INTELLIGENTNI SISTEMI

NEVRONSKE MREŽE IN KLASIFIKACIJA

Nevronske mreže

Prof. Jurij F. Tasič
Emil Plesnik

Uvod

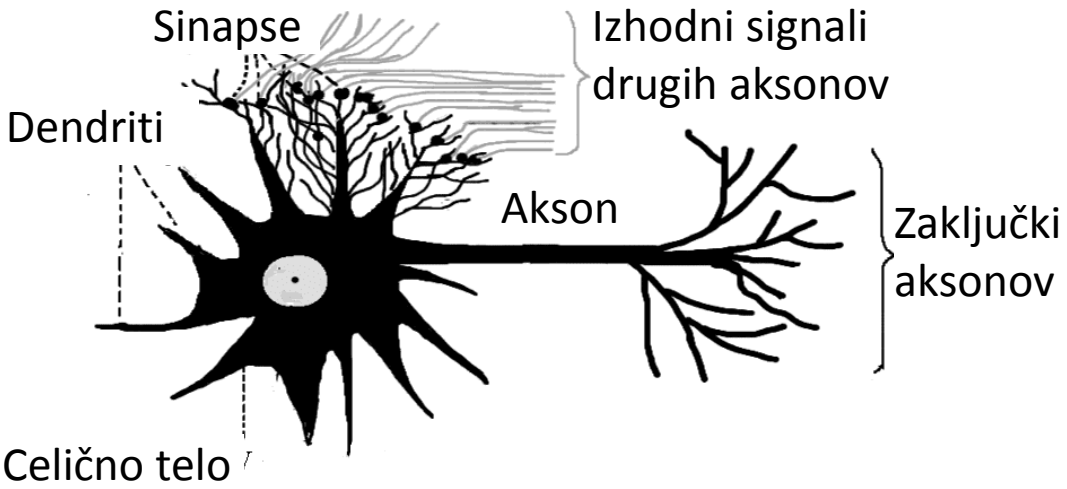
- Umetne nevronske mreže – ang. *Artificial Neural Networks* (ANN)
  - Preračunavanje povezav
  - Vzporedno porazdeljeno procesiranje
  - Računski modeli
- Biološki računski modeli
- Strojno učenje
- Umetna inteligenca

Zgodovina

- McCulloch in Pitts predstavita perceptron leta 1943.
  - Poenostavljen model biološkega nevrona
- Pade v pozabo v poznih 60-ih
  - (Minsky in Papert)
  - Omejitve perceptrona
- Preporod v sredini 80-ih
  - Nelinearne nevronske funkcije
  - Vzvratno učenje oz. “učenje nazaj”

Biološki nevron

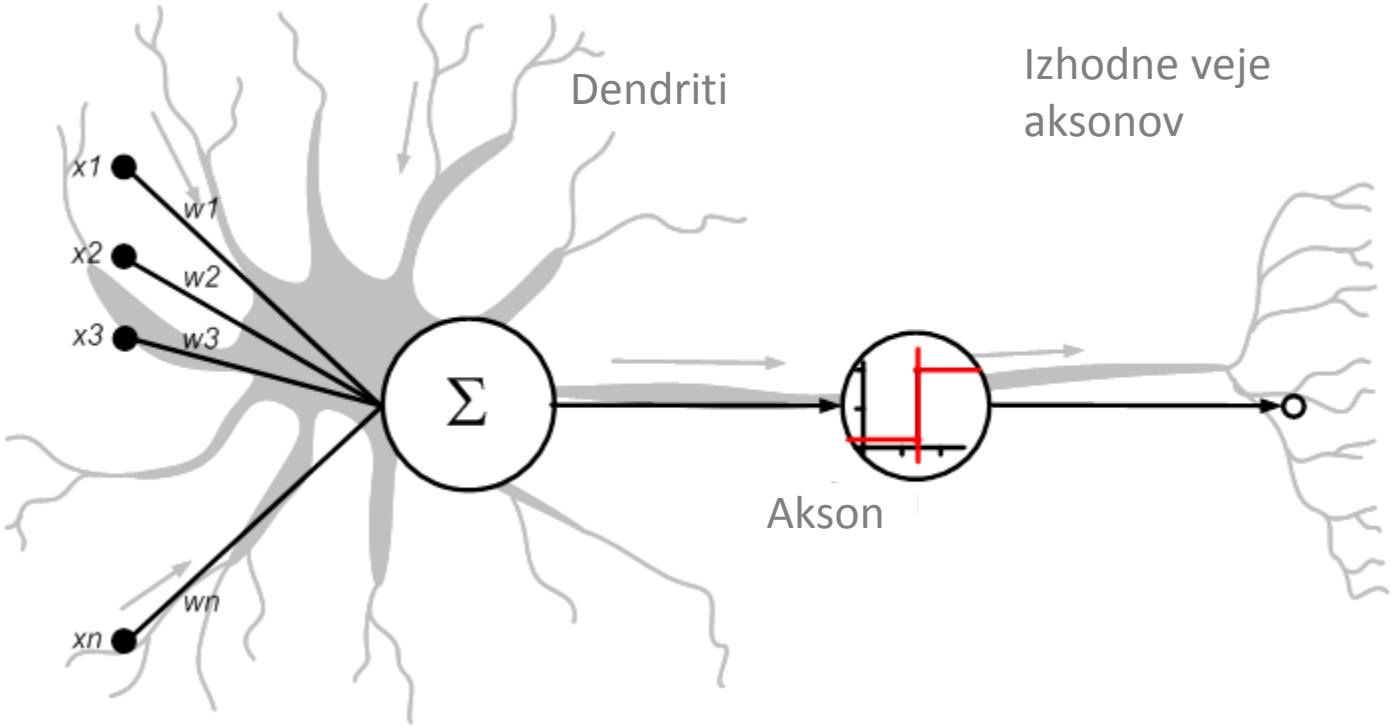
Človeški živčni sistem je zelo kompleksna nevronska mreža. Možgani predstavljajo osrednji element tega sistema, ki ga sestavljaj skoraj 10^{10} bioloških nevronov, ki so med sabo povezani s podomrežji.



Celično telo nevrona sprejema vhodne signale iz dendritov in številnih sinaps na njenem površju. Nevron odda impulz na svoj akson, če je prejel dovolj vhodnih signalov za njegovo stimulacijo do pragovne vrednosti. Če pragovna vrednost z vhodnimi signali ni dosežena, nevron ne odda impulza.

Perceptron

- Binarne klasifikacijske funkcije
- Pragovna aktivacijska funkcija



Klasifikator nevronske mreže

- Vhod: Podatki za klasifikacijo

  Vsebujejo klasifikacijske atribute

- Podatki se razdelijo kot pri vsakem klasifikacijskem problemu.

  [Učna množica in testna množica]

- Podatki morajo biti normalizirani.

(t.j. vse vrednosti atributov (značilk) v podatkovni bazi se prilagodijo na interval [0,1] ali [-1,1])

Nevronska mreža lahko dela s podatki v obsegu (0,1) ali (-1,1).

- Ločimo dva osnovna normalizacijska postopka

  [1] Max-Min normalizacija

  [2] Normalizacija z decimalnim skaliranjem

Nevronska mreža

- Nevronska mreža se uči s prilagajanjem uteži tako, da je sposobna pravilno klasificirati podatke učne množice in nato po testni fazi tudi neznane podatke.

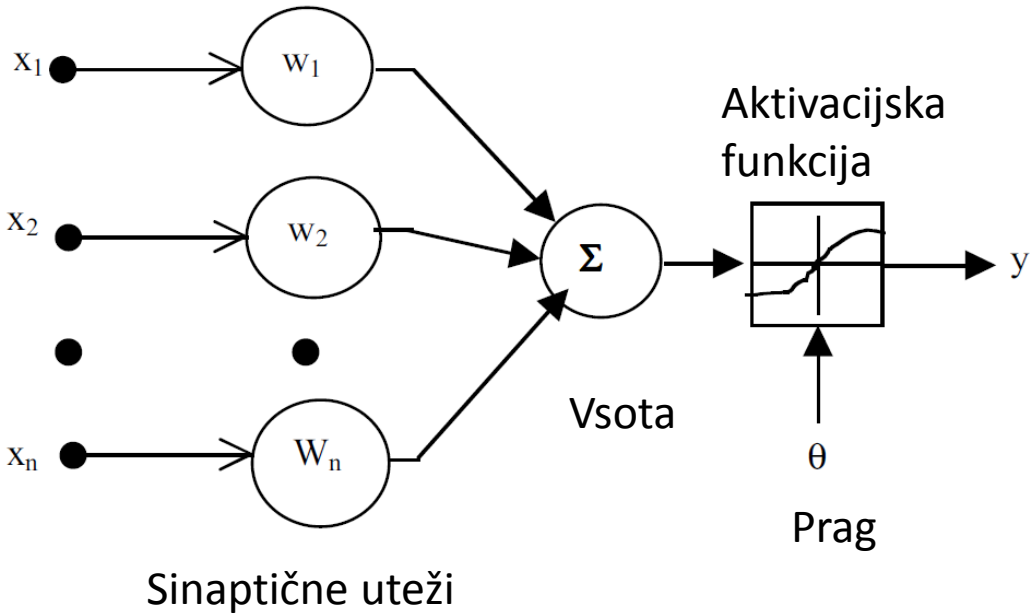
- Nevronska mreža potrebuje dalj časa za učenje.

- Nevronska mreža ima visoko toleranco za zašumljene in nepopolne podatke.

Kaj potrebujemo za uporabo NM?

- Določitev ustreznih vhodov
- Bazo podatkov za učno in testno fazo nevronske mreže.
- Določitev optimalnega števila skritih vozlišč/nevronov.
- Oceno parametrov (učenje).
- Oceno zmogljivosti mreže.
- Če zmogljivost ni zadovoljiva, potem ponovimo predhodne točke.

Osnovni elementi nevronskih mrež

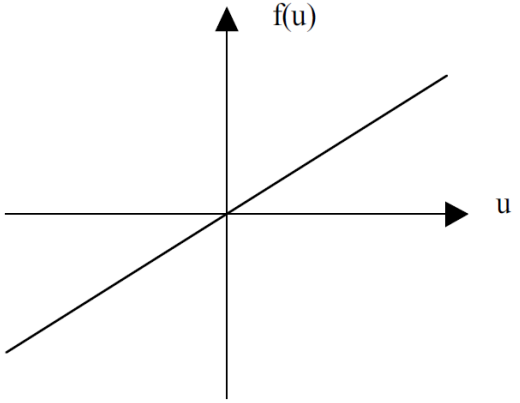


Sinaptične uteži: Vrednosti W1 ,W2 ,W3 ,…..,Wn so uteži povezane z vsakim nevronom za določitev moči vhodnega vektorja $X = [x1\ x2\ x3\ldots..,xn]^T$

Prag: Notranji prag nevrona θ predstavlja vrednost, pri kateri se aktivira izhod nevrona y

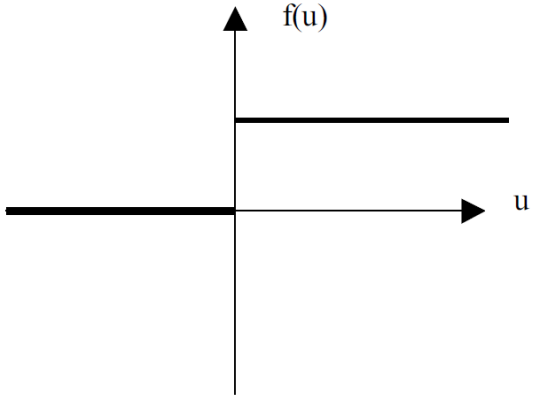
Aktivacijska funkcija: Aktivacijska funkcija izvaja določeno matematično operacijo na izhodnem signalu. Za uporabo so možne tudi zahtevnejše aktivacijske funkcije, Odvisno od vrste problema, ki ga rešuje mreža.

