{ji}^L)$$

Lokálny gradient j-teho výstupného neurónu

$$\delta_j^{(p)} = e_j^{(p)} \varphi'(net_j^{L(p)})$$

# MLP s jednou skrytou vrstvou



Opäť uplatníme reťazové pravidlo:

$$\frac{\partial E^{(p)}}{\partial net_j^{L(p)}} = \frac{\partial E^{(p)}}{\partial o_j^{(p)}} \frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} \text{ pretože } E^{(p)} = f(o_j^{(p)}) \text{ a } o_j^{(p)} = g(net_j^{L(p)})$$

$$\begin{aligned} \frac{\partial E^{(p)}}{\partial o_j^{(p)}} &= \frac{\partial}{\partial o_j^{(p)}} \left[ \frac{1}{2} \sum_{j=1}^k (d_j^{(p)} - o_j^{(p)})^2 \right] \\ &= -(d_j^{(p)} - o_j^{(p)}) = -e_j^{(p)} \end{aligned}$$

$$\frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} = \varphi'(net_j^{L(p)}) \quad \text{Podmienka diferencovateľnosti aktivačnej funkcie}$$

$$\frac{\partial net_j^{L(p)}}{\partial b_j^L} = \frac{\partial}{\partial b_j^L} \left[ \sum_{i=1}^m w_{ji}^L a_i^{(p)} + b_j^L \right] = 1$$

$$b_j^{L(p)}(k+1) = b_j^L(k) + \alpha e_j^{(p)} \varphi'(net_j^{L(p)})$$

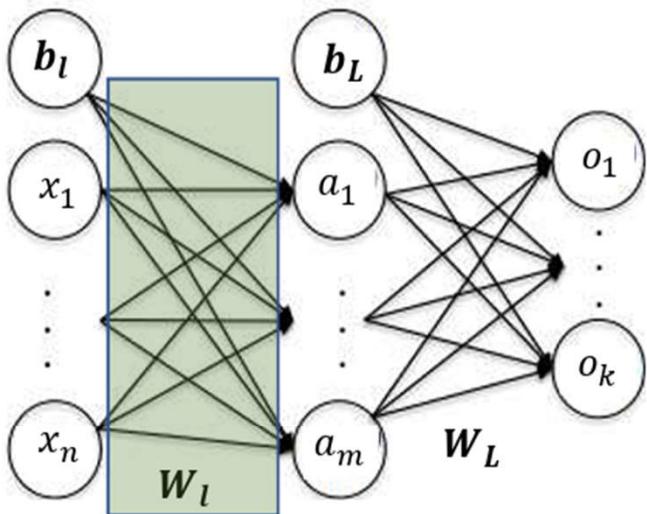
**Úprava predpäťia siete poslednej vrstvy :**

$$b_j^{L(p)}(k+1) = b_j^L(k) - \alpha \frac{\partial E^{(p)}}{\partial b_j^L}(k)$$

Uplatníme reťazové pravidlo:

$$\frac{\partial E^{(p)}}{\partial b_j^L} = \frac{\partial E^{(p)}}{\partial net_j^{L(p)}} \frac{\partial net_j^{L(p)}}{\partial b_j^L} \quad \text{pretože } E^{(p)} = f(net_j^{L(p)}) \text{ a } net_j^{L(p)} = g(b_j^L)$$

# MLP s jednou skrytou vrstvou



Opakovane uplatníme reťazové pravidlo:

$$\frac{\partial E^{(p)}}{\partial w_{ji}^l} = \frac{\partial E^{(p)}}{\partial net_j^{l(p)}} \frac{\partial net_j^{l(p)}}{\partial w_{ji}^l} \quad \text{pretože } E^{(p)} = f\left(net_j^{l(p)}\right) \text{ a } net_j^{l(p)} = g(w_{ji}^l)$$

$$\frac{\partial E^{(p)}}{\partial net_j^{l(p)}} = \frac{\partial E^{(p)}}{\partial a_j^{(p)}} \frac{\partial a_j^{(p)}}{\partial net_j^{l(p)}} \quad \text{pretože } E^{(p)} = f\left(a_j^{(p)}\right) \text{ a } a_j^{(p)} = g(net_j^{l(p)})$$

$$\frac{\partial E^{(p)}}{\partial a_i^{(p)}} = \frac{\partial E^{(p)}}{\partial net_j^{L(p)}} \frac{\partial net_j^{L(p)}}{\partial a_i^{(p)}} \quad \text{pretože } E^{(p)} = f\left(net_j^{L(p)}\right) \text{ a } net_j^{L(p)} = g(a_i^{(p)})$$

$$\frac{\partial E^{(p)}}{\partial net_j^{L(p)}} = \frac{\partial E^{(p)}}{\partial o_j^{(p)}} \frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} \quad \text{pretože } E^{(p)} = f\left(o_j^{(p)}\right) \text{ a } o_j^{(p)} = g(net_j^{L(p)})$$

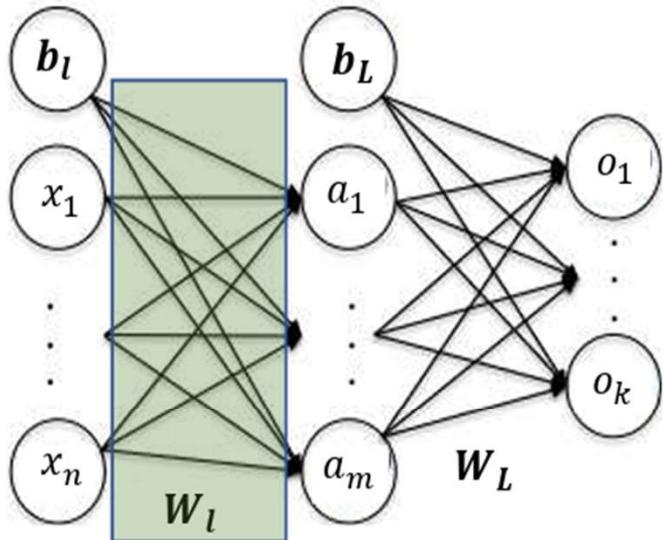
**Úprava váh siete prvej vrstvy :**

$$w_{ji}^{l(p)}(k+1) = w_{ji}^{l(p)}(k) - \alpha \frac{\partial E^{(p)}}{\partial w_{ji}^l}(k)$$

Po dosadení:

$$\frac{\partial E^{(p)}}{\partial w_{ji}^l} = \frac{\partial E^{(p)}}{\partial o_j^{(p)}} \frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} \frac{\partial net_j^{L(p)}}{\partial a_i^{(p)}} \frac{\partial a_i^{(p)}}{\partial net_j^{l(p)}} \frac{\partial net_j^{l(p)}}{\partial w_{ji}^l}$$

# MLP s jednou skrytou vrstvou



$$\frac{\partial E^{(p)}}{\partial o_j^{(p)}} = \frac{\partial}{\partial o_j^{(p)}} \left[ \frac{1}{2} \sum_{j=1}^k \left( d_j^{(p)} - o_j^{(p)} \right)^2 \right] = - \left( d_j^{(p)} - o_j^{(p)} \right) = -e_j^{(p)}$$

$$\frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} = \varphi'(net_j^{L(p)})$$

$$\frac{\partial net_j^{L(p)}}{\partial a_i^{(p)}} = \frac{\partial}{\partial a_i^{(p)}} \left[ \sum_{i=1}^m w_{ji}^L a_i^{(p)} + b_j^L \right] = w_{ji}^L$$

$$\frac{\partial a_i^{(p)}}{\partial net_j^{l(p)}} = \varphi'(net_j^{l(p)})$$

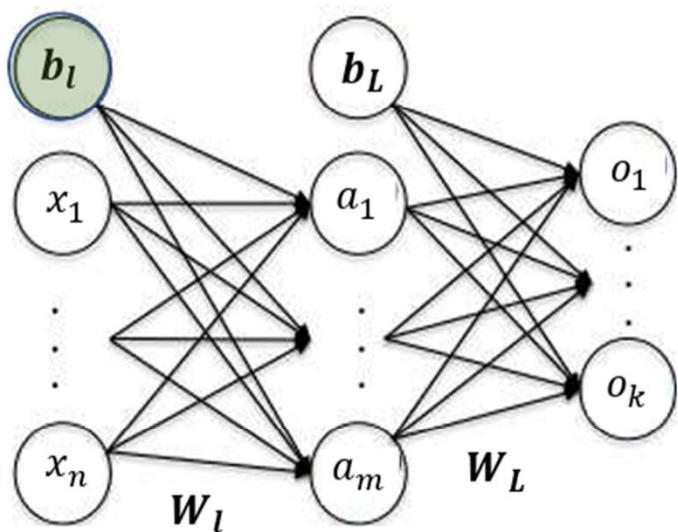
$$\frac{\partial net_j^{l(p)}}{\partial w_{ji}^l} = \frac{\partial}{\partial w_{ji}^l} \left[ \sum_{i=1}^m w_{ji}^l x_i^{(p)} + b_j^l \right] = x_i^{(p)}$$

$$w_{ji}^{l(p)}(k+1) = w_{ji}^l(k) + \alpha e_j^{(p)} \varphi'(net_j^{L(p)}) w_{ji}^L \varphi'(net_j^{l(p)}) x_i^{(p)}$$

↑                      ↓

Lokálny gradient j-teho skrytého neurónu

# MLP s jednou skrytou vrstvou



**Úprava predpäťia siete prvej vrstvy :**

$$b_j^{l(p)}(k+1) = b_j^l(k) - \alpha \frac{\partial E^{(p)}}{\partial b_j^l}(k)$$

Z predchádzajúceho kroku:

$$\frac{\partial E^{(p)}}{\partial b_j^l} = \frac{\partial E^{(p)}}{\partial o_j^{(p)}} \frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} \frac{\partial net_j^{L(p)}}{\partial a_i^{(p)}} \frac{\partial a_i^{(p)}}{\partial n_j^{l(p)}} \frac{\partial n_j^{l(p)}}{\partial b_j^l}$$

$$\frac{\partial E^{(p)}}{\partial o_j^{(p)}} = \frac{\partial}{\partial o_j^{(p)}} \left[ \frac{1}{2} \sum_{j=1}^k (d_j^{(p)} - o_j^{(p)})^2 \right] = - (d_j^{(p)} - o_j^{(p)}) = -e_j^{(p)}$$

$$\frac{\partial o_j^{(p)}}{\partial net_j^{L(p)}} = \varphi'(net_j^{L(p)})$$

$$\frac{\partial net_j^{L(p)}}{\partial a_i^{(p)}} = \frac{\partial}{\partial a_i^{(p)}} \left[ \sum_{i=1}^m w_{ji}^L a_i^{(p)} + b_j^L \right] = w_{ji}^L$$

$$\frac{\partial a_i^{(p)}}{\partial net_j^{l(p)}} = \varphi'(net_j^{l(p)})$$

$$\frac{\partial net_j^{l(p)}}{\partial b_j^l} = \frac{\partial}{\partial b_j^l} \left[ \sum_{i=1}^m w_{ji}^l x_i^{(p)} + b_j^l \right] = 1$$

$$b_j^{l(p)}(k+1) = b_j^l(k) + \alpha e_j^{(p)} \varphi'(net_j^{L(p)}) w_{ji}^L \varphi'(net_j^{l(p)})$$

## Slovné zhrnutie algoritmu spätného šírenia chyby.

1. Inicializácia – váhy a prahy sa nastavia na malé náhodné čísla
2. Postupne sa pre všetky vzorky (vektory) trénovacieho súboru dát vykoná nasledovný cyklus dopredného šírenia signálu a následne spätného šírenia poruchy – body 2a, 2b (bod 2 sa nazýva jedna „epocha“)
  - 2a. Na vstup siete sa priviedie jeden vstupný vektor (jedna vzorka, krok) a vypočítajú sa výstupy siete a výstupná chyba pre každý výstup ako aj suma kvadrátov všetkých výstupných chýb
  - 2b. Späťne sa počítajú všetky lokálne gradienty od výstupov ku vstupom a korigujú sa všetky váhy synaptických spojení a prahy neurónov
3. Pokial' sa nevykoná predpísaný počet trénovacích epoch alebo globálna chyba nepoklesne pod určenú hranicu, pokračuje sa bodom 2, inak sa ukončí trénovanie.

## Ukončenie procesu trénovania

- a) Dosiahnutie predpísanej presnosti modelu (globálnej chyby)
- b) Uskutočnenie predpísaného počtu epoch trénovania
- c) Stagnácia priebehu chyby (gradient chyby dosiahne def. úroveň)
- d) Alebo po úspešnom teste zovšeobecňovacej schopnosti n.s.

Proces trénovania je potrebné včas ukončiť, v opačnom prípade môže nastať stav "pretrénovania", keď sa parametre n.s. už prestanú zlepšovať a začínajú na ne mať vplyv rôzne parazitné signály ako šum a pod., ktoré môžu následne zhoršiť vlastnosti siete (kopírovanie šumu ...).