Najpogostejše aktivacijske funkcije



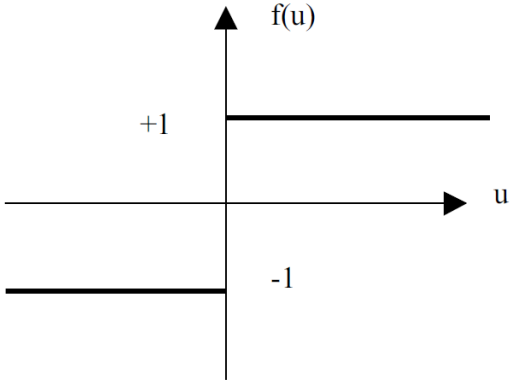
Linearna aktivacijska funkcija

$y = f(u) = a.u$



Binarna aktivacijska funkcija

$$y = f(u) \begin{cases} 0 & if\ u < 0 \\ 1 & if\ u \geq 0 \end{cases}$$

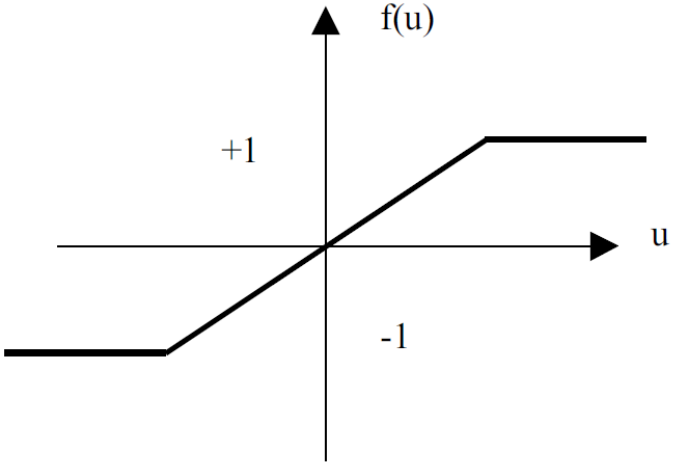


Bipolarna aktivacijska funkcija

Najpogostejše aktivacijske funkcije

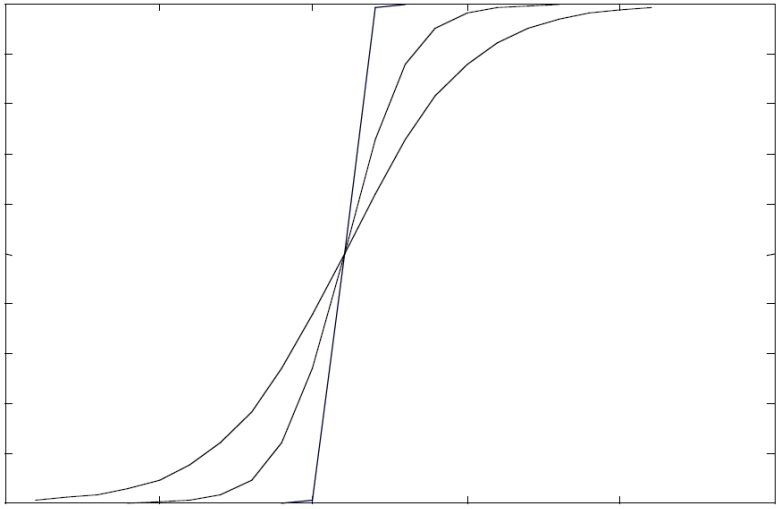
$$y = f(u)\begin{cases} -1 & if\ u < -1 \\ u & if\ -1 \geq u \geq 1 \\ 1 & if\ u \geq 0 \end{cases}$$

Odsekoma linearna aktivacijska funkcija



$$f(x) = \frac{1}{1 + e^{-\alpha x}}\ ,\ 0 \leq f(x) \leq 1$$

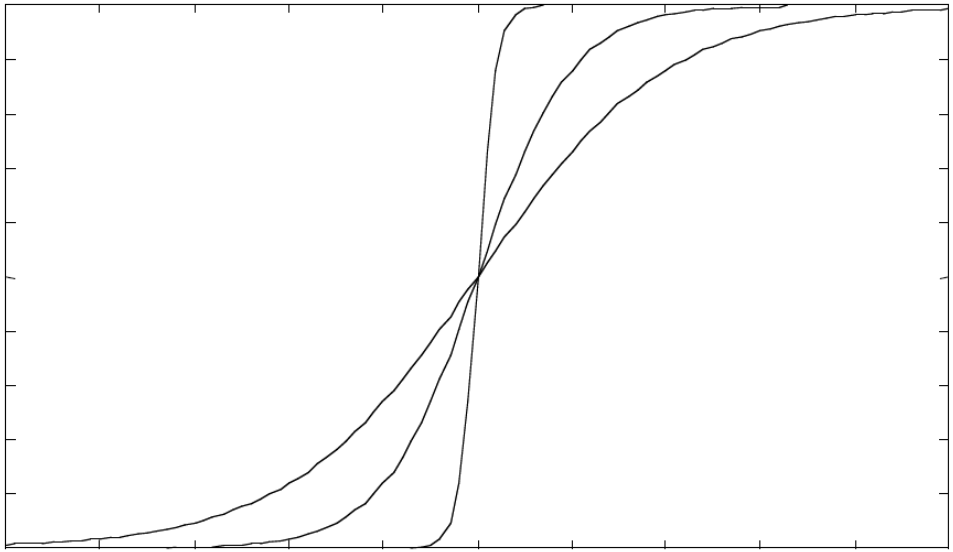
Sigmoidalna (S) aktivacijska funkcija



Najpogostejše aktivacijske funkcije

Tangentna hiperbolična aktivacijska funkcija

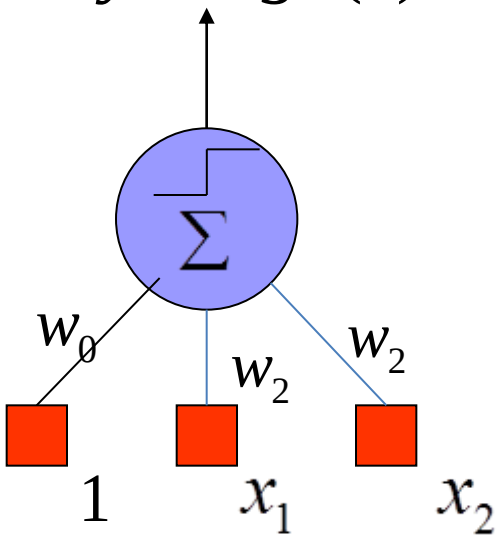
$$f(x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}} \qquad -1 \le f(x) \le 1$$



Perceptron

- Rosenblatt (1962)
- Linearna separacija
- Vhod: vektor realnih vrednosti
- Izhod: 1 ali-1

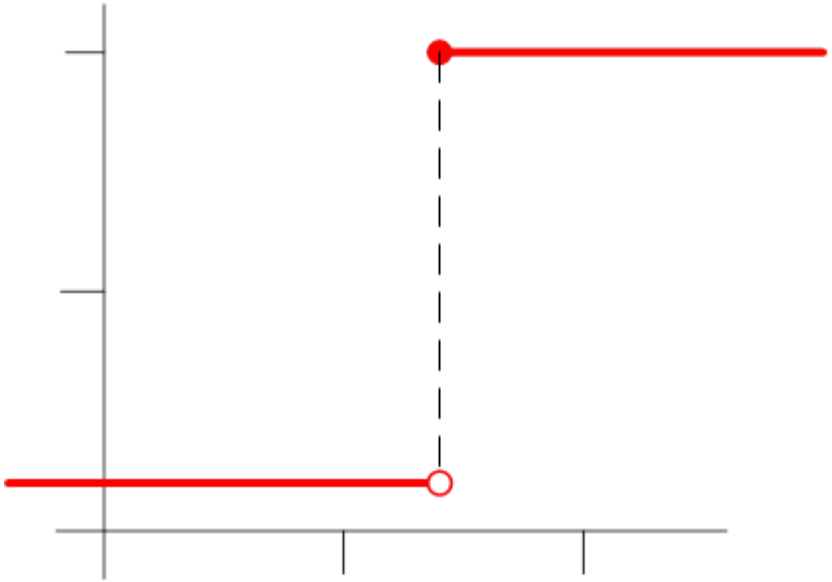
$$y = sign(v)$$



$$v = w_0 + w_1 x_1 + w_2 x_2$$

Perceptron: pragovna aktivacijska funkcija

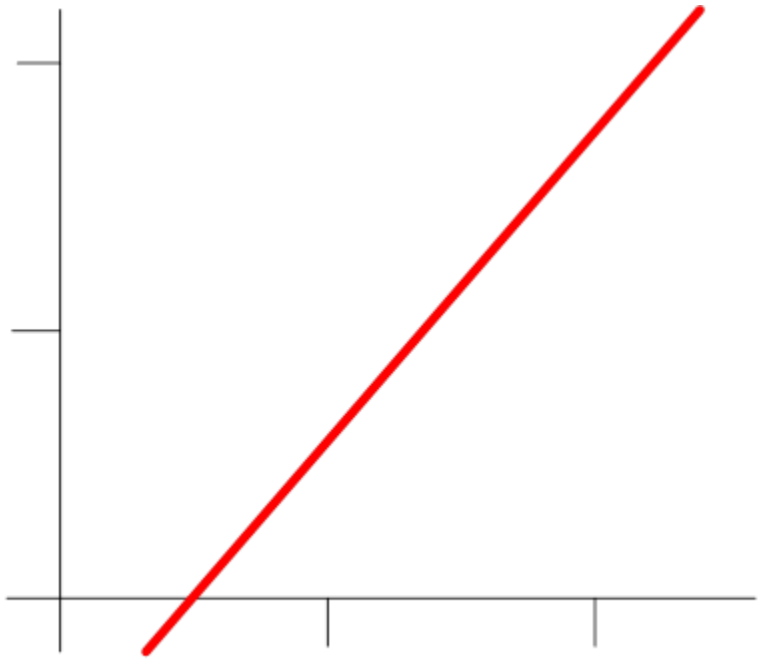
- Binarne klasifikacijske funkcije
- Pragovna aktivacijska funkcija



Stopnica

Linearne aktivacijske funkcije

- Izhod je utežena vsota vhodov.

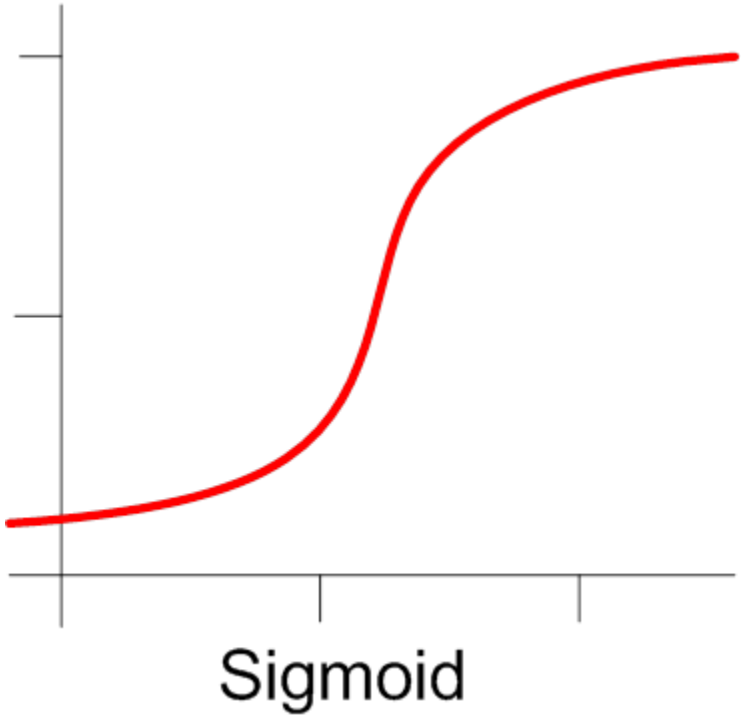


Linearno

$$y = u = \sum_{n=1}^{N} w_n x_n$$

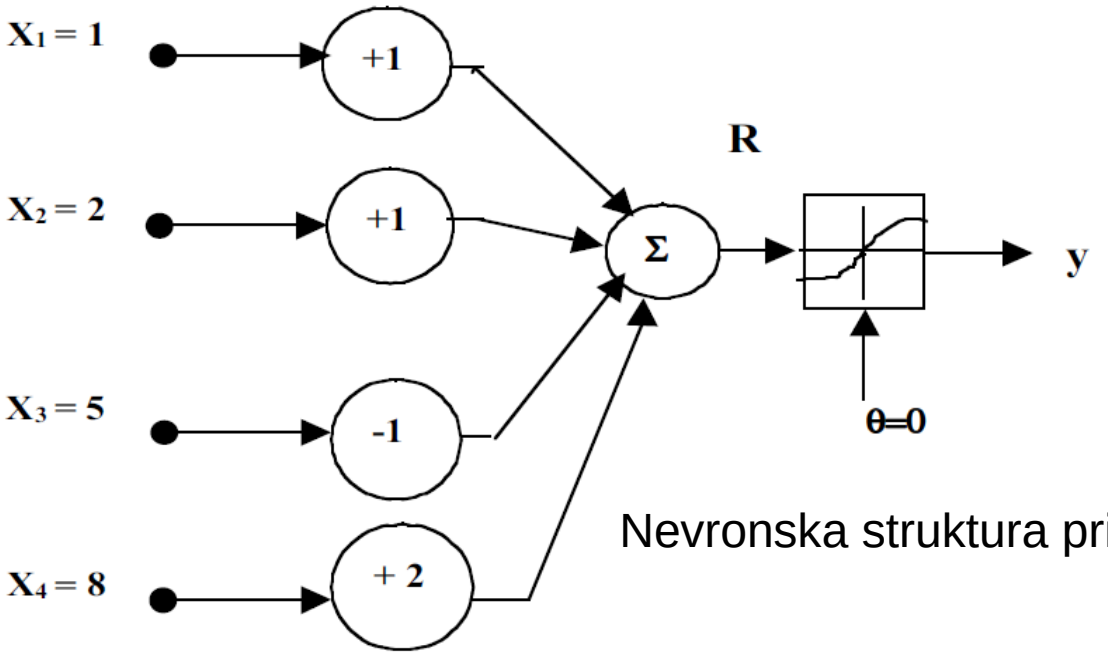
Nelinearne aktivacijske funkcije

Sigmoidalna funkcija



$$y_{hid}(u) = \frac{1}{1+e^{-u}}$$

Primer



Nevronska struktura primera

Izhodna vrednost R je določena po sledeči enačbi:

$$R = W^T . X = \begin{bmatrix} 1 & 1 & -1 & 2 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 5 \\ 8 \end{bmatrix} = 14$$