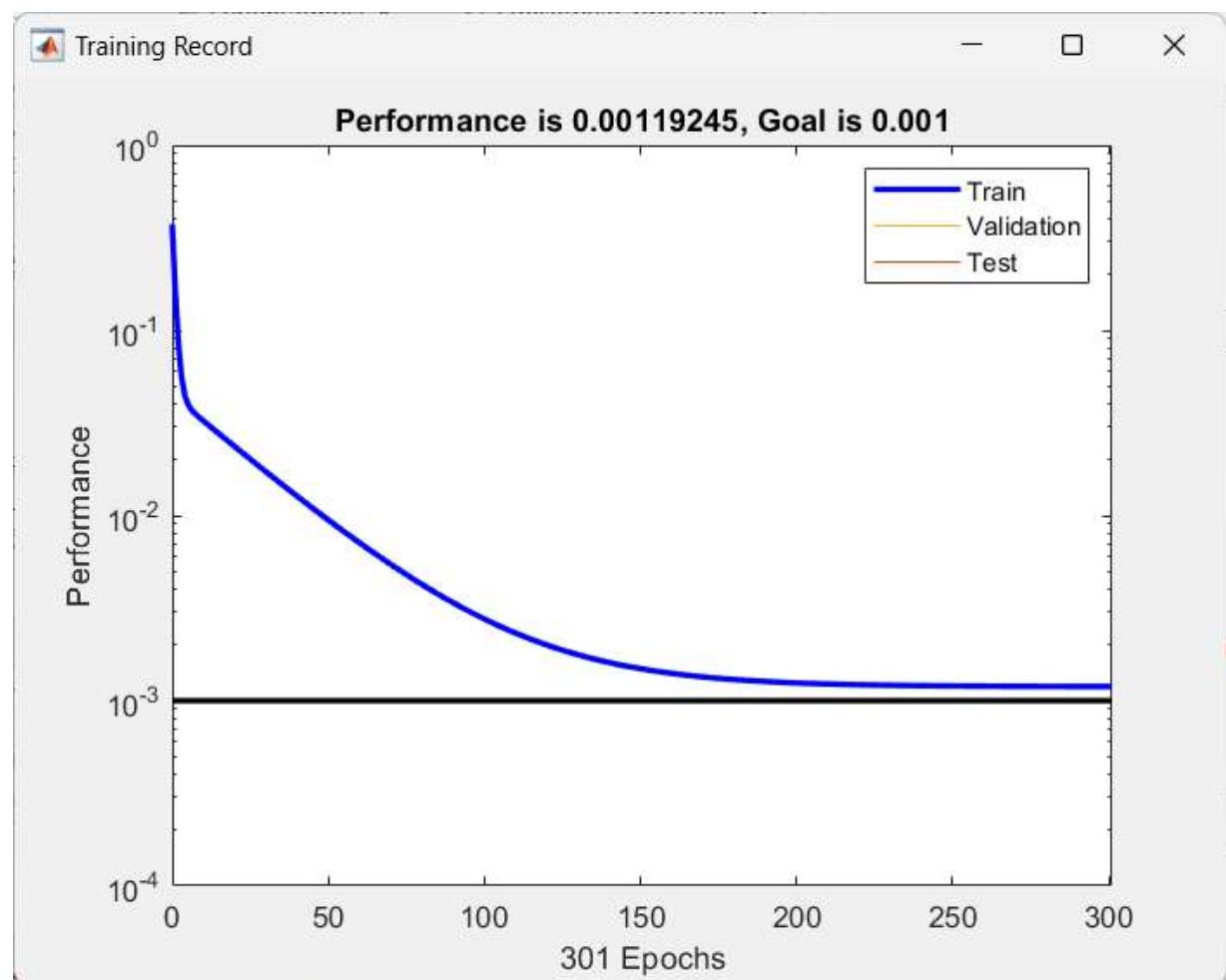
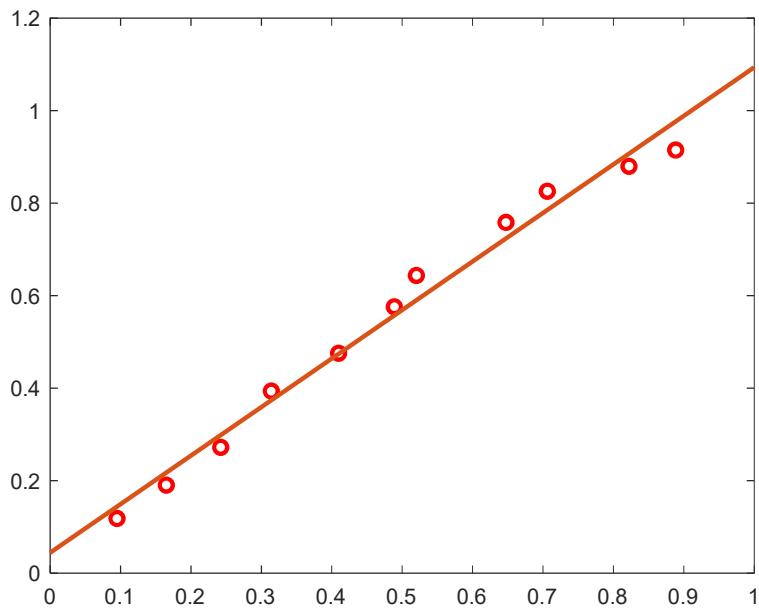
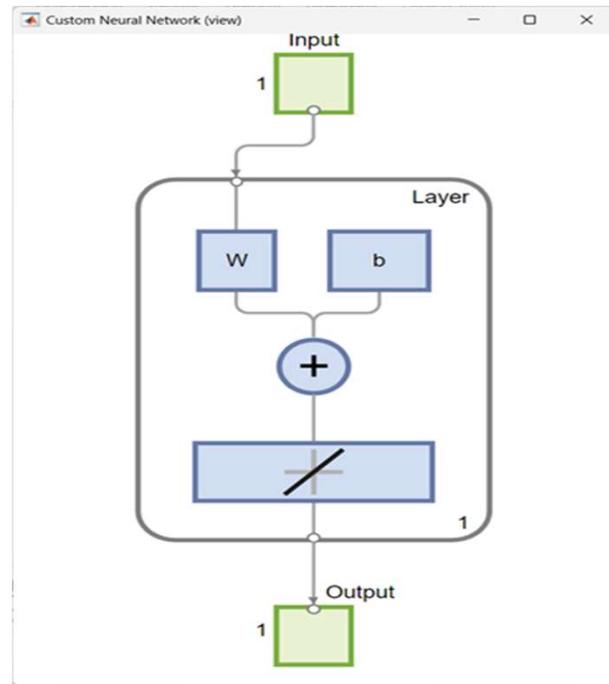
Počas trénovania sa vyhodnocuje chyba na inom balíku dát, ak chyba začne od určitej etapy trénovania narastať – trénovanie sa zastaví

# **Režim trénovania**

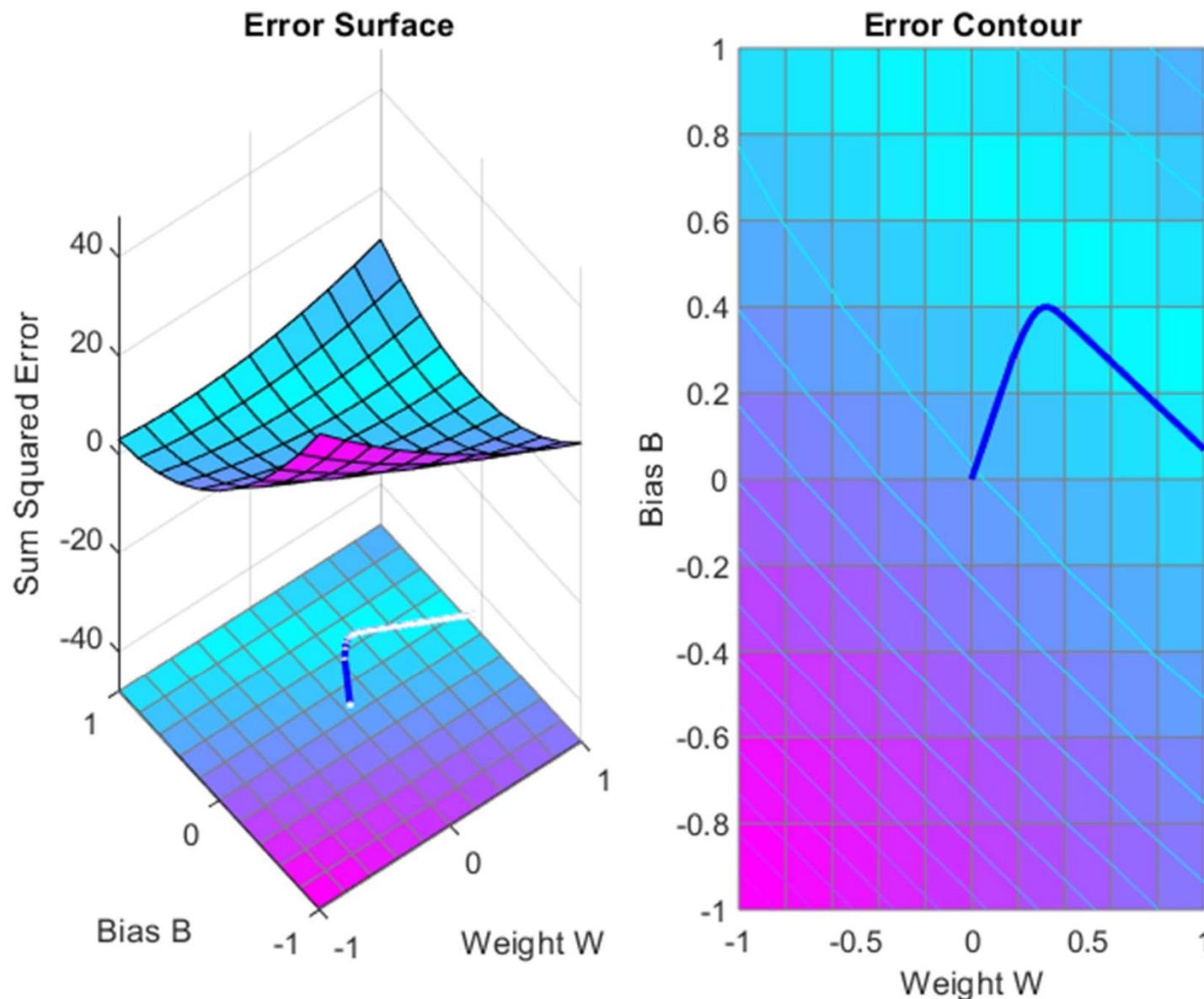
**Vzorkový režim trénovania** - váhy siete sa upravujú po privedení každej novej vzorky dát.

**Dávkový režim trénovania** - váhy sa korigujú až po celej epoche - po jednom kompletom prechode celého súboru trénovacích dát cez siet', využitím globálnej chyby zo všetkých vzoriek a výstupov.

# Ukážka trénovalia jedného neuróna



# Ukážka trénovania jedného neuróna

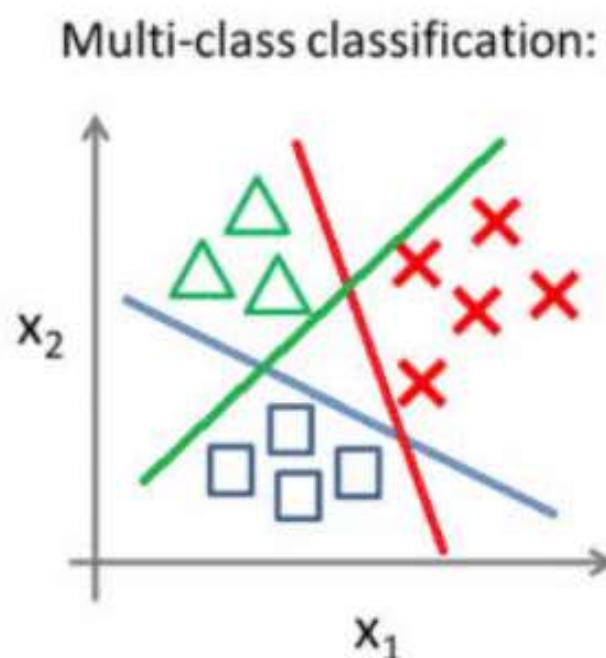
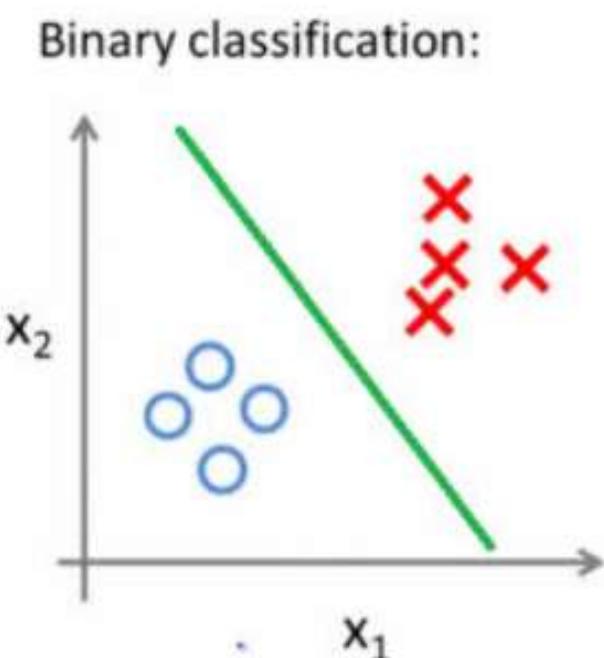


# **Typy aplikácií MLP v praxi**

- **Klasifikácie do tried**
- **Rozpoznávanie**
- **Aproximácia**
- **Modelovanie**
- **Predikcia**

# Čo je klasifikácia

**Klasifikácia** - je proces súvisiaci s kategorizáciou, triedením. Klasifikácia je proces, v ktorom sa idey, objekty alebo predmety rozpoznávajú, rozlišujú a delia do tried (skupín) na základe súvisiacich faktov, parametrov, vlastností. Môže sa to týkať aj procesu, ktorý spája podobné veci a oddeluje rozdielne veci.