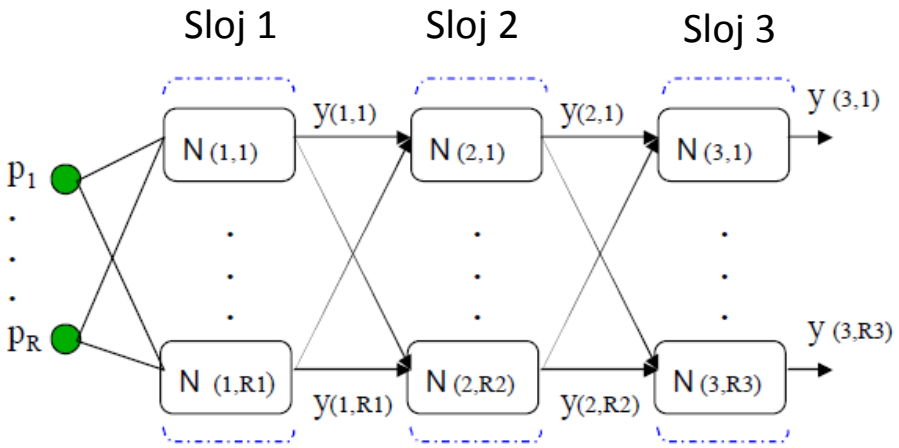
Za binarno in sigmoidalno aktivacijsko funkcijo sta izhoda nevrona:

$$\mathbf{y}(Threshold) = 1;$$

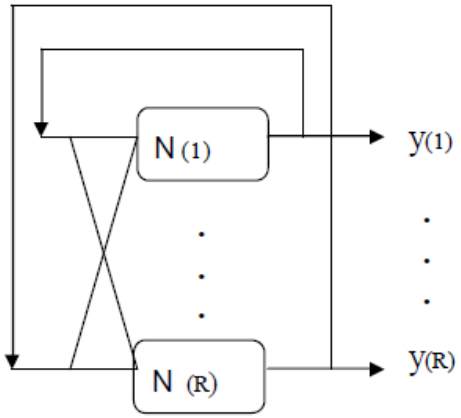
$$\mathbf{y}(Sigmoid) = 1.5*2^{-8}$$

Ponavljajoča in Feed-forward arhitektura

- Mreže Feed-forward ne morejo ponazarjati časovnih razmerij
- Ponavljajoče mreže vsebujejo interne povratne povezave, ki omogočajo izkazovanje časovnega vedenja



Feed-forward arhitektura s tremi sloji



Ponavljajoča arhitektura (Hopfieldova nevronska mreža)
V mreži so izhodi uporabljeni kot vhodi za vsak naslednji cikel.

Funkcije Feed-Forward omrežij

Nevronska mreža je lahko predstavljena na podoben način kot linearni modeli, vendar so bazne funkcije posplošene

$$y(\mathrm{x},\mathrm{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathrm{x})\right)$$

Aktivacijska funkcija

Za regresijo: funkcija identitete
Za klasifikacijo: nelinearna funkcija, npr. sigmoidalna.

Koeficienti w_j se spreminjajo med učenjem.
Lahko je več aktivacijskih funkcij.

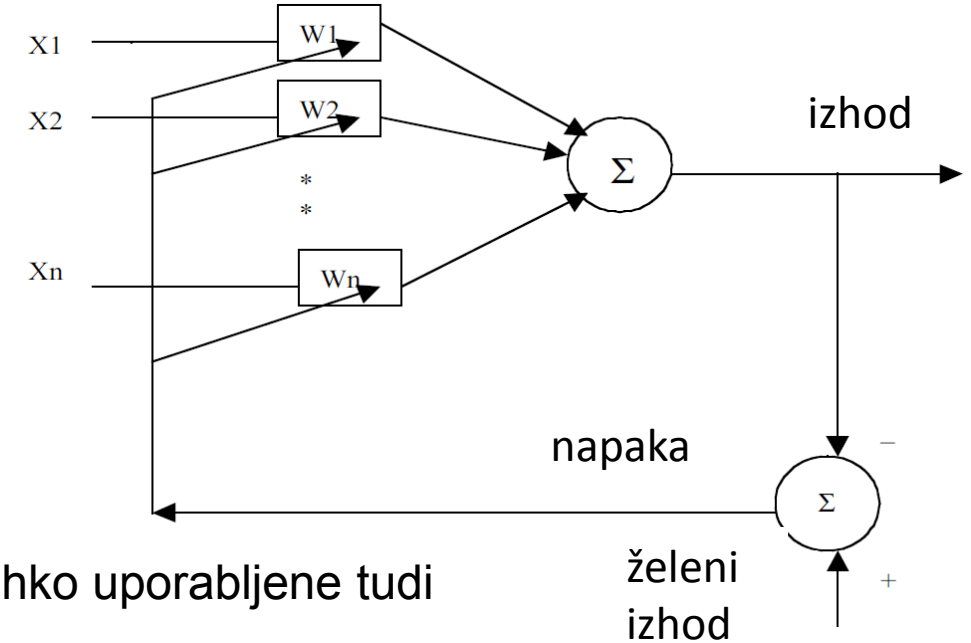
Bazne funkcije

$\Phi_j(x)$ je nelinearna funkcija, npr. tanh linearne kombinacije D vhodov.

Parametri se spreminjajo med učenjem.

ADALINE – učna nevronska mreža

ADAptive LINear Element (ADALINE) je sestavljena iz enega nevrona tipa McCulloch-Pitts, njegove uteži so določene z normalizirano učno metodo najmanjših kvadratov (LMS).



Za učenje nevronske mreže so lahko uporabljene tudi aproksimacije prilagajanja krivulj.

Učna faza ADALINE- vhodni vektor $X \in R^n$: $X = [x_1\ x_2\ x_3 \ldots x_n]^T$ in želeni izhod so podani.

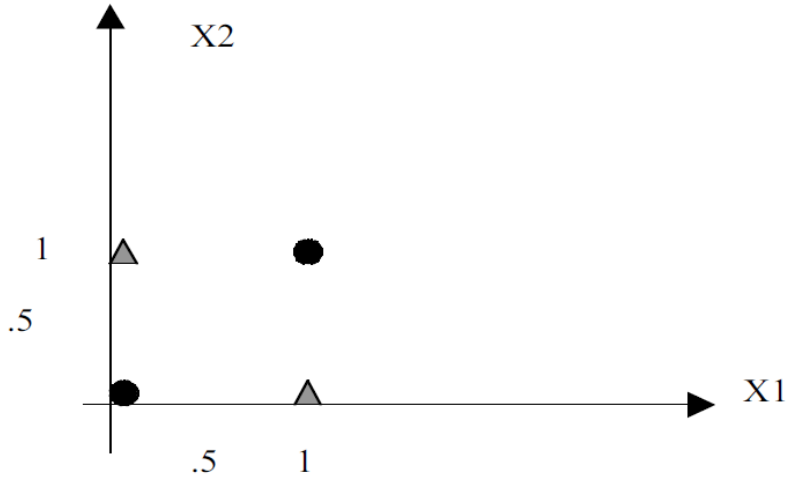
Po končanem učenju ADALINE, je rezultat za poljubni vhodni vektor za mrežo s fiksnimi utežmi skalarna vrednost.

Primer

Klasični primer nedoločljive preslikave je funkcija XOR (exclusive or).

X1	*X2*	*Output*
0	0	0
0	1	1
1	0	1
1	1	0

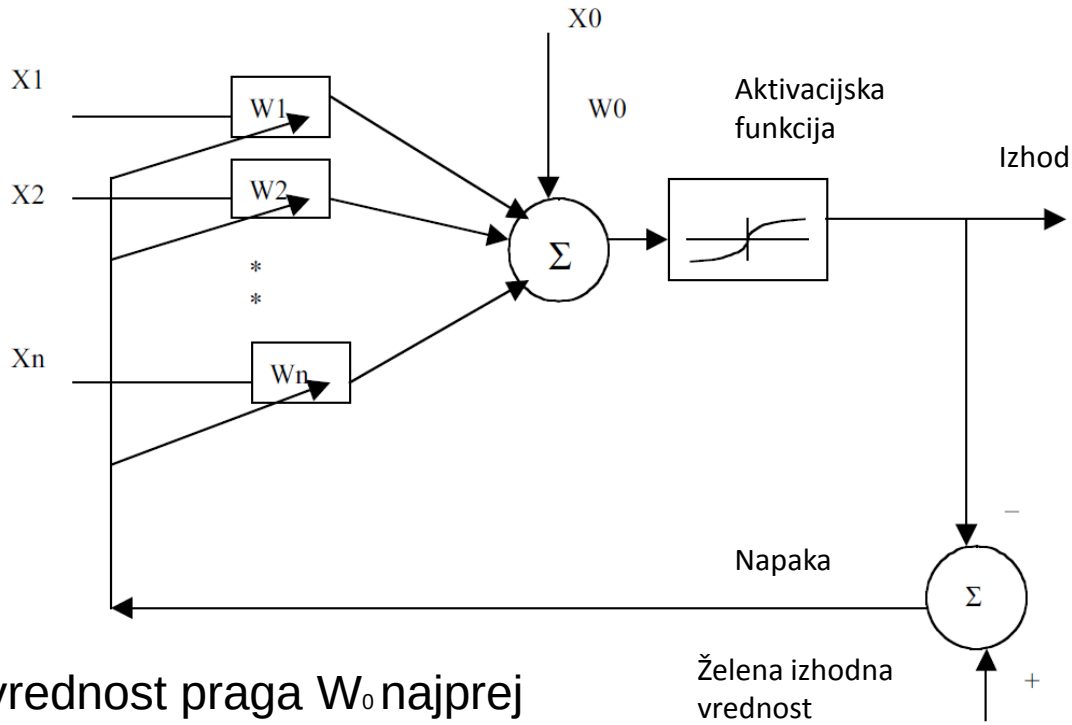
Ravnina izhodov XOR X1- X2



Enonivojski perceptron

Arhitektura: Osnovno idejo perceptrona je razvil Rosenblatt v poznih 1950ih.

Uteži povezav in prag perceptrona so lahko fiksne ali adaptivne z uporabo različnih algoritmov.



Uteži povezav W_1, W_2,…,W_n in vrednost praga W_0 najprej nastavimo na majhne, od nič različne vrednosti. Potem sprejmemo preko senzorskih merilnih naprav novo vhodno vrednost določeno z N vrednostmi in določimo (izračunamo) novo vhodno vrednost. Uteži povezav se prilagodijo samo v primeru napake.

Linearna klasifikacija

Vzemimo 2 vhodna razreda vzorcev C1 in C2.
Prilagajanje uteži pri k-ti učni fazi lahko zapišemo kot:

1. Če je k-ti vzorec učnega vektorja x(k) pravilno razvrščen, korekcija vektorja uteži ni potrebna. Ob uporabi stopničaste aktivacijske funkcije sta veljavni naslednji operaciji:

W (k+1)=W (k) če izhod>0 in x (k)∈C1 , in
W(k+1)=W(k) če izhod <0 in x(k)∈C2.

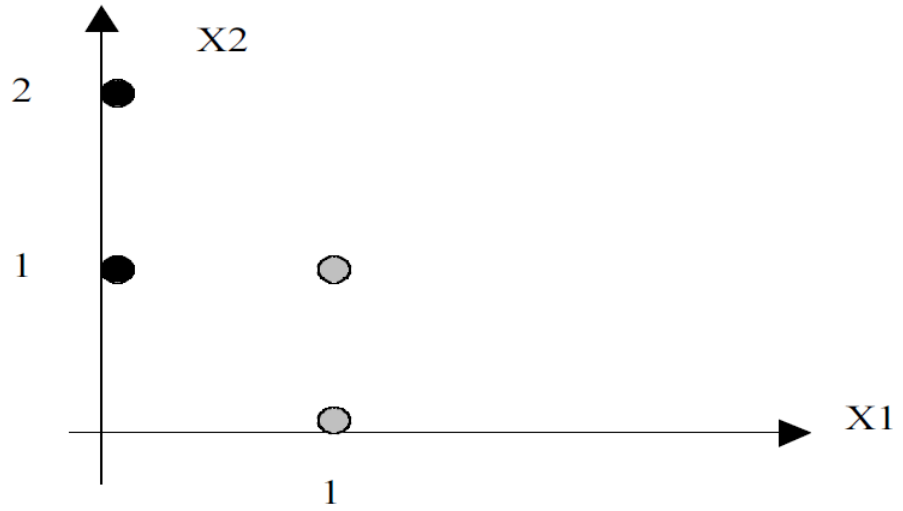
2. V nasprotnem primeru moramo popraviti uteži glede na naslednje pravilo:

*W(k+1)=W(k)+*η *x(k) če izhod ≥0 in x(k)*ε *C1*
*W(k+1)=W(k)-*η *x(k) če izhod ≤0 in x(k)*ε *C2*

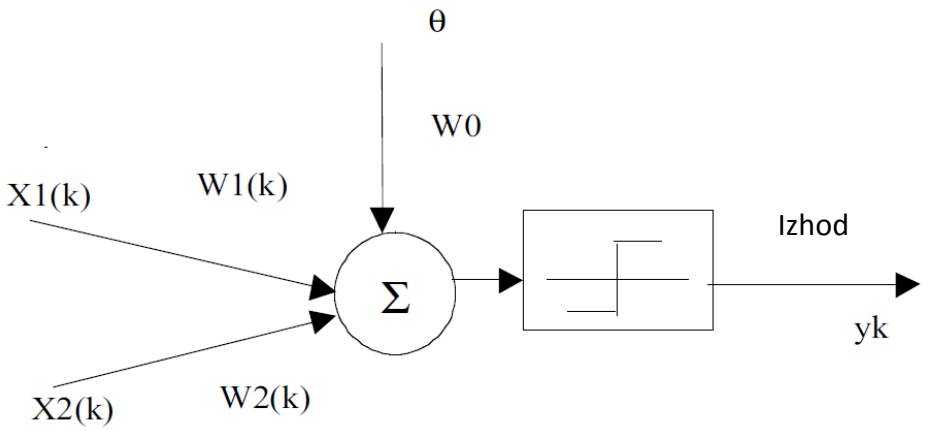
Pri tem je η parameter učne hitrosti, izbran med 0 in 1.

Primer linearne klasifikacije

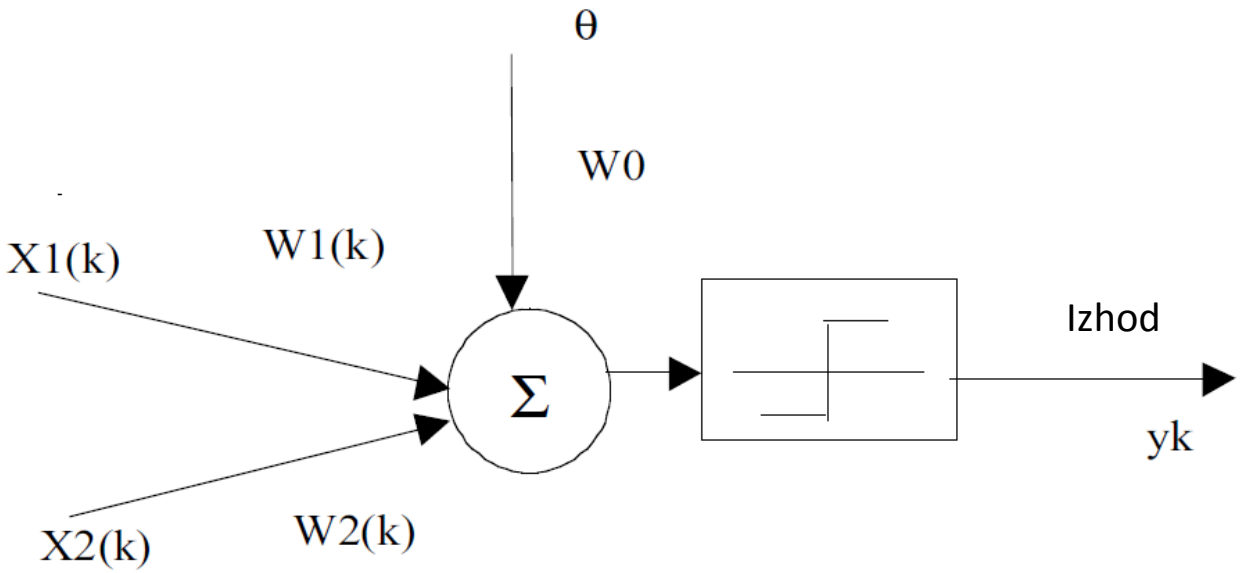
Vzemimo vhodna razreda vzorcev C1 in C2, kjer je C1: {(0,2), (0,1)} in C2: {(1,0), (1,1)}. Cilj je pridobiti odločitveno ravnino na osnovi učenja perceptrona.



Ker vhodni vektor sestavljata dva elementa, je struktura perceptrona sledeča:



Struktura perceptrona v našem primeru



Privzemimo η=1 in začetni vektor uteži $W(1)=[0\ 0]$

Iteracija 1

$$W^T(1)..x(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 0$$

Posodobitev uteži

$$W(2) = W(1) + x(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

Struktura perceptrona v našem primeru

Iteracija 1 $$W^T(1)..x(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 0$$

Posodobitev uteži $$W(2) = W(1) + x(1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

Iteracija 2 $$W^T(2)..x(2) = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 2 > 0$$ $x(k) \in C_1$

Posodobitev uteži $$W(3) = W(2)$$

Iteracija 3 $$W^T(3)..x(3) = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

Posodobitev uteži $$W(4) = W(3) - x(3) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

Iteracija 4 $$W^T(4)..x(4) = \begin{bmatrix} -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1$$ $x(k) \in C_1$

Posodobitev uteži $$W(5) = W(4) - x(4) = \begin{bmatrix} -1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

Struktura perceptrona v našem primeru

$x(k) \in C_1$

Iteracija 5 $W^T(5).x(5) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 2 > 0$: Pravilna klasifikacija

Iteracija 6 $W^T(6).x(6) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 > 0$: Pravilna klasifikacija

$x(k) \in C_2$

Iteracija 7 $W^T(7).x(7) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -2 < 0$: Pravilna klasifikacija

Iteracija 8 $W^T(8).x(8) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -1 < 0$: Pravilna klasifikacija

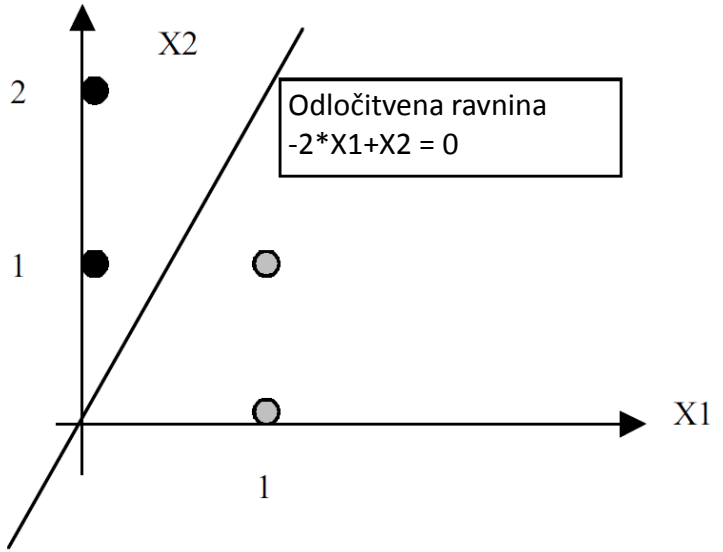
Algoritem konvergira in odločitvena ravnina perceptrona je

$$d(x) = -2X_1 + X_2 = 0$$

TEST

Vzemimo **vhodne podatke {1,2},** ki niso enaki učni množici. Izračunajmo izhod:

$$Y = W^T X = [-2 \;\; 1] \begin{bmatrix} 2 \\ 1 \end{bmatrix} = -3 < 0$$



Izhod Y pripada razredu C2 kot pričakovano.

Učni algoritem perceptrona

Učni algoritem (pravilo delta) perceptrona lahko povzamemo kot:

Korak 1: Inicializacija uteži W_1, W_2…W_n in praga (majhne naključne vrednosti).
Korak 2: Vpeljemo nove vhodne vrednosti X1, X2,..Xn in želeni izhod d_k.
Korak 3: Izračunamo dejanski izhod s sledečo enačbo:

$$y_k = f_h \left(\sum_{i=1}^{n} (XiWi) - \theta_k \right)$$

Korak 4: Popravimo uteži glede na enačbo:

$$W_i(new) = W_i(old) + \eta(d_k - y_k)\, x_i\,, \quad 0 \leq i \leq N$$

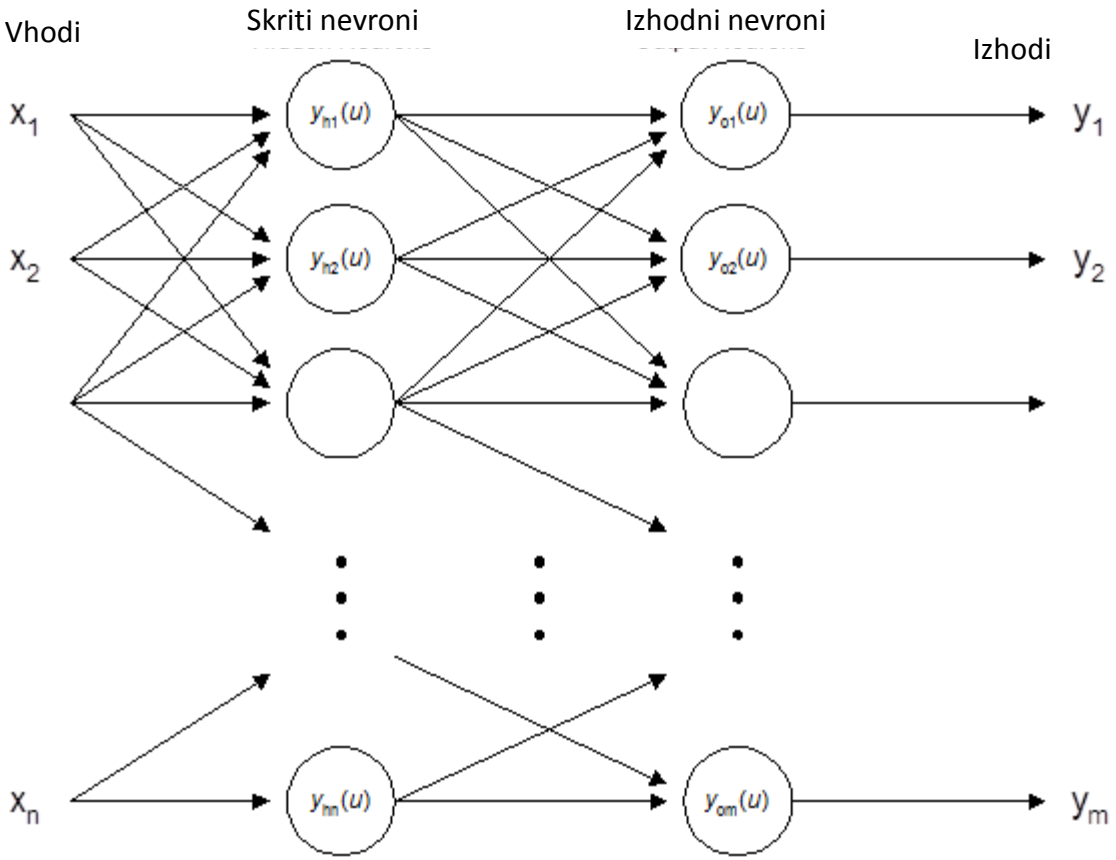
Kjer je η pozitivno število manjše od 1 in je d_k želeni izhod.

Korak 5: Ponovimo korake 2 – 4 dokler ni klasifikacija končana.

Večslojni perceptron

Splošna arhitektura

Večslojni perceptron predstavlja posplošitev enoslojnega.



Večslojni perceptron

Vhodni sloj: Sloj nevronov, ki prejema podatke od zunanjih virov in jih predaja naprej v mrežo v obdelavo.

Skriti sloj: Sloj nevronov, ki prejema podatke, ki prejema podatke od vhodnega sloja in jih obdela v ozadju. Ta sloj nima direktne povezave z zunanjim svetom. Vse povezave med skritim slojem in ostalimi sloji so skrite v notranjosti sistema.