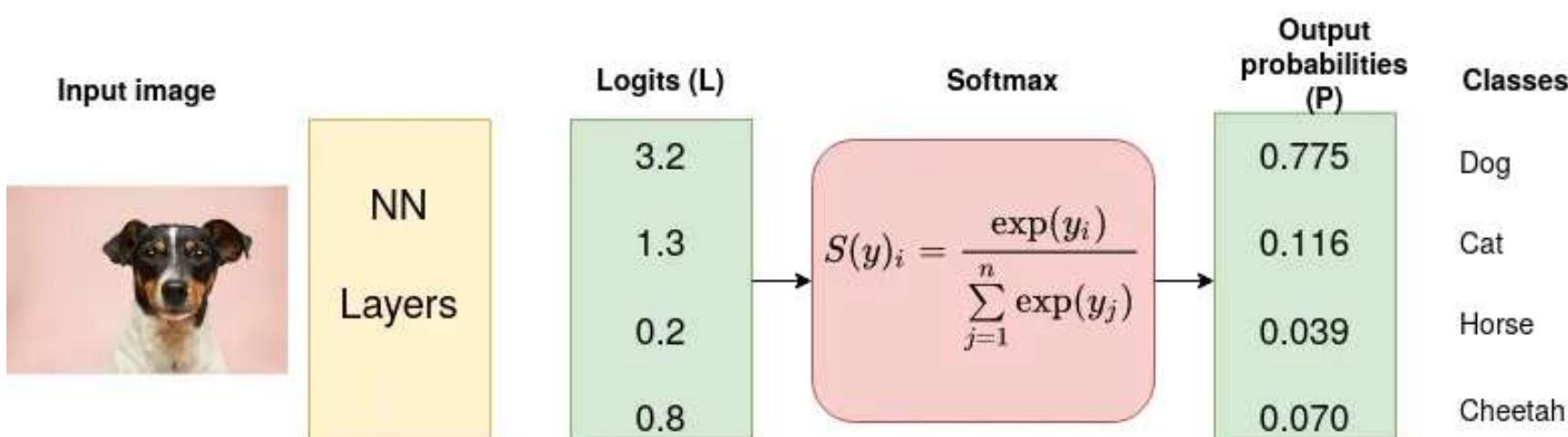
# Klasifikácia – softmax aktivačná funkcia

Softmax - je aktivačná funkcia, ktorá škáluje vstupné čísla na pravdepodobnosti. Výstupom je vektor s pravdepodobnosťou priradenia do jednotlivých výstupov (tried). Súčet pravdepodobností vo vektore pre všetky možné triedy sa rovná 1.

Definícia:

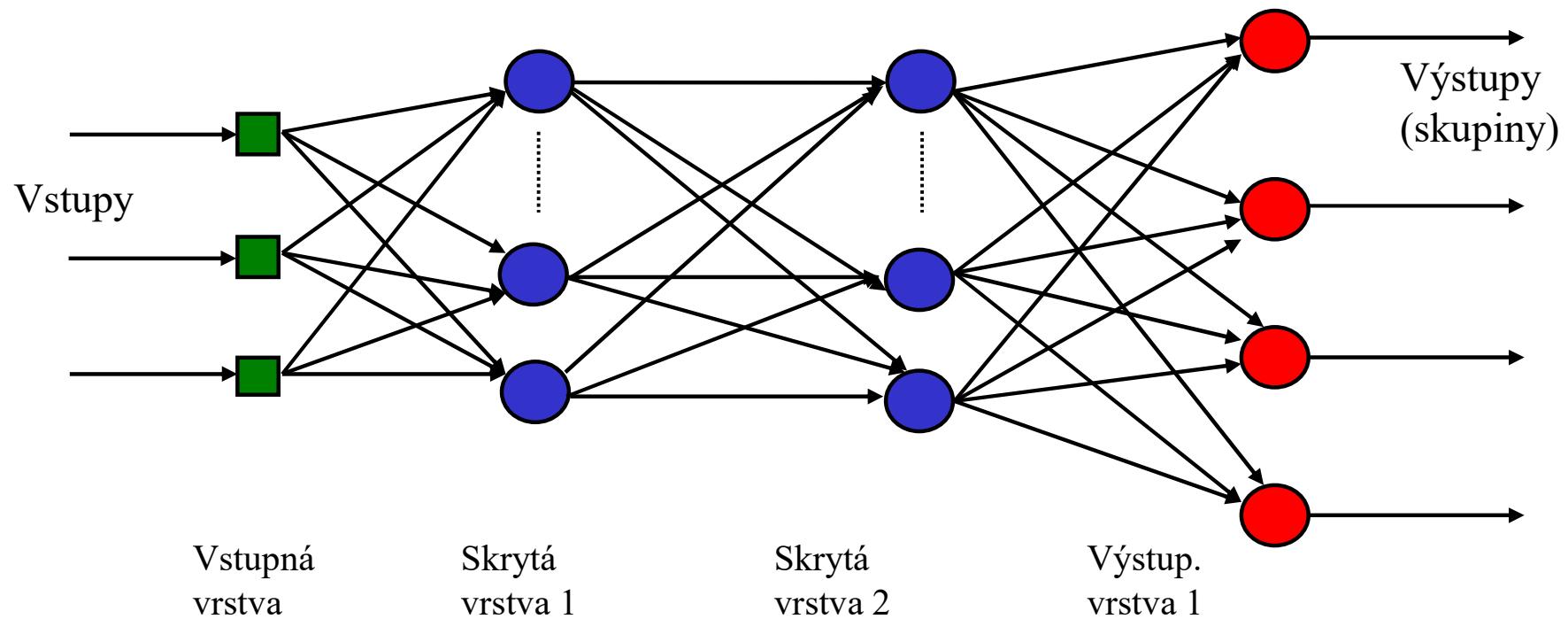
y – vstupný vektor,  $y_i$  – i-ty prvok vstupného vektora, n – počet tried, S – vektor pravdepodobností

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^n \exp(y_j)}$$



# Klasifikácia – MLP siet'

## Štruktúra MLP siete pre klasifikáciu, rozpoznávanie (patternnet)



- Vstupy NS sú parametre na základe, ktorých realizujeme triedenie do skupín
- Výstupy NS prislúchajú k jednotlivým skupinám, t.j. počet výstupov sa rovná počtu skupín, rozsah výstupu je  $<0,1>$  a vyjadruje pravdepodobnosť zatriedenia do skupiny.
- Aktivačné funkcie vo všetkých skrytých vrstvách môžu byť „tansig“ , „logsig“ alebo ReLu, vo výstupnej vrstve sa často používa „softmax“
- Počty skrytých vrstiev ako aj počty neurónov v nich sa definuje experimentálne podľa zložitosti klasifikačnej úlohy.

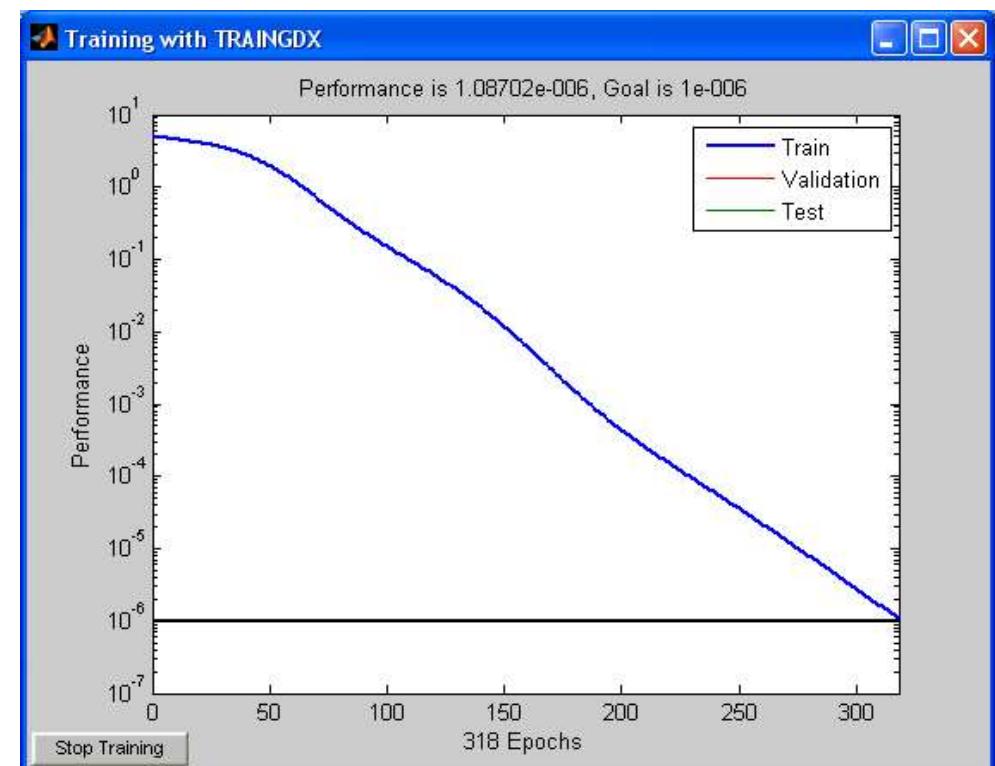
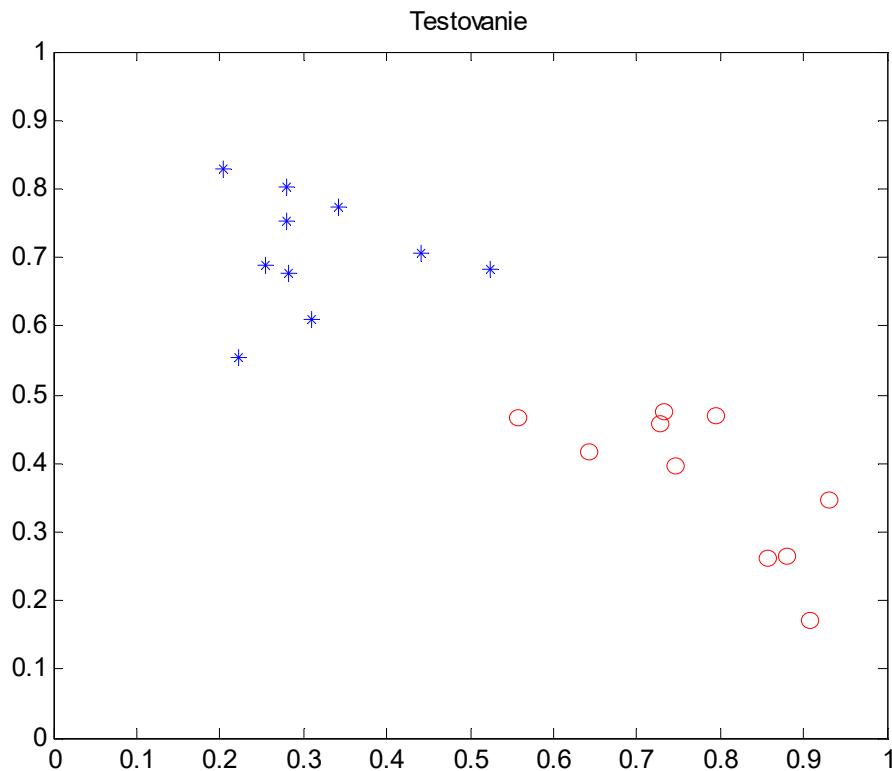
# Príklad: klasifikácia objektov do dvoch tried

% vytvorenie štruktúry MLP siete s 2 vstupmi s rozsahmi (0,1), skrytou vrstvou a 1 výstupom, aktivačné funkcie ‘tansig’ a ‘softmax’

```
net = patternnet(3);
```

% trénovanie siete a simulácia výstupu NS

```
net = train(net,X,P); f = sim(net,[0.7,0.3]);
```



## Lineárne separovateľné skupiny bodov

```
% zadanie bodov myšou
```

```
[xT,yT]=ginput
```

```
% simulácia výstupu NS
```

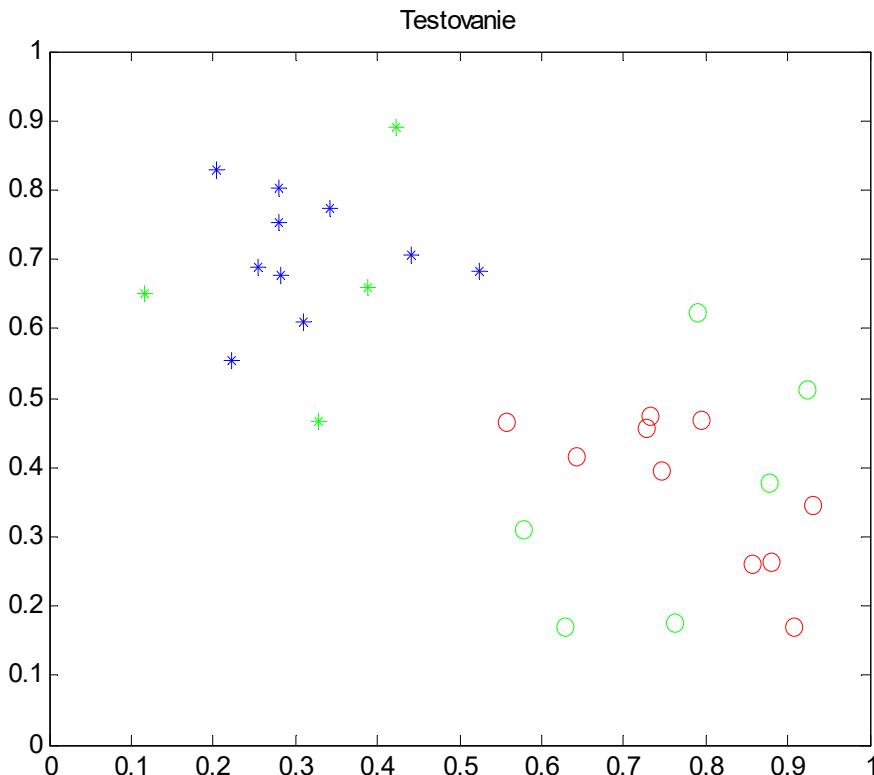
```
predmet=sim(net,[xT';yT' ]);
```

```
% priradenie bodu farbu
```

```
for k=1:length(predmet),
```

```
    if predmet(k)<0.4
```

```
...
```



```
% vygenerovanie mriežky bodov
```

```
[xTpom,yTpom]=meshgrid(0:0.05:1,0:0.05:1);
```