Izhodni sloj: Sloj nevronov, ki prejema obdelane podatke in oddaja izhodne signale sistema.

Pristranskost, začetna vrednost (bias): Deluje na nevron podobno kot "offset". Namen pristranskosti je zagotavljanje praga za aktivacijo nevronov. Vhod pristranskosti je povezan z vsemi skritimi in izhodnimi nevroni mreže.

Preslikava vhod-izhod

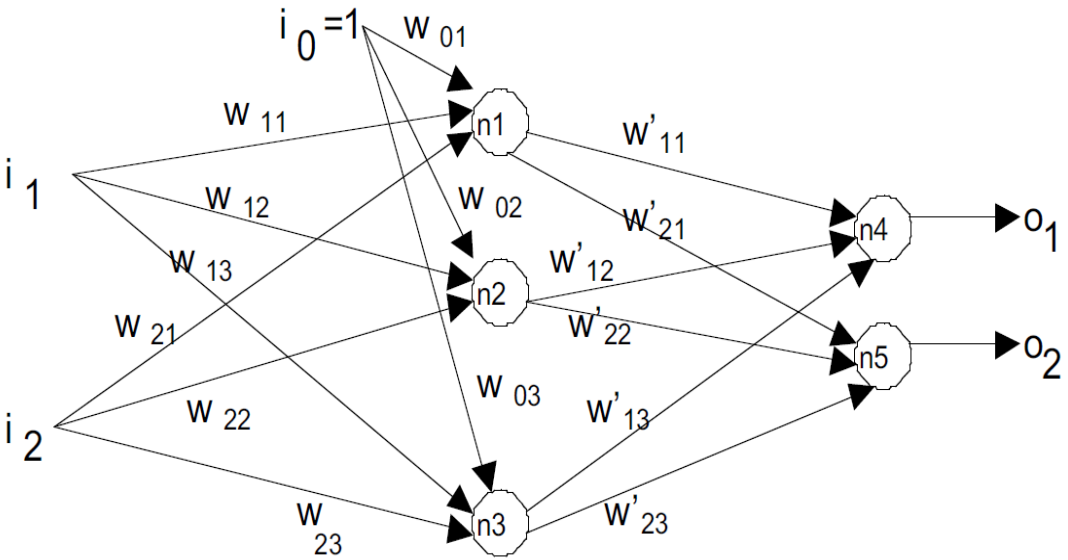
Preslikava vhod-izhod mreže je izvedena glede na uteži in aktivacijske funkcije nevronov vseh slojev.

Vzemimo naslednjo dvonivojsko feed-forward mrežo s tremi nevroni v skritem sloju in dvema nevronoma drugega nivoja:

Za vsak sloj velja linearnost:

$A_1 = W_1 X$

Kjer je A_1 stolpčni vektor dolžine m, W_1 je $m \times n$ matrika uteži, X pa je vhodni stolpčni vektor dolžine n



$$\begin{cases} a_{11} = w_{11} i_1 + w_{21} i_2 \\ a_{12} = w_{12} i_1 + w_{22} i_2 \\ a_{13} = w_{13} i_1 + w_{23} i_2 \end{cases}$$

$$O_1 = F \,.A_1$$

Izhod

Izhodni vektor skritega sloja lahko izračunamo po sledeči enačbi:

$$O_1 = F \,.A_1$$

Pri tem je A1 definiran z enačbo na prejšnji strani, O_1 pa je izhodni stolpčni vektor skritega nivoja dolžine m. F je diagonalna matrika, ki predstavlja nelinearne aktivacijske funkcije prvega skritega nivoja:

$$F = \begin{bmatrix} f_1(.) & 0 & 0 & \dots & 0 \\ 0 & f_2(.) & & & 0 \\ . & & .. & & .. \\ . & & & .. & 0 \\ 0 & 0 & \dots & 0 & f_m(.) \end{bmatrix}$$

Na primer, če so vse aktivacijske funkcije nevronov skritega nivoja izbrane podobno, potem lahko izhode nevronov n_1 do n_3 izračunamo kot:

$$\begin{cases} O_{11} = f(a_{11}) \\ O_{12} = f(a_{12}) \\ O_{13} = f(a_{13}) \end{cases}$$

Izhod skritega sloja

Na podoben način lahko določimo izhod drugih skritih slojev. Izhod mreže z enim samim skritim slojem je glede na prejšnje enačbe določen kot:

$$A_2 = W_2 . O_1$$
$$O_2 = G . A_2$$

Pri tem je A2 vektor aktivnostnih nivojev izhodnega sloja, O2 pa je izhod mreže. G je diagonalna matrika sestavljena iz nelinearnih aktivacijskih funkcij izhodnega sloja:

$$G = \begin{bmatrix} g_1(.) & 0 & 0 & \dots & 0 \\ 0 & g_2(.) & & & 0 \\ . & & .. & & .. \\ . & & & .. & 0 \\ 0 & 0 & \dots & 0 & g_q(.) \end{bmatrix}$$

Aktivnostni nivo izhodnih nevronov n_4 in n_5 lahko izračunamo kot:

$$\begin{cases} a_{21} = W'_{11} O_{11} + W'_{12} O_{21} + W'_{13} O_{31} \\ a_{22} = W'_{21} O_{11} + W'_{22} O_{21} + W'_{23} O_{31} \end{cases}$$

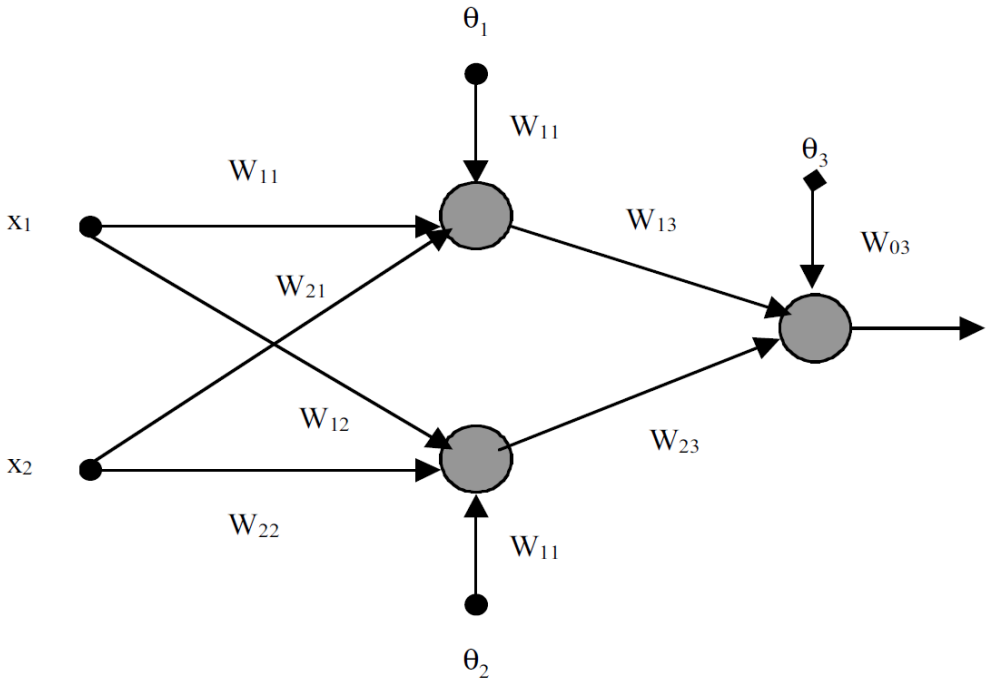
Oba izhoda mreže s podobnimi aktivacijskimi funkcijami lahko izračunamo kot:

$$\begin{cases} O_1 = g(a_{21}) \\ O_2 = g(a_{22}) \end{cases}$$

XOR realizacija

Pokazali smo, da enonivojski perceptron ne zmore klasificirati vhodnih vzorcev, ki niso linearno različni, kot npr. XOR funkcija. Ta primer lahko obravnavamo kot posebni primer bolj splošnega nelinearnega problema preslikave. V primeru XOR funkcije moramo upoštevati vsa štiri oglišča enotinega kvadrata, ki ustrezajo vhodnemu vzorcu. Problem lahko rešimo z večslojnim perceptronom z enim skritim nivojem (slika).

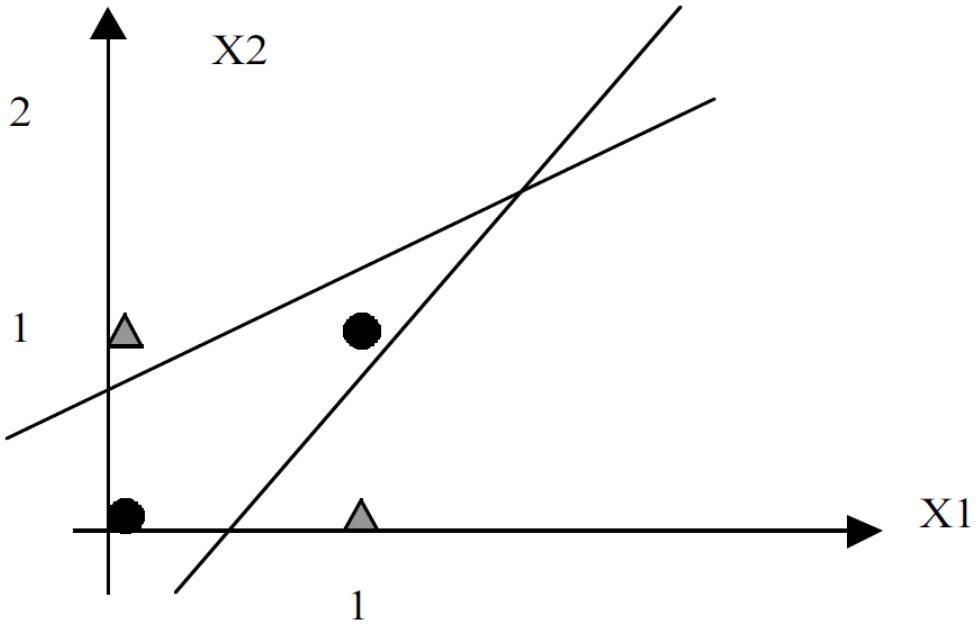
Arhitektura nevronske mreže za rešitev XOR problema.



Odločitvene ravnine za XOR problem

V omenjeni arhitekturi so uporabljeni McCulloh-Pitts nevroni, ki uporabljajo stopničasto aktivacijsko funkcijo. S primerno izbiro uteži mreže, lahko XOR implementiramo z uporabo odločitvenih ravnin, kot so prikazane na sliki.

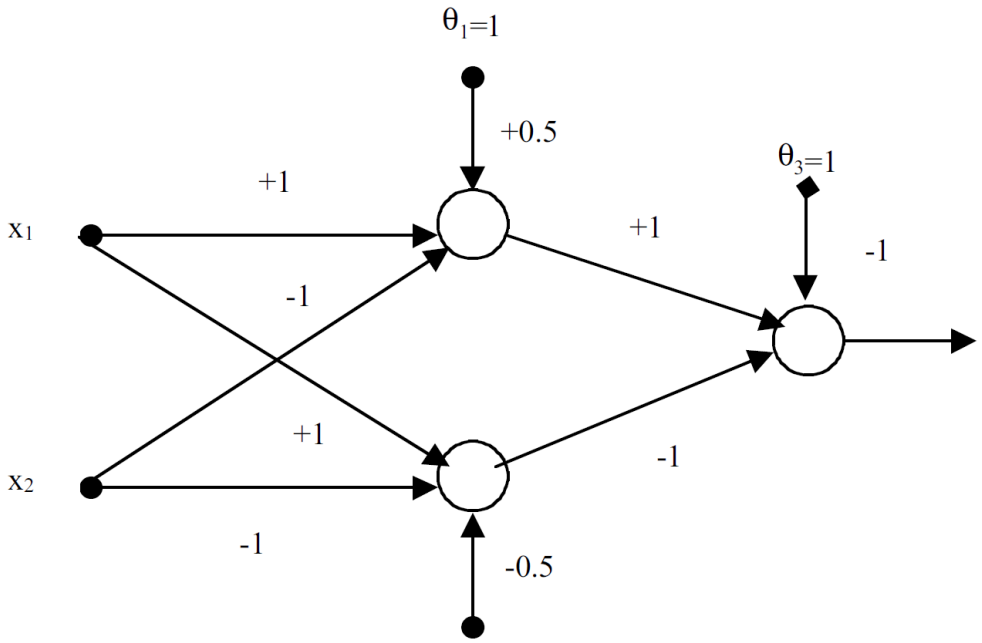
Odločitvene ravnine za rešitev XOR problema.