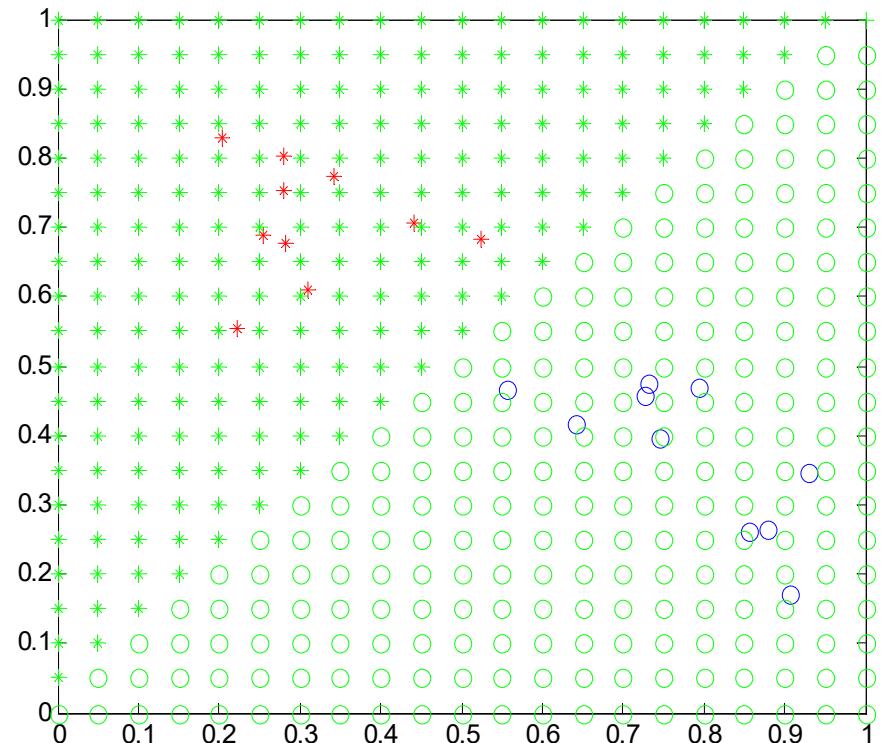
```
xT2=xTpom(1:end); yT2=yTpom(1:end);
```

```
predmet2=sim(net,[xT2; yT2]);
```

```
for k=1:length(predmet2),
```

```
    if predmet2(k)<0.4
```

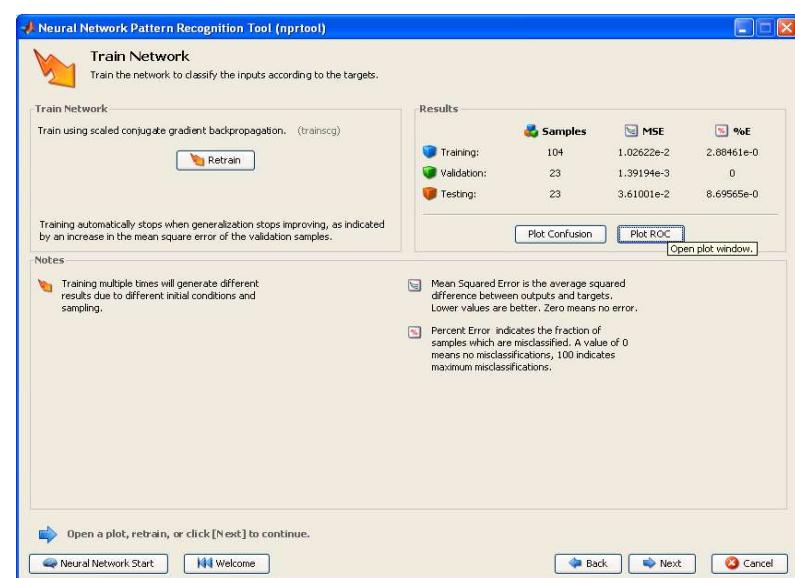
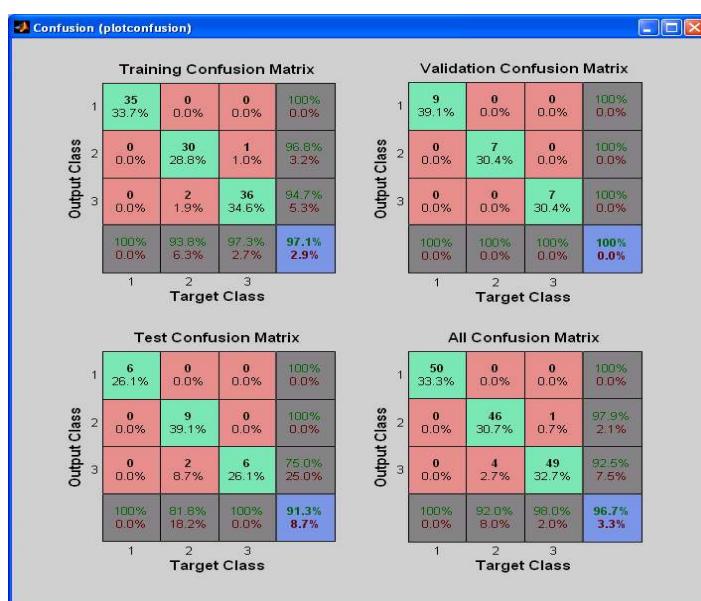
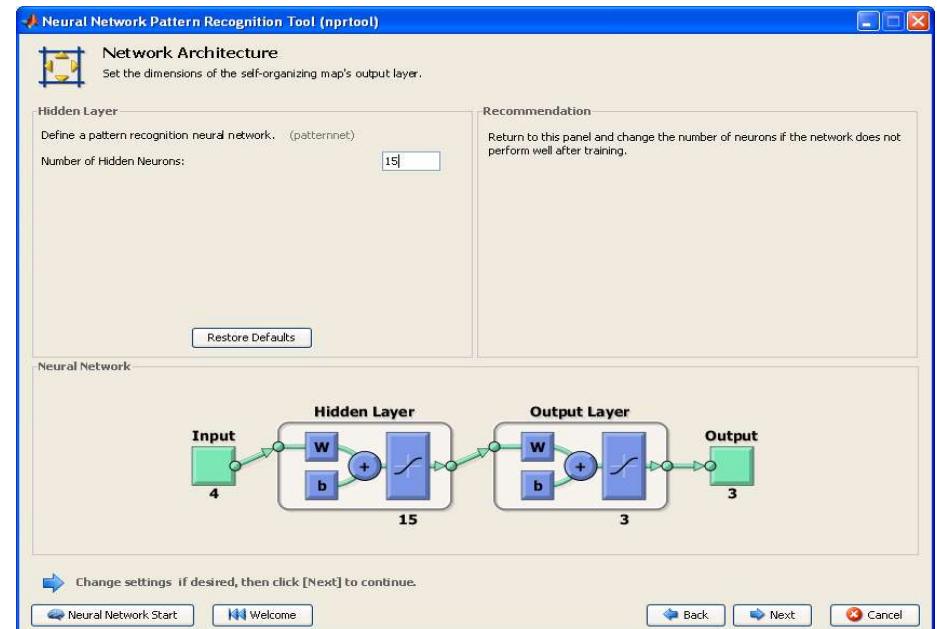
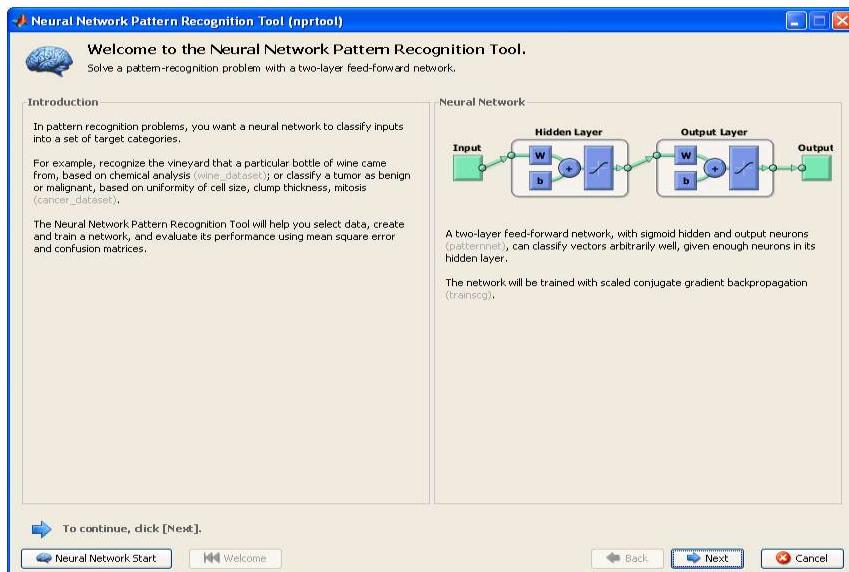
```
....
```



# Klasifikácia reálnych dát

## Grafické prostredie pre klasifikáciu (Pattern recognition tool)

*nprtool*



# *Reálne dátá*

- cancer\_dataset - Breast cancer dataset.  
(Data 699 vzoriek – 9 parametrov z biopsie – výstup je nezhubný/zhubný nález buniek)
- crab\_dataset - Crab gender dataset.  
(Data 200 vzoriek – 6 fyzikálnych parametrov krabu – výstup je pohlavie krabu)
- glass\_dataset - Glass chemical dataset.  
(Data 214 vzoriek – 9 parametrov skla – výstupom je vhodnosť skla pre okno A/N)
- iris\_dataset - Iris flower dataset.  
(Data 1000 vzoriek – 4 parametre kvetu Iris – výstup je druh kvetu – 3 druhy kvetu)
- thyroid\_dataset - Thyroid function dataset.  
(Data 7200 vzoriek – 21 parametrov o pacientoch – výstup je stav pacienta – 3 stavy (normal, hyperfunkcia, subnormal))
- wine\_dataset - Italian wines dataset.  
(Data 178 vzoriek – 13 parametrov z chemickej analýzy vín – výstup druh vína, 3 druhy)

## *Príklad: Dáta pre Iris kvety*

Dáta pre klasifikáciu Iris kvetov do troch skupín (150 vzoriek od 1000 kvetov):

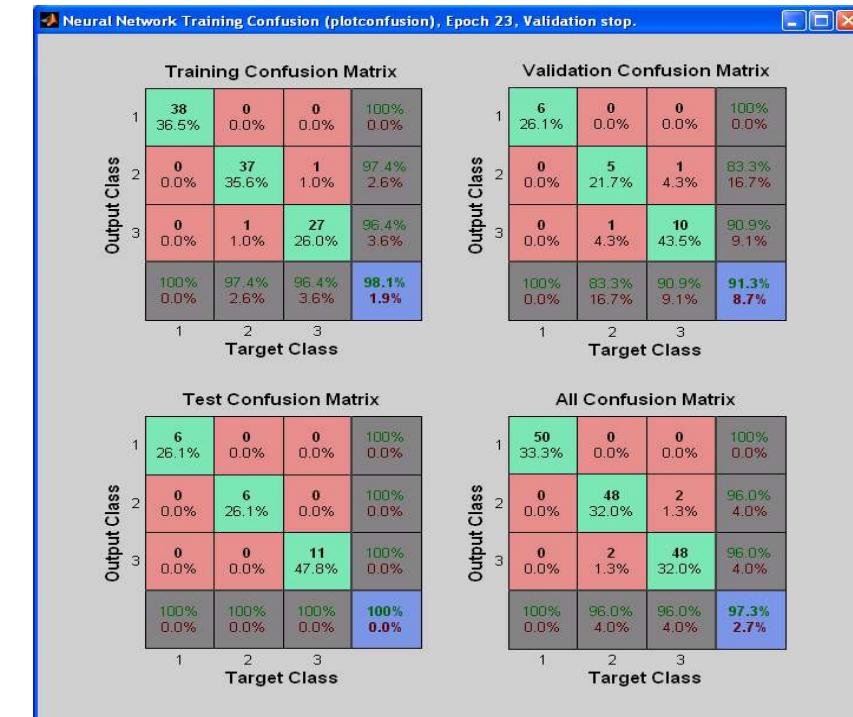
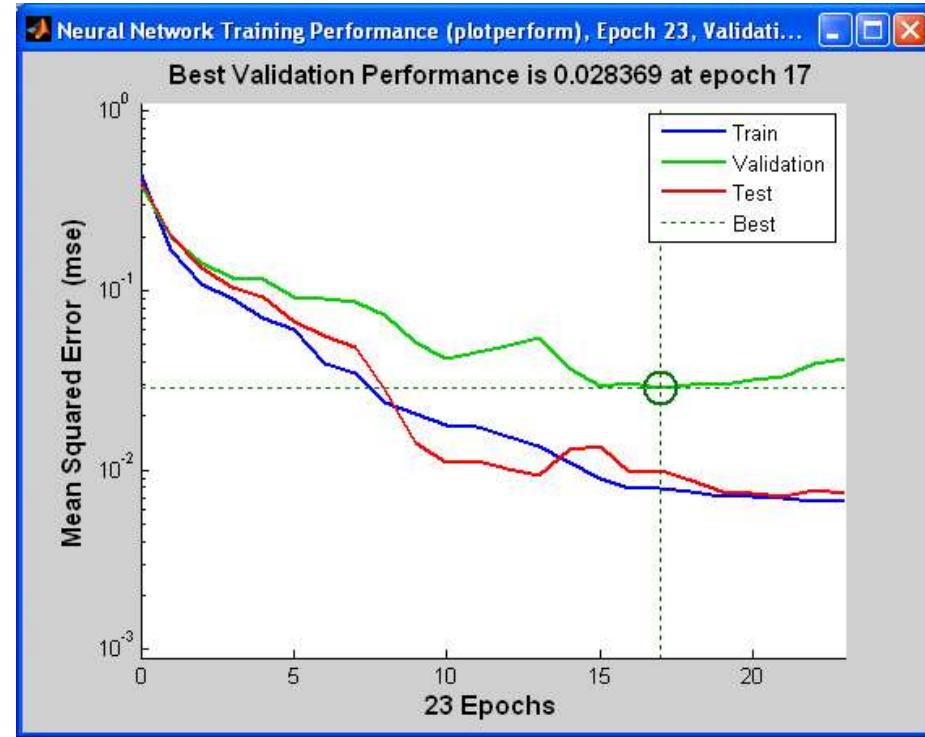
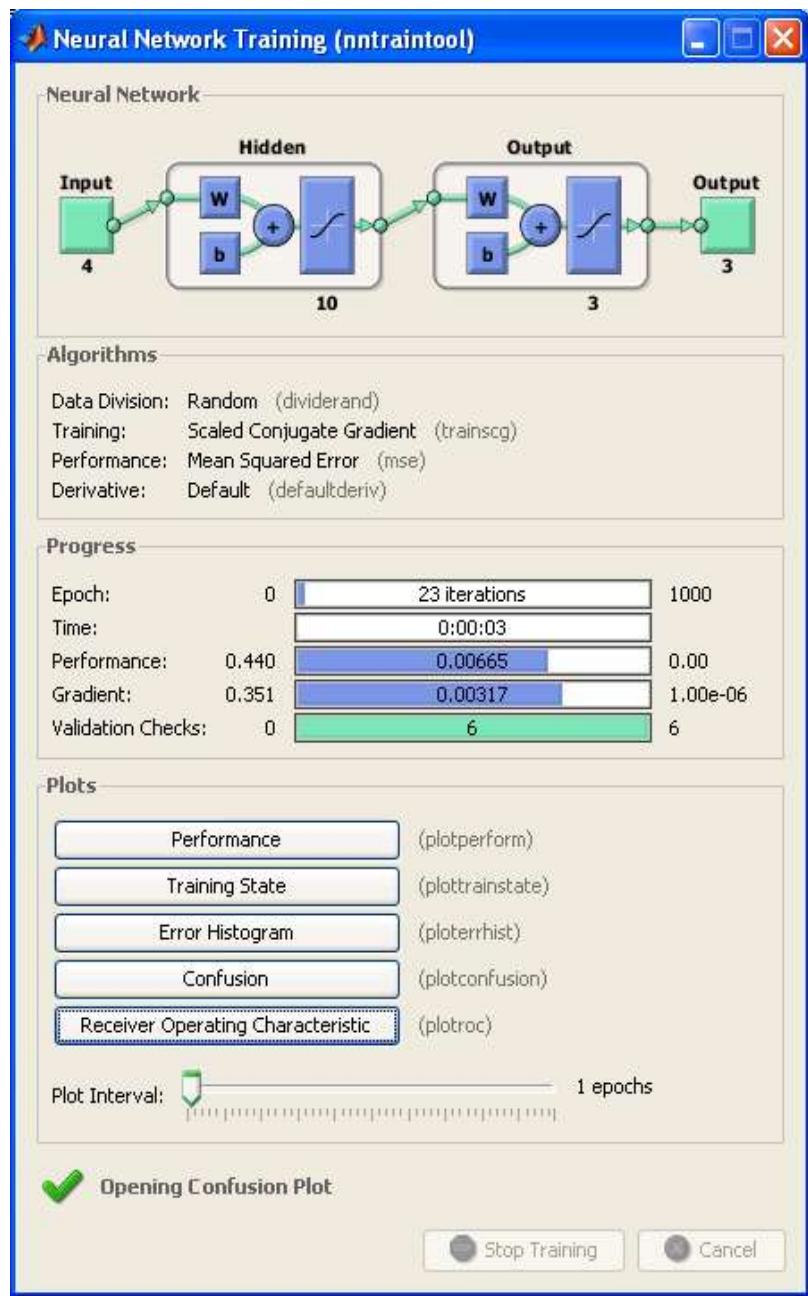
Vstupy:

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm



```
[x,t] = iris_dataset; % načítanie dát  
net = patternnet(10); % vytvorenie NS  
net = train(net,x,t); % trénovanie  
view(net) % štruktúra siete  
y = net(x); % simulácia výstupu NS  
plotconfusion(t,y) % vyhodnotenie NS
```

# NS pre Iris kvety

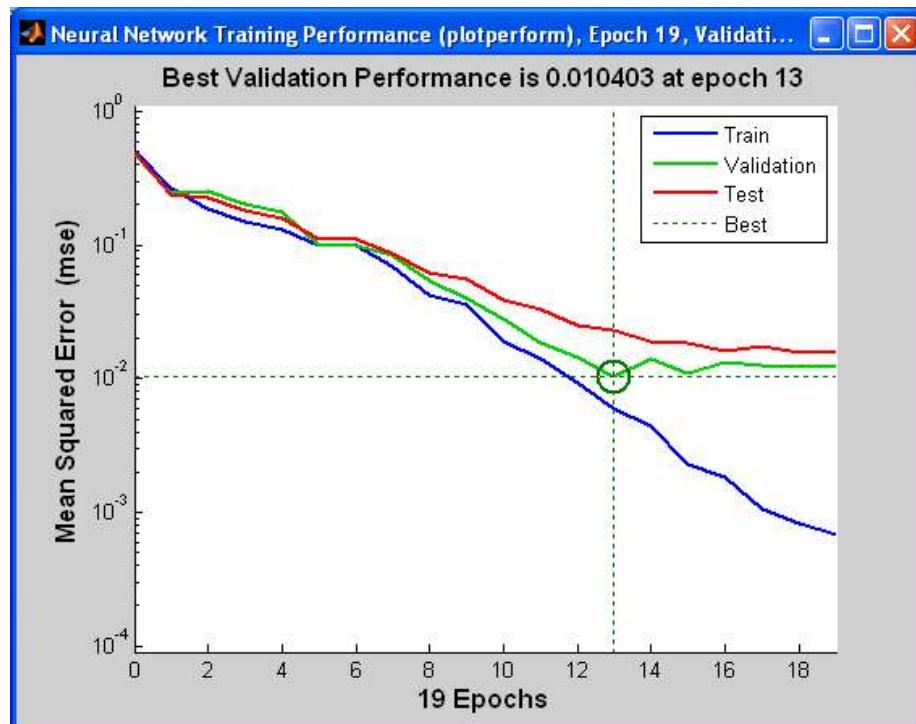
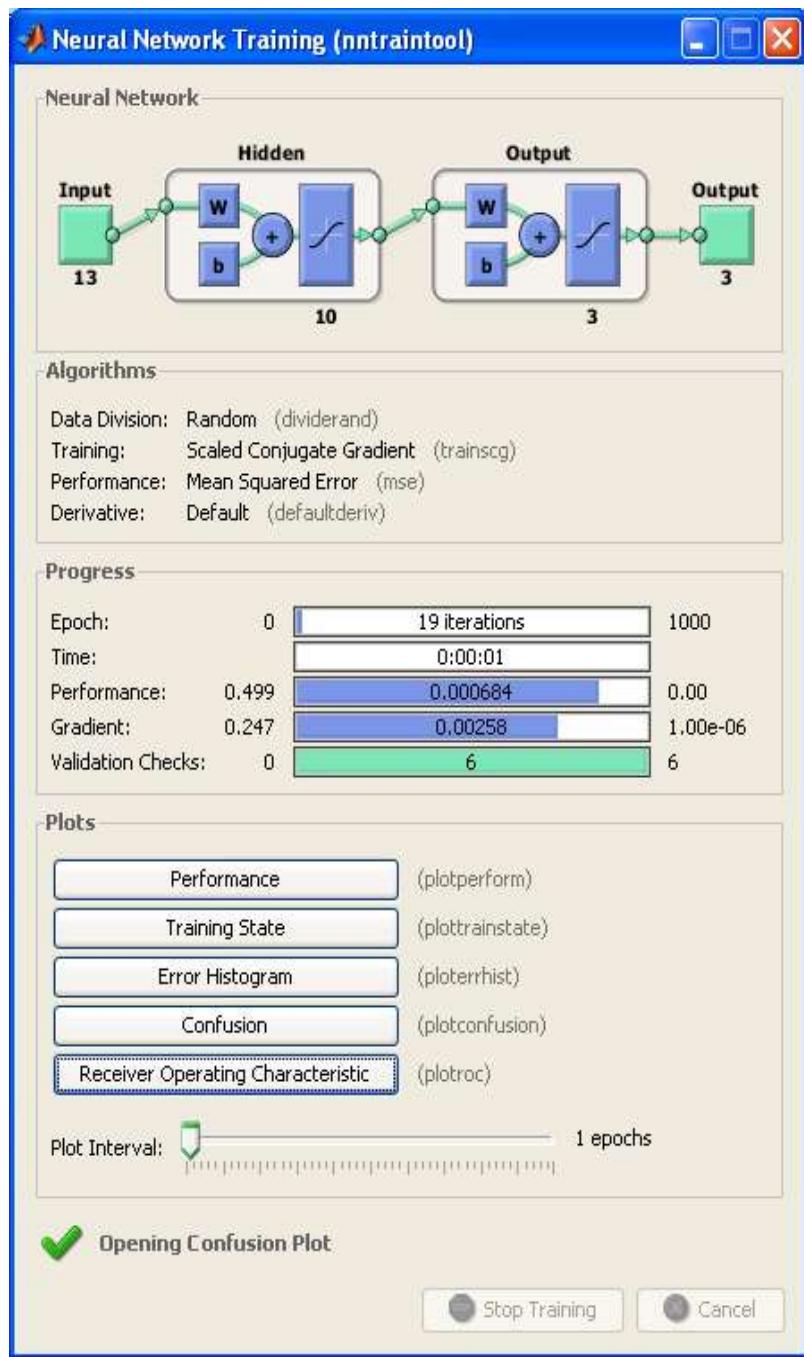


## *Priklad: Dáta pre Talianske vína*

Dáta pre klasifikáciu Talianskych vín do troch skupín  
(178 vzoriek – 13 parametrov z chemickej analýzy vín):

- |                                  |                       |
|----------------------------------|-----------------------|
| 1. Alcohol                       |                       |
| 2. Malic acid                    | [x,t] = wine_dataset; |
| 3. Ash                           | net = patternnet(10); |
| 4. Alcalinity of ash             | net = train(net,x,t); |
| 5. Magnesium                     | view(net)             |
| 6. Total phenols                 | y = net(x);           |
| 7. Flavanoids                    | plotconfusion(t,y)    |
| 8. Nonflavanoid phenols          |                       |
| 9. Proanthocyanins               |                       |
| 10. Color intensity              |                       |
| 11. Hue                          |                       |
| 12. OD280/OD315 of diluted wines |                       |
| 13. Proline                      |                       |

# NS pre Talianske vína



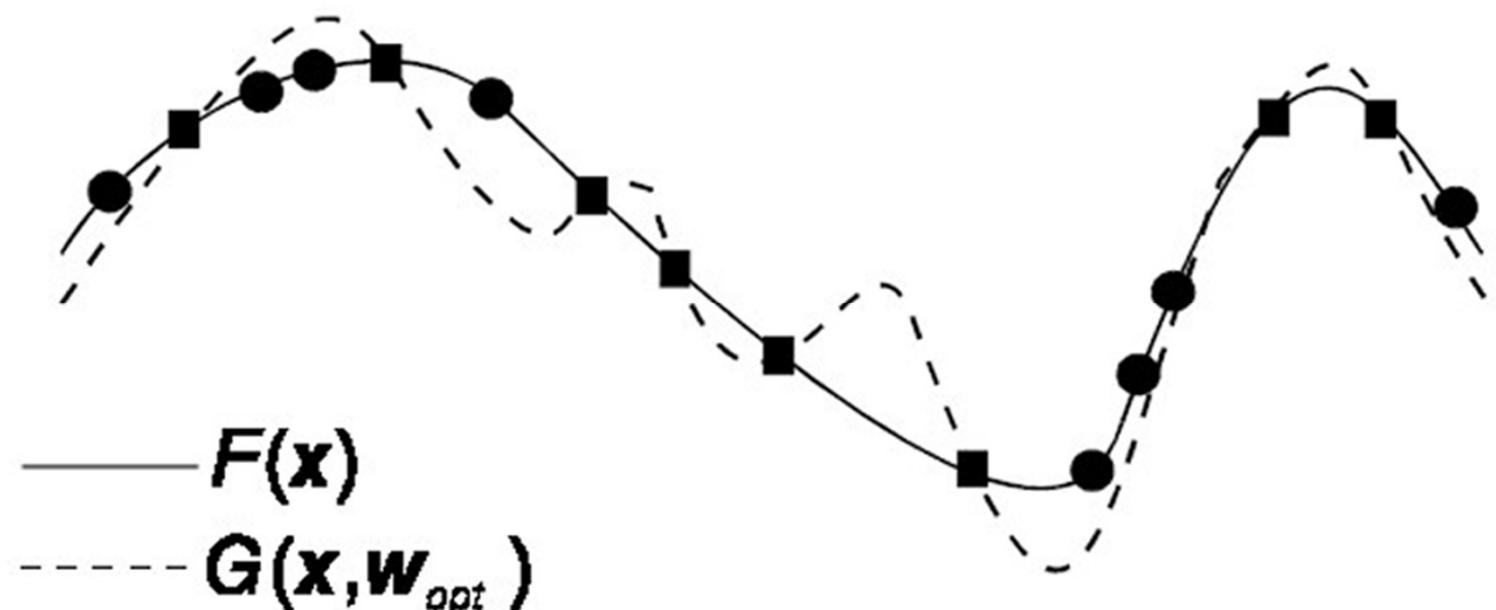
# Čo je aproximácia

Aproximácia - predstavuje proces odhadu nelineárnej závislosti výstupných veličín systému (funkcie) od vstupných pomocou MLP sieti. Cieľom je minimalizovať odchýlku medzi výstupom systému  $Y_s$  a výstupom neurónovej siete  $Y_n$ .

$$Y_s = F(x)$$

$$Y_n = G(x, w)$$

Hľadáme optimálne váhy



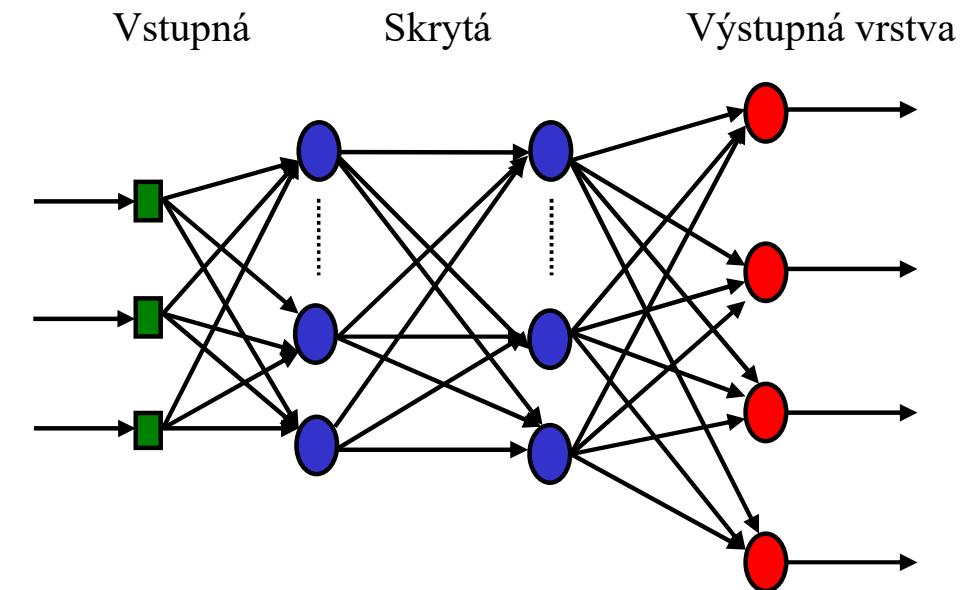
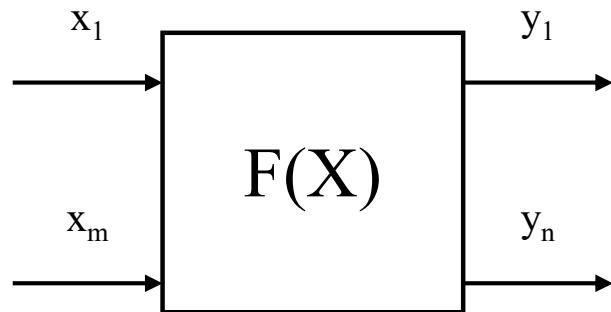
$$E = \sum (Y_s - Y_n)^2 \rightarrow \min$$

# Aproximácia, modelovanie – MLP siet'

## Štruktúra MLP siete pre aproximáciu, modelovanie (fitnet)

Aproximovaná funkcia, modelovaný systém:

$Y=F(X)$ , Y-výstupy, X-vstupy, F-nelineárna funkcia,  $X=[x_1, \dots, x_m]$ ,  $Y=[y_1, \dots, y_n]$



- Vstupy NS sú vstupy systému (funkcie), výstupy NS sú výstupy systému (funkcie)
- Aktivačné funkcie v skrytých vrstvách sú „tansig“ alebo „ReLU“ a vo výstupnej „purelin“.
- Počty skrytých vrstiev ako aj počty neurónov v nich sa definuje experimentálne podľa zložitosti modelovaného systému.

# Aproximácia nelineárnej funkcie $y=f(x)$ z meraných dát pomocou UNS

```
[x,y]=simplefit_dataset;
```

```
net=fitnet(12);
```

```
net.divideFcn='divideint';
```

```
net.divideParam.trainRatio=0.8;
```

```
net.divideParam.valRatio=0.1;
```

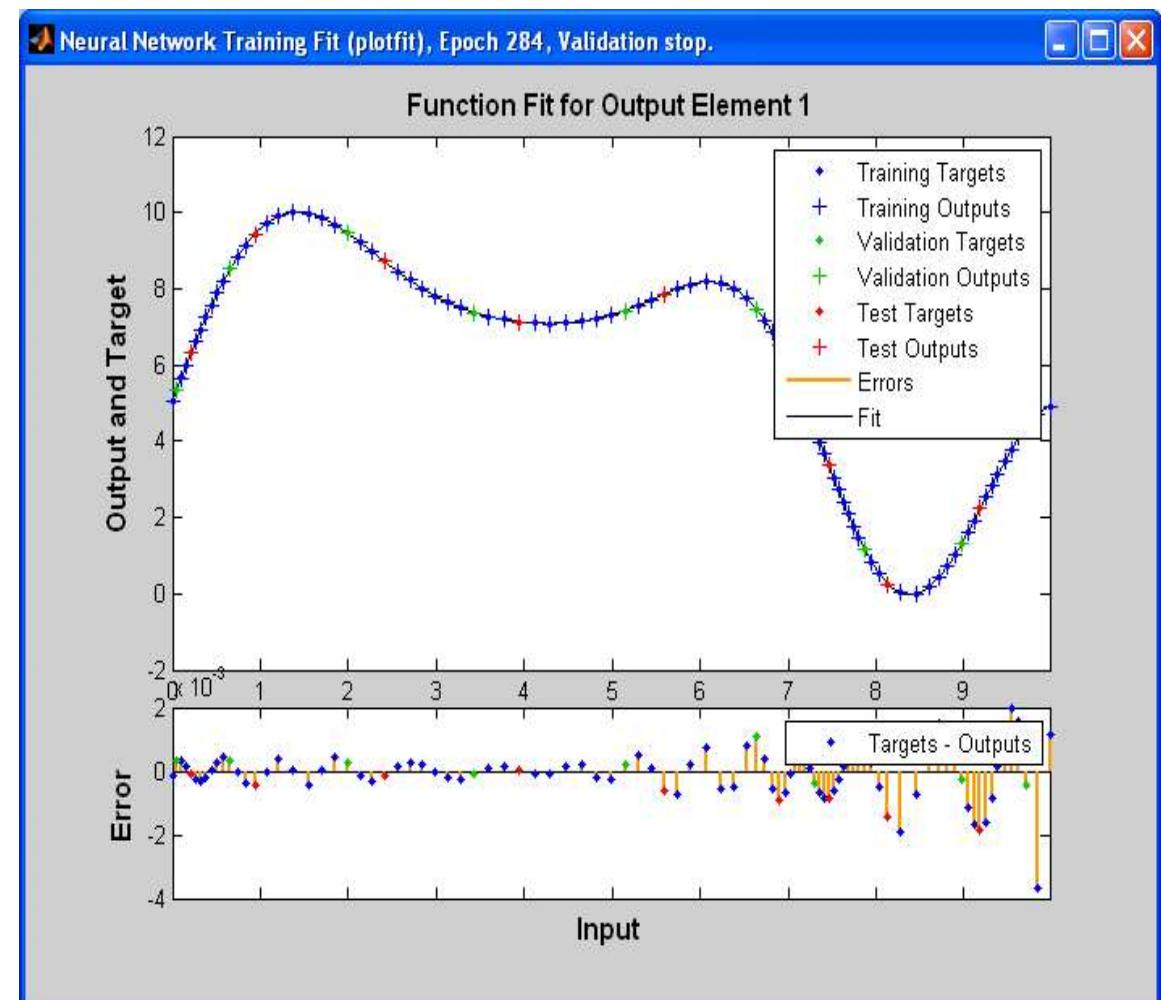
```
net.divideParam.testRatio=0.1;
```

```
net.trainParam.goal = 1e-7;
```

```
net.trainParam.show = 5;
```

```
net.trainParam.epochs = 100;
```

```
net=train(net,x,y);
```



## Spôsoby generovania (rozdeľovania) trénovacích, validačných a testovacích dát

Typ funkcie na generovanie dát  
**net.divideFcn**

net.divideFcn='dividerand'; % náhodné rozdelenie  
net.divideFcn='divideblock'; % rozdelenie po blokoch dát za sebou  
net.divideFcn='divideint'; % je použitá každá n-tá vzorka  
net.divideFcn='dividetrain'; % všetky dáta sú iba trénovacie

% parametre rozdelenia dát  
net.divideParam.trainRatio=0.8;  
net.divideParam.valRatio=0.1;  
net.divideParam.testRatio=0.1;

## **Spôsoby generovania (rozdeľovania) trénovacích, validačných a testovacích dát**

Typ funkcie na generovanie dát  
**net.divideFcn**

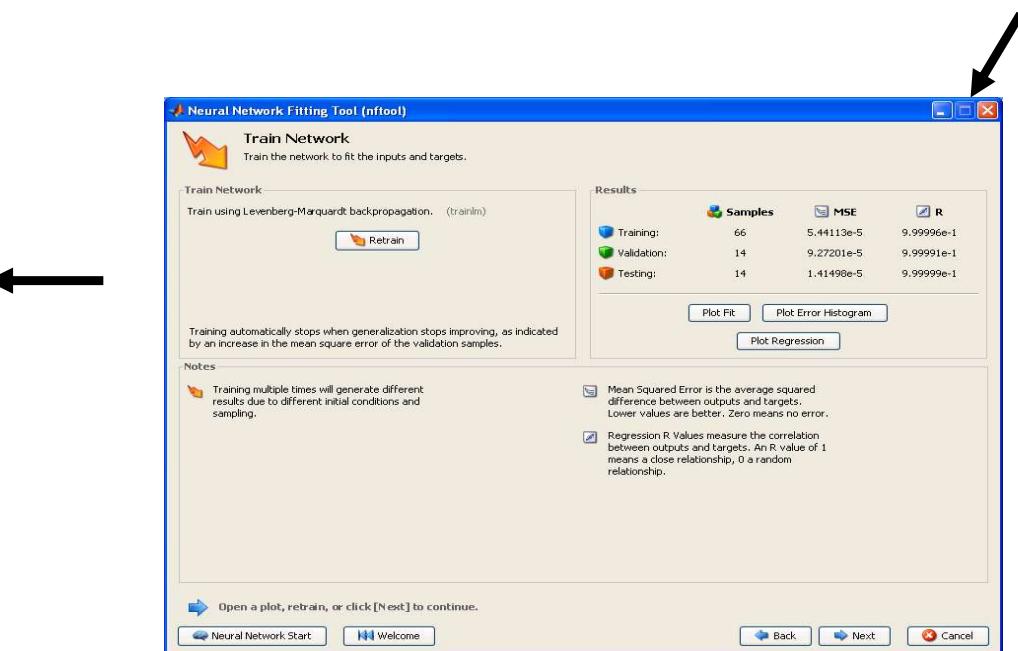
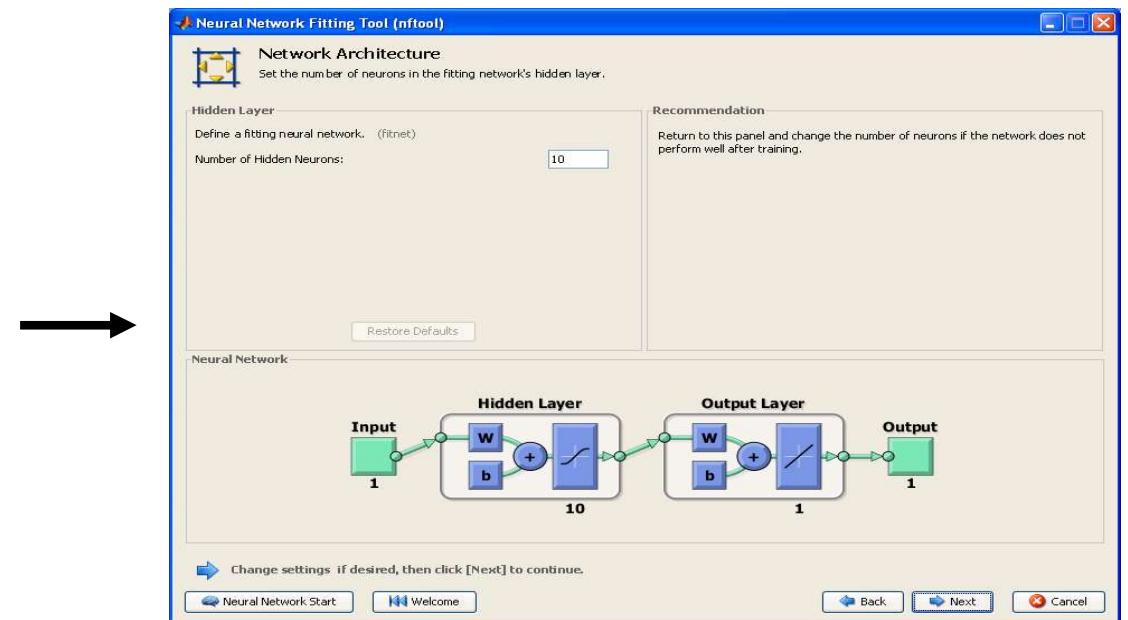
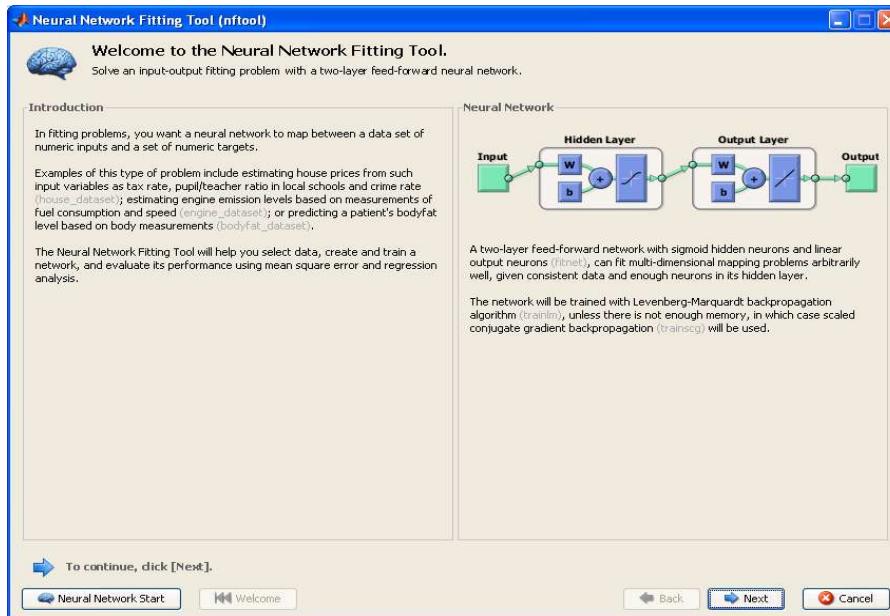
net.divideFcn='divideind'; % indexové rozdelenie

% parametre rozdelenia dát  
net.divideParam.trainInd=1:2:n;  
net.divideParam.valInd=2:2:n2;  
net.divideParam.testInd=n2+1:2:n;

# Modelovanie (aproximácia) reálnych dát

## Grafické prostredie pre approximáciu (Neural Network Fitting Tool)

### *nftool*



## *Reálne dátá*

- simplefit\_dataset
  - Simple fitting dataset.
- abalone\_dataset
  - Abalone shell rings dataset.
- bodyfat\_dataset
  - Body fat percentage dataset.
- building\_dataset
  - Building energy dataset.
- chemical\_dataset
  - Chemical sensor dataset.
- cho\_dataset
  - Cholesterol dataset.
- engine\_dataset
  - Engine behavior dataset.
- house\_dataset
  - House value dataset.