Primer (1/3)

Vzemimo, da so uteži in začetne vrednosti izbrane kot prikazuje slika. Nevroni so določeni z McCulloh-Pitts-ovim modelom (binarna stopničasta aktivacijska funkcija). Pokažimo, da podano omrežje reši XOR problem. Prav tako narišimo odločitvene ravnine, kot jih določa mreža.

Arhitektura nevronske mreže

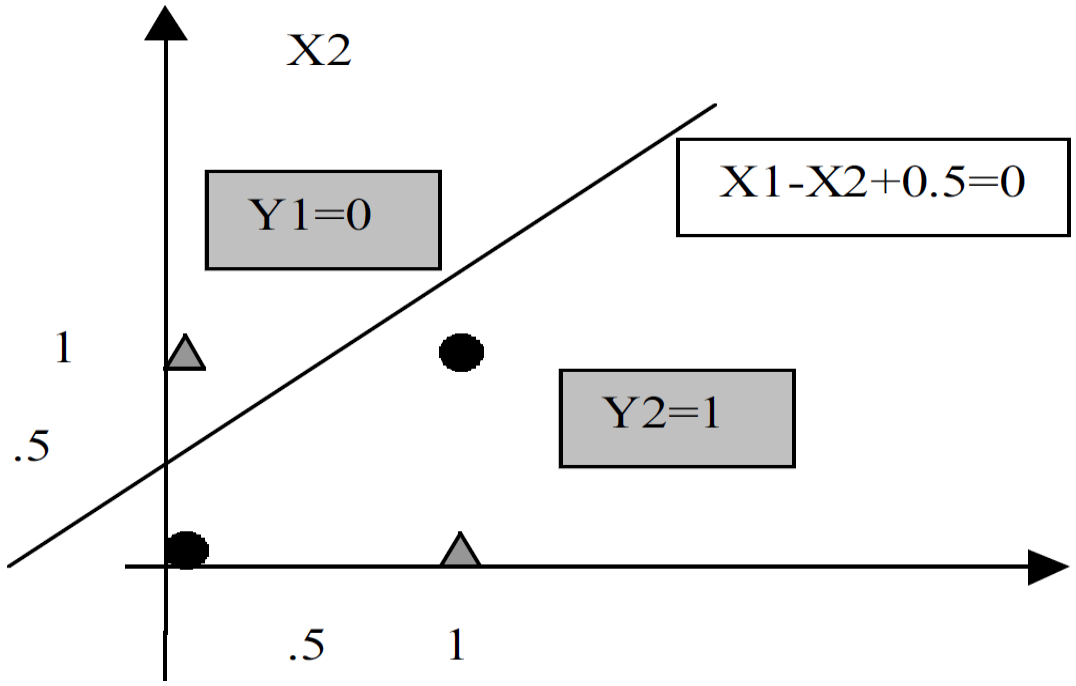


Izhode nevronov (pred aktivacijsko funkcijo) označimo kot O_1, O_2, and O_3. Izhodi prvega nivoja so določeni kot:

$$O_1 = x_1 - x_2 + 0.5$$
$$O_2 = x_1 - x_2 - 0.5$$

Primer (2/3)

Izhoda y_1 in y_2 sta določena ob upoštevanju binarne stopničaste aktivacijske funkcije.

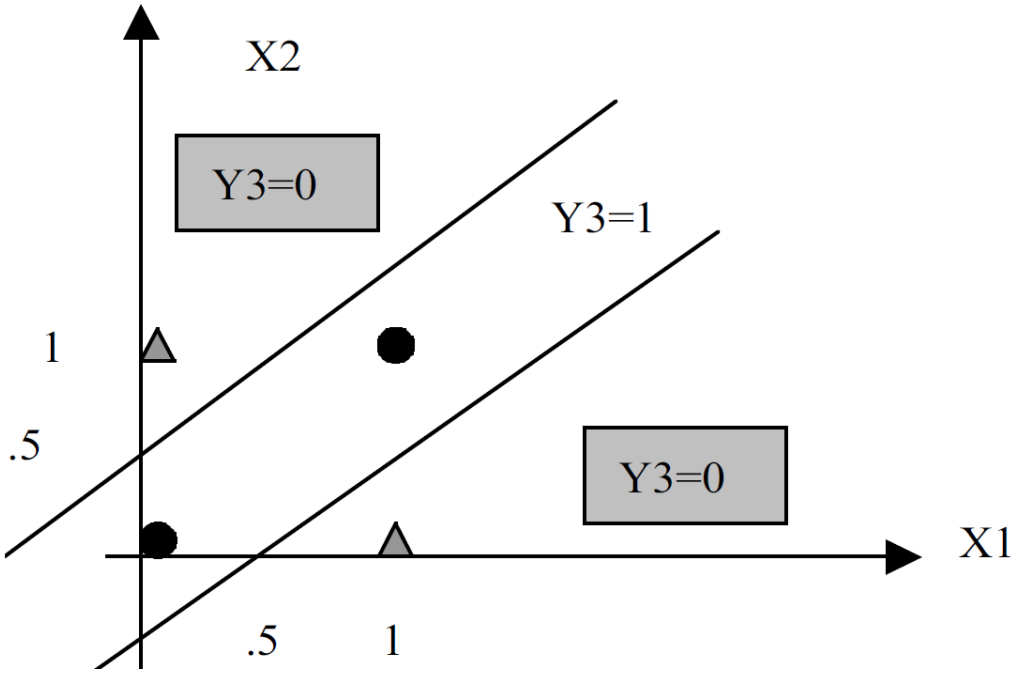


Primer (3/3)

Izhod drugega nivoja je določen kot:

$$O_3 = y_1 - y_2 - 1$$

Slika prikazuje odločitveni ravnini mreže.



ARHITEKTURE NEVRONSKIH MREŽ

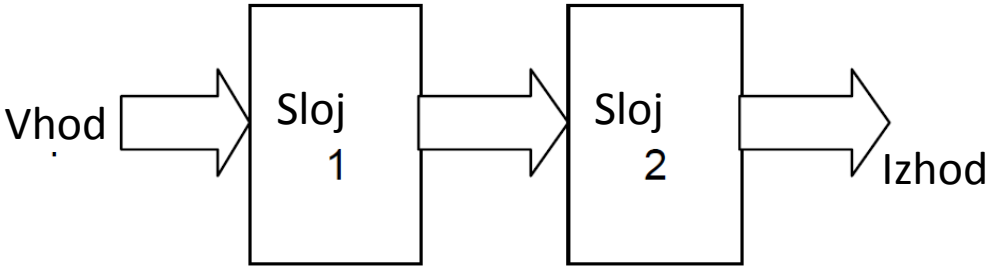
ARHITEKTURE NEVRONSKIH MREŽ

V tem delu je predstavljen konceptualni pregled arhitektur nevronskih mrež. Poudarek je na mehanizmu njihovega delovanja.

- Obstajajo različne klasifikacije nevronskih mrež glede na njihovo delovanje in/ali strukture.
- Mreže brez povratne povezave (feedforward) in z povratno povezavo (feedback).
- Opisani sta dve metodi učenja: nadzorovano in nenadzorovano učenje.
- Algoritem vzvratnega popravljanja (back propagation – BP).
- Mreža z radialnimi baznimi aktivacijskimi funkcijami (Radial Basis Function Network – RBFN) je mreža brez povratne povezave z nadorovanim učenjem.
- Kohonenovo samoorganiziranje
- Hopfieldove mreže.

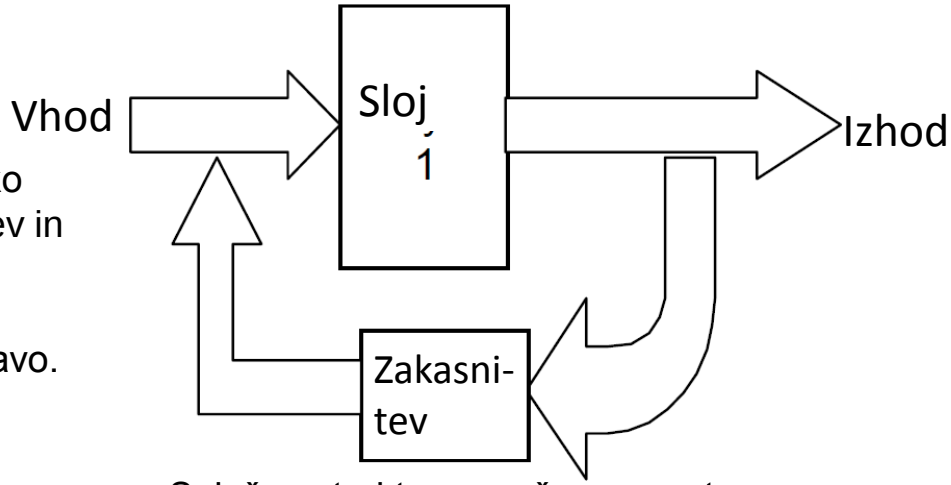
Mreže brez in z povratno povezavo (Feedforward in Feedback)

V strukturi nevronske mreže brez povratne povezave so edine sprejemljive povezave med izhodi sloja in vhodi naslednjega sloja. Povezave med izhodi sloja in vhodi istega ali prejšnjih slojev ne obstajajo.



Splošna struktura dvonivojske mreže brez povratne povezave

V mrežah s povratno povezavo lahko obstajajo povezave med izhodi slojev in vhodi istih ali prejšnjih slojev. Slika prikazuje preprosto enonivojsko nevronsko mrežo s povratno povezavo.



Splošna struktura mreže s povratno povezavo.

Mreže z nadzorovanim in nenadzorovanim učenjem

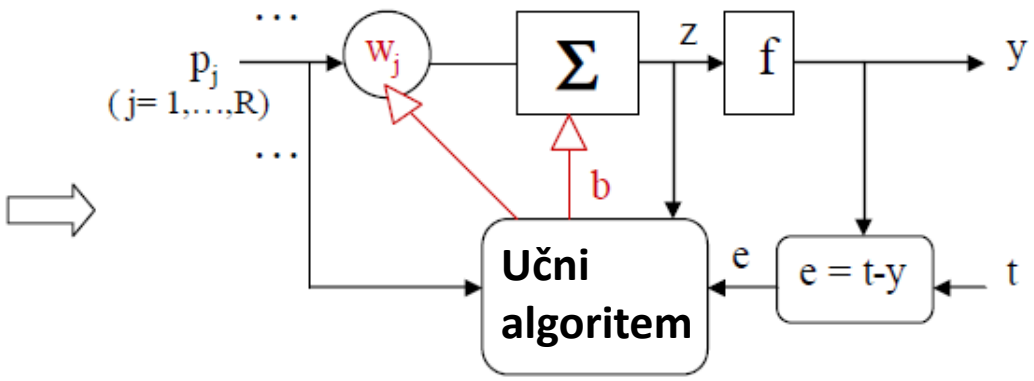
Nadzorovano učenje:
Nadzorovano učenje zahteva zunanjega učitelja za nadzor nad učenjem in za združevanje globalnih informacij. Vlogo učitelja lahko prevzame učna množica podatkov ali opazovalec, ki ocenjuje učinkovitost mreže. Primer algoritma nadzorovanega učenja je algoritem najmanjših kvadratov (least mean square – LMS) in njegova posplošitev znana kot algoritem vzvratnega razširjanja (back propagation algorithm) in mreža z radialnimi baznimi aktivacijskimi funkcijami.

Nenadzorovano učenje :
Ob odsotnosti zunanjega učitelja se mora sistem organizirati sam z notranjimi kriteriji in lokalnimi informacijami vgrajenimi v mrežo. Nenadzorovanemu učenju zato pravijo tudi *samoorganizirano učenje*, tj. samostojno učenje klasificiranja. V tej kategoriji so na voljo samo vhodni podatki , ki jih mreža nato razvrsti v različne skupine. Kohonenova mreža je primer nenadzorovanega učenja.

Pravila učenja

Pravila učenja (učni algoritmi): postopek/algoritem za prilagoditev uteži in začetnih vrednosti, ki jih potrebuje mreža za svoje delovanje.

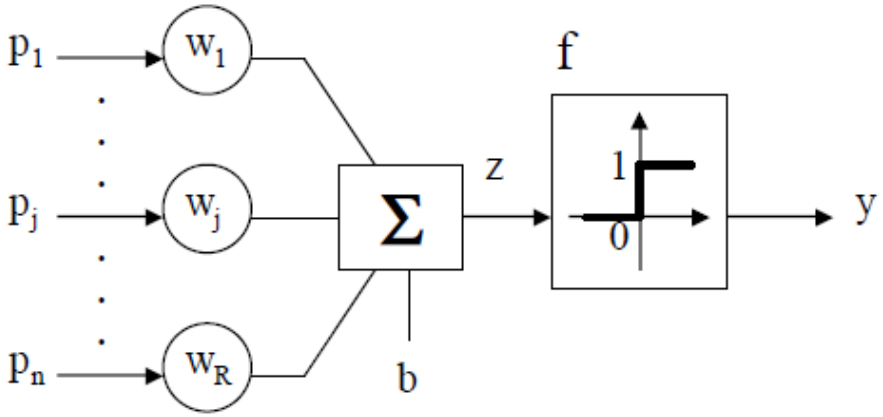
Nadzorovano učenje
Za dano učno množico {p(1),t(1)}, …, {p(n),t(n)}, kjer je p(i) element vhodnega vektorja, t(i) pa pripadajoča ciljna vrednost izhoda y, izračuna učni algoritem novo vrednost nevronovih uteži in začetne vrednosti.