## *Priklad: Dáta pre Cholesterol*

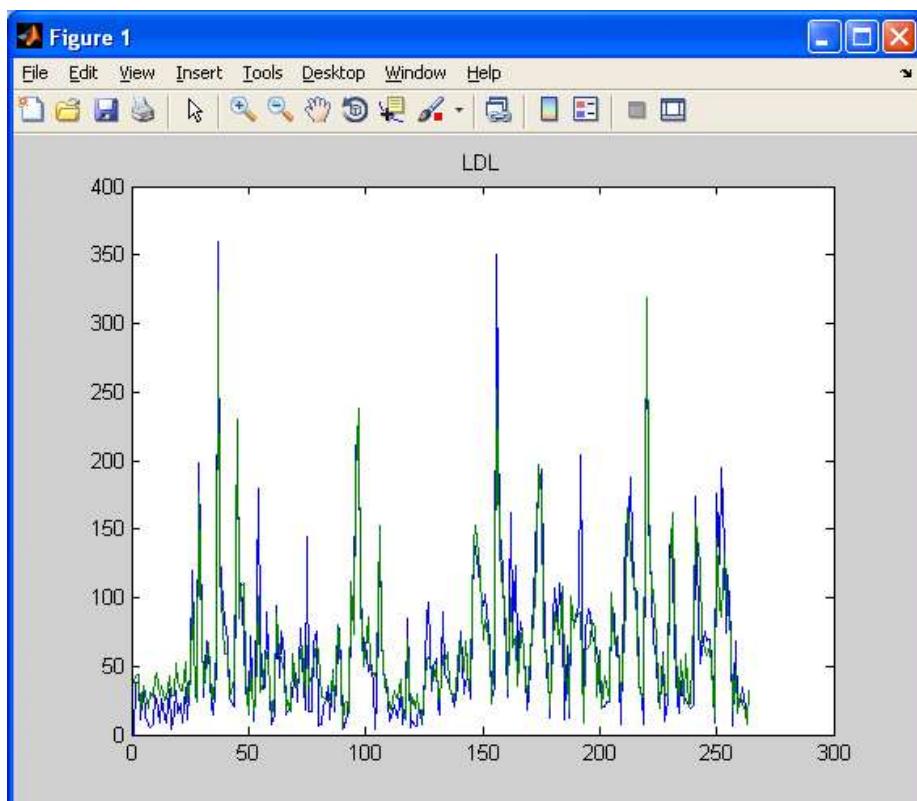
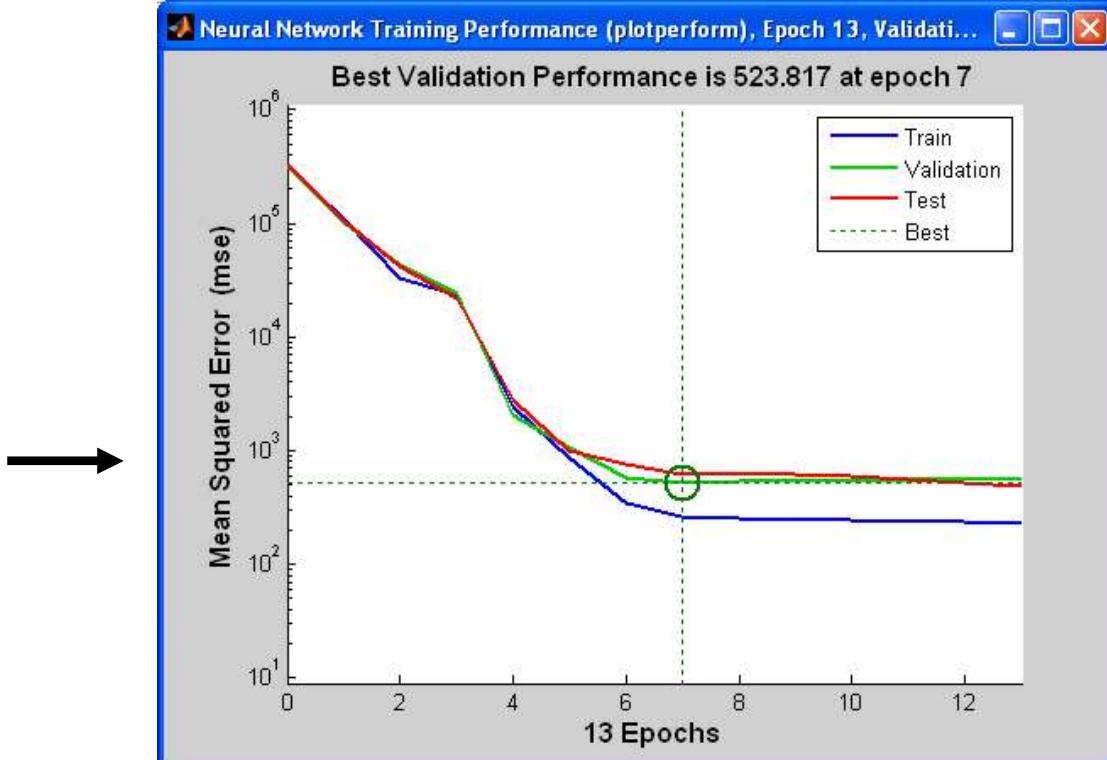
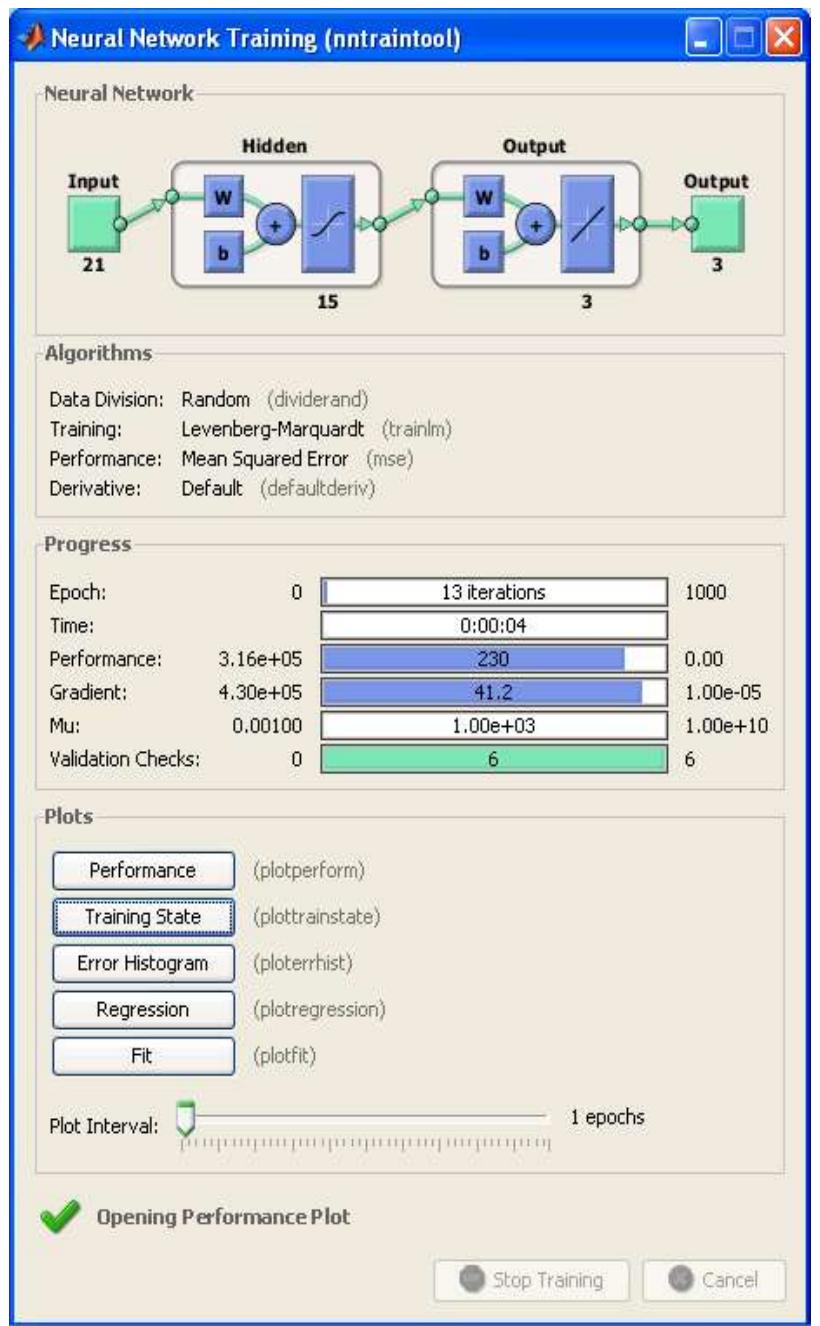
Dáta pre modelovanie hladiny cholesterolu na základne 21  
meraných parametrov (264 vzoriek):

Vstupy: 21 meraných parametrov od 264 pacientov

Výstupy: 3 hladiny cholesterolu (LDL, VLDL, HDL)

```
[x,t] = cho_dataset; % načítanie dát
net = fitnet(15); % vytvorenie NS
net = train(net,x,t); % trénovanie
view(net) % štruktúra siete
y = net(x); % simulácia výstupu NS
plot([t(1,:)' y(1,:)']) % porovnanie dát 1. výstupu s výstupom NS
title('LDL')
```

# *NS pre Cholesterol*



## *Príklad: Dáta pre Spal'ovací motor*

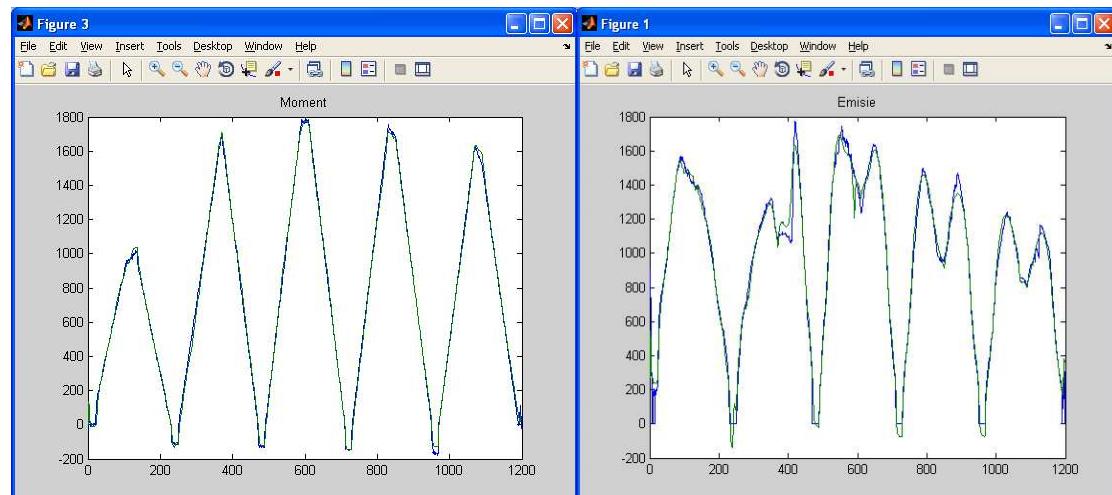
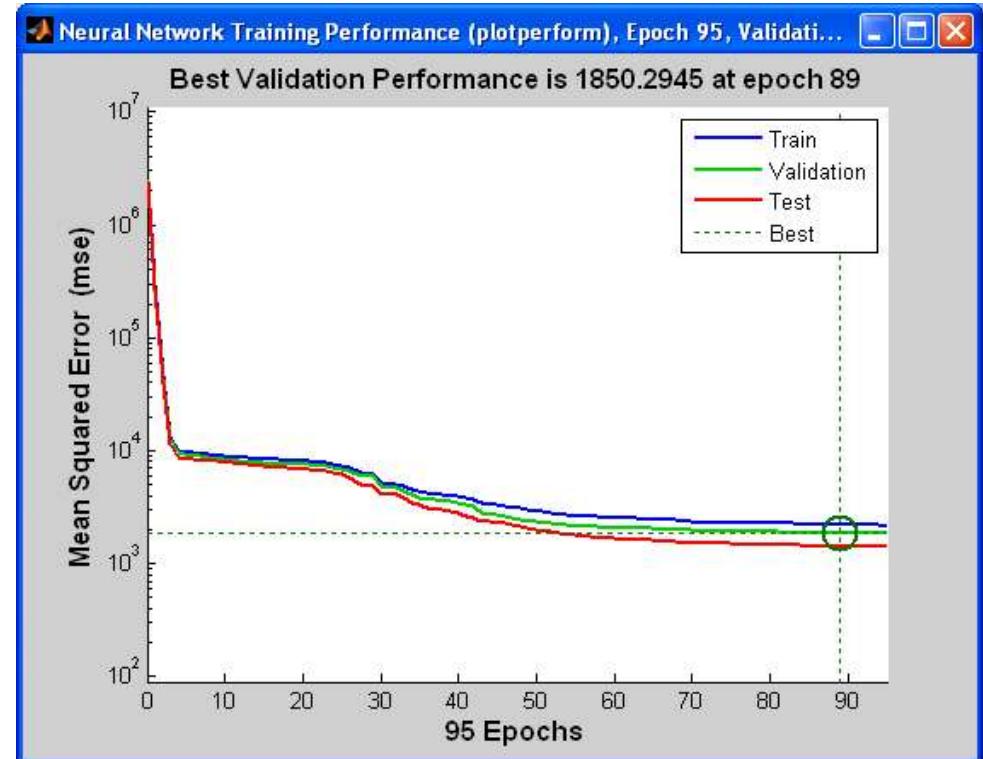
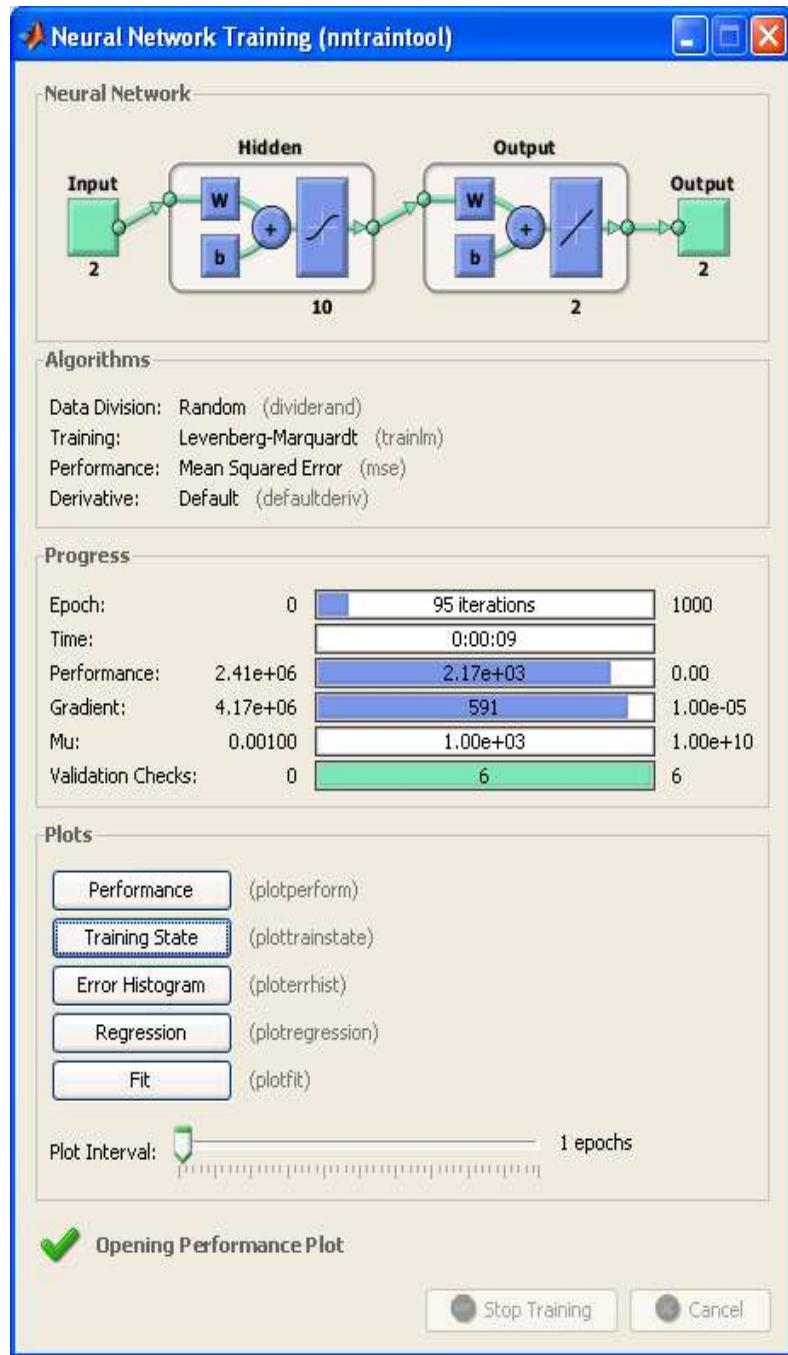
Dáta pre modelovanie krútiaceho momentu a množstva emisií motora na základne spotreby paliva a rýchlosťi (1199 vzoriek):