Nenadzorovano učenje
Uteži so določene samo glede na stanje vhoda. Večina teh algoritmov razvršča vhodne parametre v končno število razredov, npr. aplikacije z vektorsko kvantizacijo.

Algoritem učenja perceptrona

- **Nadzorovano učenje**
  t ← ciljna vrednost
  $e = t - y$ ← napaka



Zaradi stopničaste aktivacijske funkcije nevrona lahko y, t in e zavzamejo le binarne vrednosti.

$\boldsymbol{p} = (p_1, \ldots, p_R)^T$ Vhodni stolpčni vektor

$\boldsymbol{W} = (x_1, \ldots, x_R)$ Vrstični vektor uteži

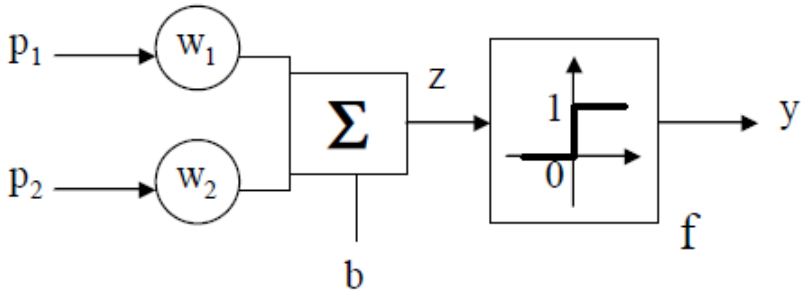
Učni algoritem perceptrona:

$$\begin{cases} \text{if } e = 1, \text{ then } \boldsymbol{W}^{new} = \boldsymbol{W}^{old} + \boldsymbol{p}, \; b^{new} = b^{old} + 1; \\ \text{if } e = -1, \text{ then } \boldsymbol{W}^{new} = \boldsymbol{W}^{old} - \boldsymbol{p}, \; b^{new} = b^{old} - 1; \\ \text{if } e = 0, \text{ then } \boldsymbol{W}^{new} = \boldsymbol{W}^{old}. \end{cases}$$

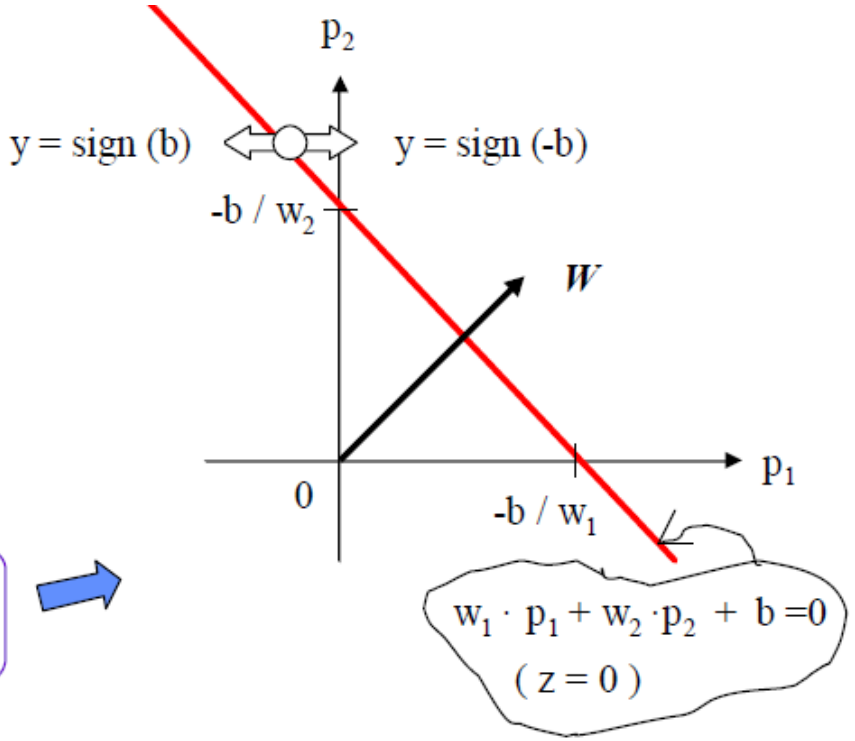
$$\Rightarrow \quad \boxed{\begin{array}{l} \boldsymbol{W}^{new} = \boldsymbol{W}^{old} + e\,\boldsymbol{p}^T \\ b^{new} = b^{old} + e \end{array}}$$

Primer dvovhodnega perceptrona

Dvovhodni perceptron



$$y = hardlim\ (z) = hardlim\{ [w_1 , w_2] \cdot [p_1 , p_2]^T + b\}$$

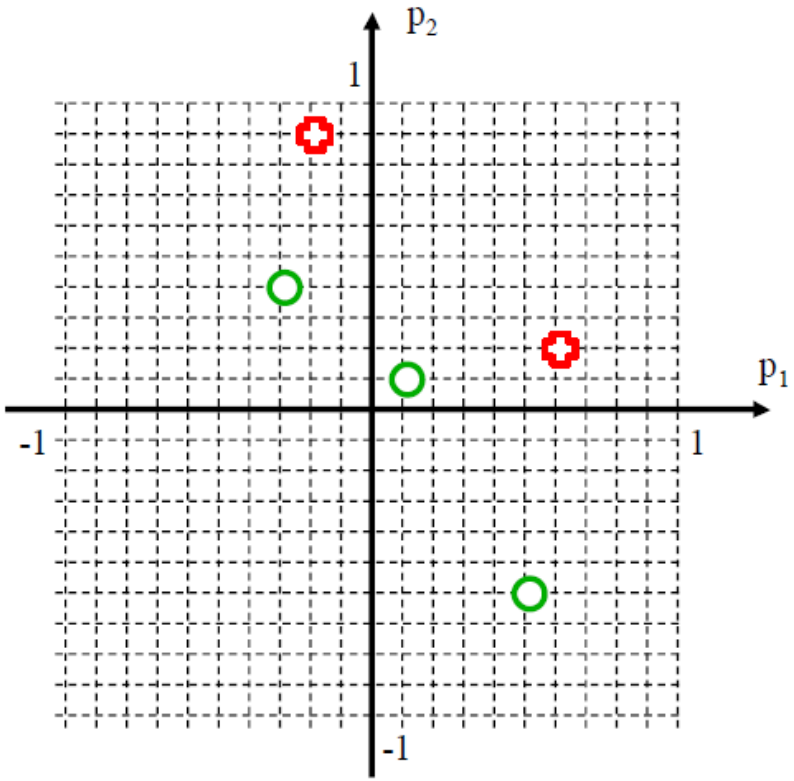


Dva razreda (linearno ločljivi območji) v 2D vhodnem prostoru (p_1,p_2) sta ločena s premico $z = 0$.
Meja je vedno ortogonalna na vektor uteži ***W***.

Primer: Učenje dvovhodnega perceptrona klasificiranja pet vhodnih vektorjev

$$\left\{\begin{matrix} p(1) = (0.6, 0.2)^T \\ t(1) = 1 \end{matrix}\right\} \quad \left\{\begin{matrix} p(2) = (-0.2, 0.9)^T \\ t(2) = 1 \end{matrix}\right\} \quad \left\{\begin{matrix} p(3) = (-0.3, 0.4)^T \\ t(3) = 0 \end{matrix}\right\} \quad \left\{\begin{matrix} p(4) = (0.1, 0.1)^T \\ t(4) = 0 \end{matrix}\right\} \quad \left\{\begin{matrix} p(5) = (0.5, -0.6)^T \\ t(5) = 0 \end{matrix}\right\}$$

Rešitev v Matlabu:




```
P=[0.6 -0.2 -0.3 0.1 0.5;
   0.2  0.9  0.4 0.1 -0.6];
T=[1 1 0 0 0];
W=[-2 2];
b=-1;
plotpv(P,T);
plotpc(W,b);
nepoc=0
Y=hardlim(W*P+b);
while any(Y~=T)
Y=hardlim(W*P+b);
E=T-Y;
[dW,db]= learnp(P,E);
W=W+dW;
b=b+db;
nepoc=nepoc+1;
disp('epochs='),disp(nepoc),
disp(W), disp(b);
plotpv(P,T);
plotpc(W,b);
end
```

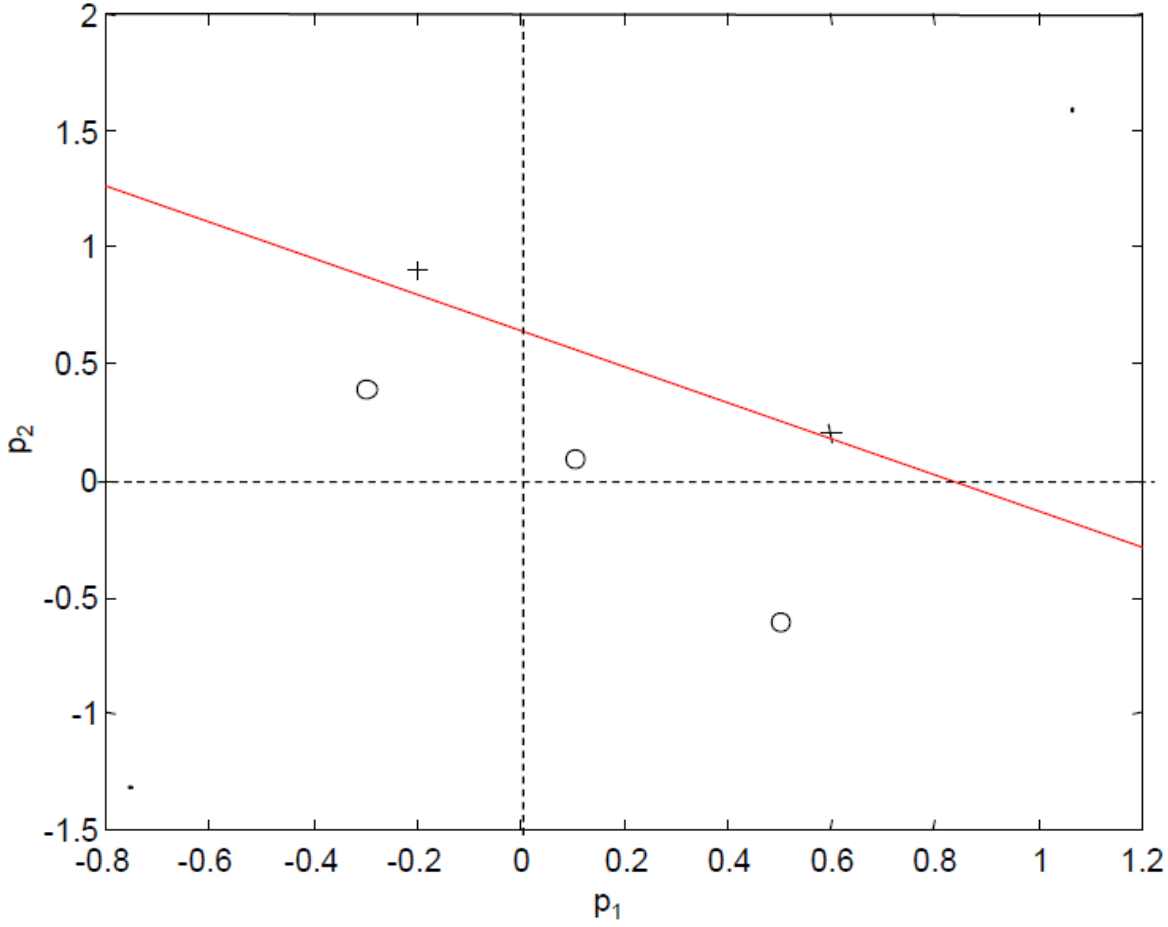
Primer: rezultat učenja

Po aktivaciji *nepoc* = 11 (št. aktivacij ob začetnem vektorju uteži ***W*** = [-2 2] in začetni vrednosti *b* = -1) so uteži:

$$w_1 = 2.4$$
$$w_2 = 3.1$$

Začetna vrednost pa je:

$$b = -2$$



Povzetek

Večji je vhodni vektor p, večji je njegov učinek na vektor uteži W med procesom učenja
→ (pre)dolgi učni procesi so lahko posledica vhodnega vektorja z veliko večjo magnitudo od ostalih vhodnih vektorjev (pravimo mu tudi “outlier” vektor).