Vstupy: spotreba paliva, rýchlosť (otáčky)

Výstupy: krútiaci moment, množstvo emisií

```
[x,t] = engine_dataset;% načítanie dát  
net = fitnet(10); % vytvorenie NS  
net = train(net,x,t); % trénovanie  
view(net) % štruktúra siete  
y = net(x); % simulácia výstupu NS  
plot([t(1,:)' y(1,:)']) % porovnanie dát 1. výstupu s výstupom NS  
title('Moment')  
plot([t(2,:)' y(2,:)']) % porovnanie dát 2. výstupu s výstupom NS  
title('Emisie')
```

# *NS pre Spal'ovací motor*



# Realizácie neurónových sietí v Pythone

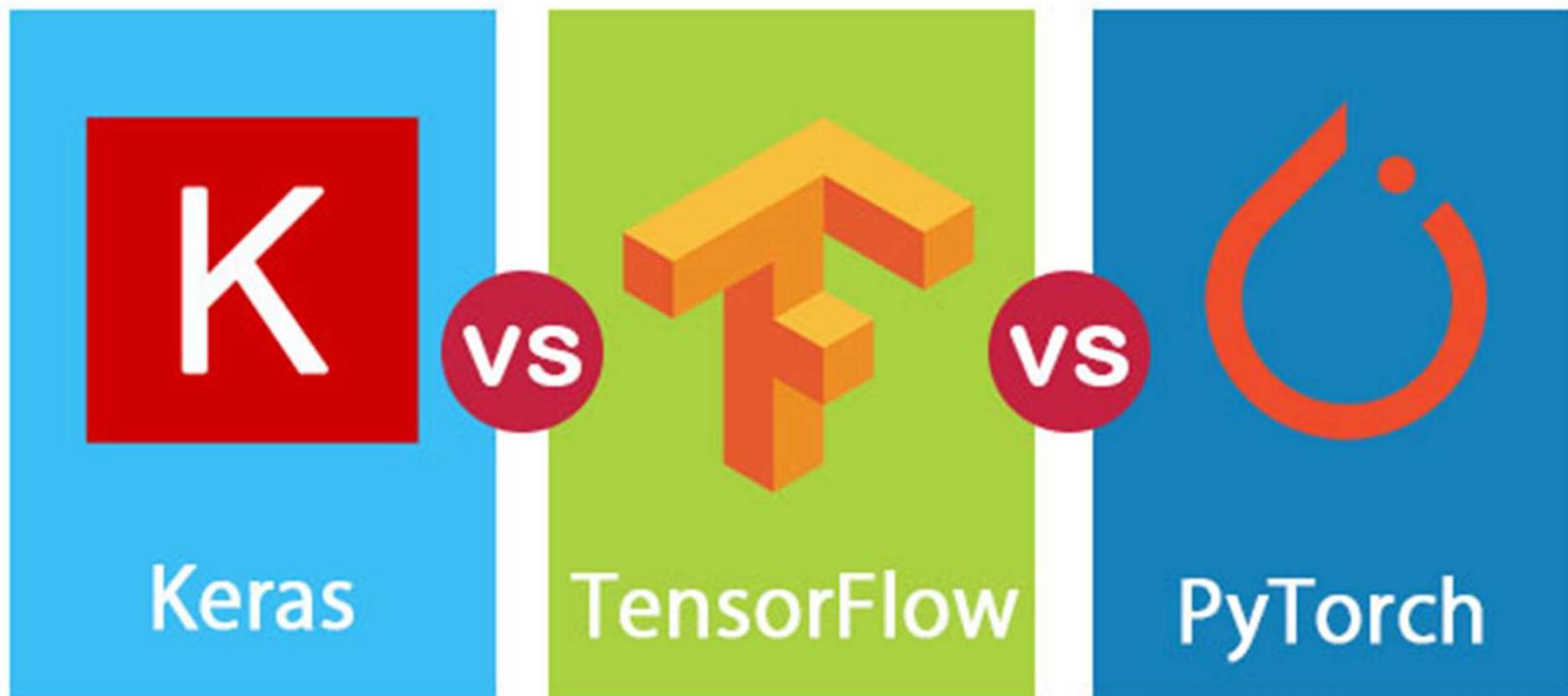
Python ML – Vývojové prostredia

1. IDLE – oficiálny editor, jednoduchý
2. PyCharm – profesionálny editor pre Python
3. Jupyter notebook – webový nástroj, prehľadný
4. VS Code, Spyder a ďalšie...



# Realizácie neurónových sietí v Pythone

Python ML – Knižnice



Free & Open source  
Podpora pre CPU a GPU

- <https://keras.io/>
- <https://www.tensorflow.org/>
- <https://pytorch.org/>

# Ďalšie dôležité balíky pre strojové učenie a dátovej analýzy

**NumPy** – Multi-dimenzionálne polia a matice

**SciPy** – Vedecké a technické výpočty s poliami NumPy (manipulácia s obrazmi, lineárna algebra, štatistika, ...)

**Matplotlib** – Zobrazovanie grafov

**TensorBoard** – Vizualizačný nástroj pre experimenty strojového učenia

**Pandas** – Práca s komplexnými dátami (JSON, SQL, Microsoft Excel, CSV, ...)

**Pillow / PIL (Python Imaging Library)** – Práca s obrazmi

**OpenCV** – Počítačové videnie v reálnom čase



# Realizácia - Keras

## Načítanie knižníc

Dokumentácia

[https://www.tensorflow.org/api\\_docs/python/tf/keras](https://www.tensorflow.org/api_docs/python/tf/keras)

<https://keras.io/api/>

```
import tensorflow as tf
```

```
from tensorflow.keras import layers, models, activations
```

## Načítanie dát a príprava do formátu pre MLP

- načítanie dát, normalizácia, zmena formátu vstupný a výstupných dát (napr. obrazové 2D data do 1D, výstupné data do typu categorical)
- rozdelenie dát na trénovacie a testovacie

```
x_train_mlp, x_test_mlp, y_train_mlp, y_test_mlp =  
train_test_split(x, y, test_size=0.4, shuffle = True)
```

x-vstupné data, y-výstupné data, test\_size-koľko percent sú testovacie, shuffle – premiešanie dát

# Realizácia - Keras

## Vytvorenie MLP

Vytvoríme sekvenciu vrstiev a pridáme do nej plne-prepojenú vrstvu.

# Vytvorenie modelu

```
model = Sequential()
```

# Vstupná vrstva, ak treba meniť rozmer napr. 2D na 1D

```
model_mlp.add(layers.Flatten(input_shape=(IMG_HEIGHT, IMG_WIDTH, 1)))
```

# Prvá skrytá vrstva (pocet\_neuronov1, aktivačná funkcia ReLU)

```
model.add(Dense(pocet_neuronov1, activation='relu', input_shape=(input_size,)))
```

# Druhá skrytá vrstva (pocet\_neuronov2, aktivačná funkcia ReLU)

```
model.add(Dense(pocet_neuronov2, activation='relu'))
```

# Dropout vrstva, vypínanie náhodných prepojení siete, rate – pravdepodobnosť vypínania

```
model_mlp.add(layers.Dropout(rate=0.5))
```

# Výstupná vrstva (pocet\_tried, aktivačná funkcia Softmax pre klasifikáciu)

```
model.add(Dense(pocet_tried, activation='softmax'))
```

# Realizácia - Keras

## Kompilácia, trénovanie a testovanie MLP modelu

Priamo v compile zadefinujeme optimalizátor (Adam), chybovú funkciu (krížová entrópia) a metriku klasifikácie (úspešnosť) - použijú sa defaultné nastavenia

```
# Kompilácia modelu
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Trénovanie modelu
```

```
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test))
```

```
# Vyhodnotenie modelu na testovacích dátach
```

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
print(f'Test accuracy: {test_acc}')
```

```
# Predikcia na nových dátach
```

```
Y_pred = model.predict(x_test)
```

# Realizácia - Pytorch

## Načítanie knižníc

Dokumentácia

<https://pytorch.org/docs/stable/index.html>

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

## Načítanie dát a príprava do formátu pre MLP

```
# Definícia transformácie pre predspracovanie (normalizácia)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)) # Normalizácia na rozsah -1 až 1
])
```

```
# Dátové loadery na trénovanie a testovanie
trainloader = DataLoader(trainset, batch_size=32, shuffle=True)
testloader = DataLoader(testset, batch_size=32, shuffle=False)
```

# Realizácia - Pytorch

## Vytvorenie MLP

Vytvoríme triedu (class) pre MLP siet', s metódami pre inicializáciu štruktúry a pre dopredné šírenie signálu.

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        # Prvá skrytá vrstva
        self.fc1 = nn.Linear(input_size, pocet_neuronov1)
        # Druhá skrytá vrstva
        self.fc2 = nn.Linear(pocet_neuronov1, pocet_neuronov2)
        # Výstupná vrstva (počet tried)
        self.fc3 = nn.Linear(pocet_neuronov2, pocet_tried)

    def forward(self, x):
        # Vstup: x má tvar (batch_size, 1, 28, 28)
        x = x.view(-1, input_size) # Preformátovanie na tvar (batch_size)

        # Propagácia cez skryté vrstvy s ReLU aktiváciou
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))

        # Výstupná vrstva s log-softmax aktiváciou
        x = self.fc3(x)
        return x
```

# Realizácia - Pytorch

## Vytvorenie MLP

Vytvoríme MLP siet' pomocou vytvorenej triedy MLP. Zadefinujeme optimalizátor (Adam), krok učenia lr, chybovú funkciu (krížová entrópia- použijú sa defaultné nastavenia

```
# Vytvorenie modelu  
model = MLP()
```

```
# Chybová funkcia (krížová entropia)  
criterion = nn.CrossEntropyLoss()
```

```
# Optimalizátor (Adam)  
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

# Realizácia - Pytorch

## Trénovanie MLP modelu

```
num_epochs = 5
```

```
for epoch in range(num_epochs):
    model.train() # Nastavenie modelu do tréningového režimu
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in trainloader:
        optimizer.zero_grad() # Nastavenie gradientov na nulu
        outputs = model(inputs) # Propagácia dopredu
        loss = criterion(outputs, labels) # Výpočet straty
        loss.backward() # Spätná propagácia
        optimizer.step() # Aktualizácia váh
        running_loss += loss.item()
        # Výpočet presnosti
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    # Zobrazenie informácií o tréningu
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(trainloader):.4f},
Accuracy: {100 * correct / total:.2f}%")
```

# Realizácia - Pytorch

## Testovanie MLP modelu

```
model.eval() # Nastavenie modelu do hodnotiaceho režimu  
correct = 0  
total = 0  
  
with torch.no_grad(): # Zabráňujeme počítaniu gradientov počas vyhodnocovania  
    for inputs, labels in testloader:  
        outputs = model(inputs)  
        _, predicted = torch.max(outputs.data, 1)  
        total += labels.size(0)  
        correct += (predicted == labels).sum().item()  
  
    print(f'Test Accuracy: {100 * correct / total:.2f}%')
```

## Kontingečná tabuľka

```
cm = confusion_matrix(y_true, y_pred)  
  
disp = ConfusionMatrixDisplay(confusion_matrix=cm)  
disp.plot()  
plt.show()
```