→ **Normaliziran algoritem učenja perceptrona**: učinek vsakega vhodnega vektorja na uteži ima enako magnitudo

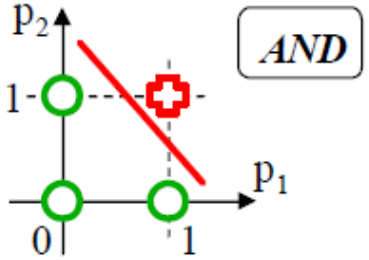
$$W^{new} = W^{old} + e\,p^T / \|p\|$$

$$b^{new} = b^{old} + e$$

Stopničasta aktivacijska funkcija perceptrona zagotavlja razvrščanje vhodnih vektorjev z odločanjem, kateremu od obeh *linearno ločljivih razredov* pripadajo.

$\mathbf{p} = [0\ 0\ 1\ 1;\ 0\ 1\ 0\ 1]$

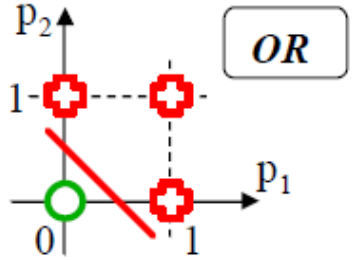
$\mathbf{t}_{AND} = [0\ 0\ 0\ 1]$



$\mathbf{W} = [2\ 2]$
$b = -3$

$\mathbf{p} = [0\ 0\ 1\ 1;\ 0\ 1\ 0\ 1]$

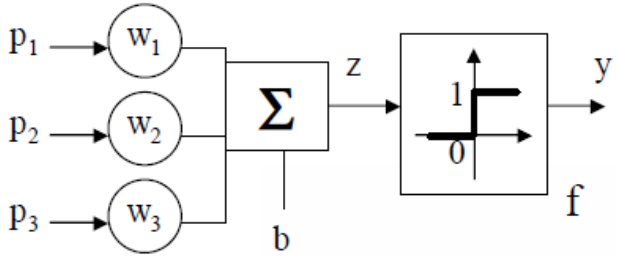
$\mathbf{t}_{OR} = [0\ 1\ 1\ 1]$



$\mathbf{W} = [2\ 2]$
$b = -1$

Perceptron s tremi vhodi

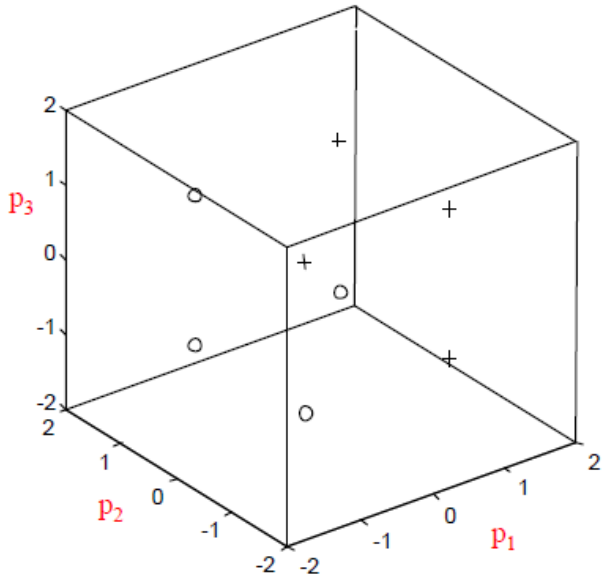
Razreda v 3D vhodnem prostoru (p_1,p_2,p_3) sta ločena z ravnino $z = 0$.

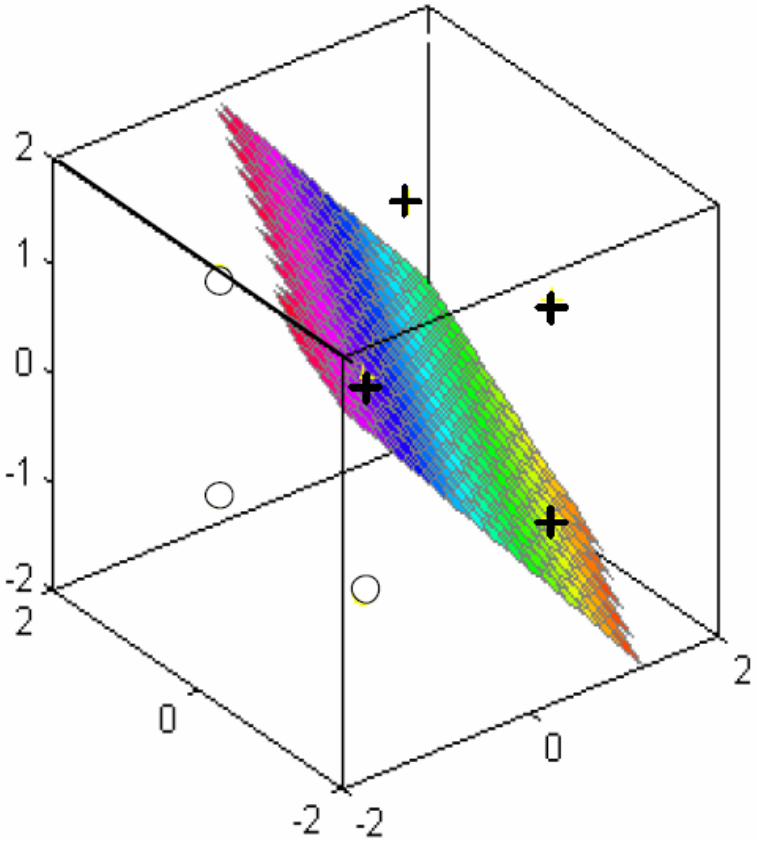


$$y = hardlim(z) = hardlim\{[w_1, w_2, w_3] \cdot [p_1, p_2 p_3]^T + b\}$$

EXAMPLE

$$P = \begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1; \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1; \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad T = [0\ 1\ 0\ 0\ 1\ 1\ 1\ 0]$$





Koncepti vzvratnega razširjanja (backpropagation)

Vzvratno razširjanje (backpropagation)

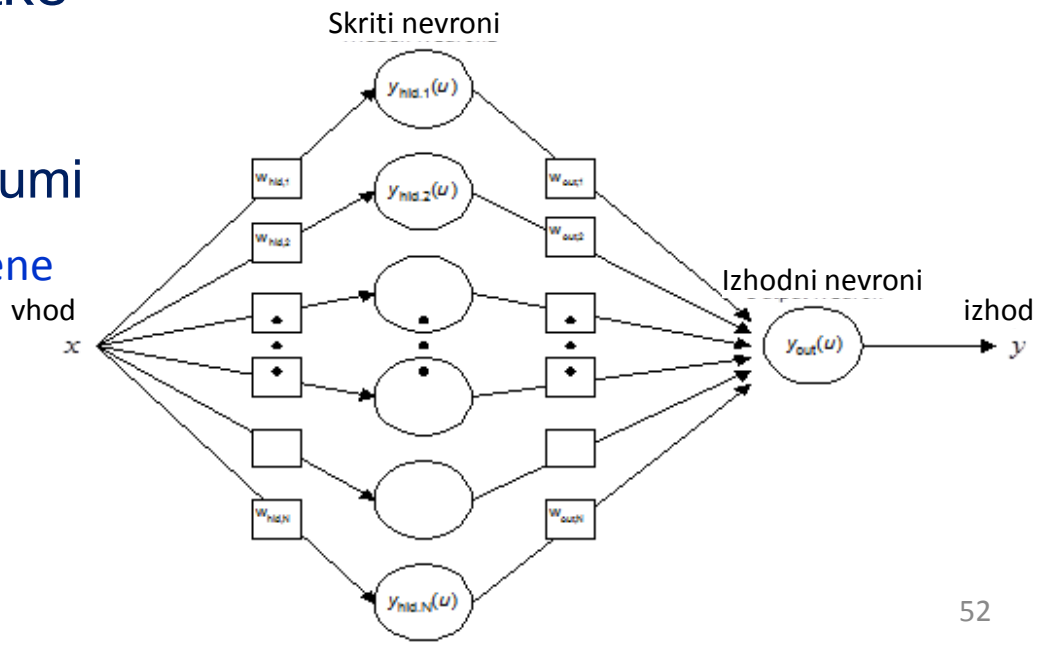
Najpogostejša metoda za določanje uteži v mreži

- nadzorovano učenje
- Osnovni algoritem je osnovan na minimiziranju napake v mreži z uporabo odvodov funkcije napake
  - ► preprost
  - ► počasen
  - ► težave z lokalnimi minimumi

Določi uteži (w) za učenje določene ciljne funkcije:

$$y = f(x)$$

Glede na učno množico $X \rightarrow Y$



Algoritem vzvratnega razširjanja

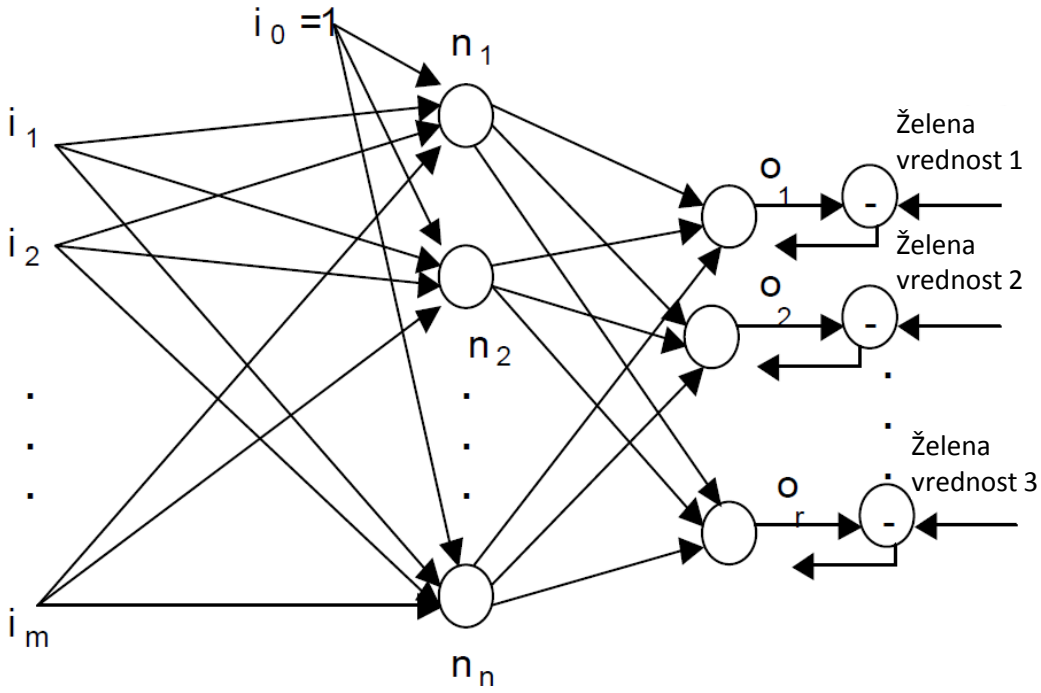
Algoritem sestavljata dve fazi:

Učna faza in spominska faza

Najprej se uteži mreže naključno določijo. Izhod mreže se izračuna in primerja z želeno vrednostjo. V nadaljevanju se izračuna napaka mreže ter se uporabi za prilagoditev uteži izhodnega sloja. Napaka mreže se prenese nazaj in se uporabi za prilagoditev uteži prejšnjih slojev. Slika prikazuje ustvarjanje in razširjanje vrednosti napake v mreži.

Slika:

Vzvratno razširjanje napake v dvonivojski mreži.



Vzvratno razširjanje

- Najpogostejše merilo napake je povprečna vrednost kvadrata napake:

$$E = (target - output)^2$$

- Delni odvodi napake wrt uteži:

▸ Izhodni nevroni:

naj bo: $\delta_j = f'(net_j)\,(target_j - output_j)$ j = izhodni nevron

$\partial E/\partial w_{ji} = -output_i\,\delta_j$ i = nevron v zadnjem skritem sloju

▸ Skriti nevroni:

naj bo: $\delta_j = f'(net_j)\,\sum(\delta_k w_{kj})$ j = skriti nevron

$\partial E/\partial w_{ji} = -output_i\,\delta_j$ i = nevron prejšnjega sloja

k = nevron naslednjega soja

Vzvratno razširjanje

- Izračun odvodov se širi nazaj po mreži, od koder tudi ime vzvratnega razširjanja.

- Odvodi kažejo v smeri največjega povečevanja funkcije napake.

- majhen korak (učna hitrost, krivulja) v nasprotni smeri bo povzročil največje zmanjšanje (lokalne) funkcije napake:

$$w_{new} = w_{old} - \alpha . \partial E / \partial w_{old}$$

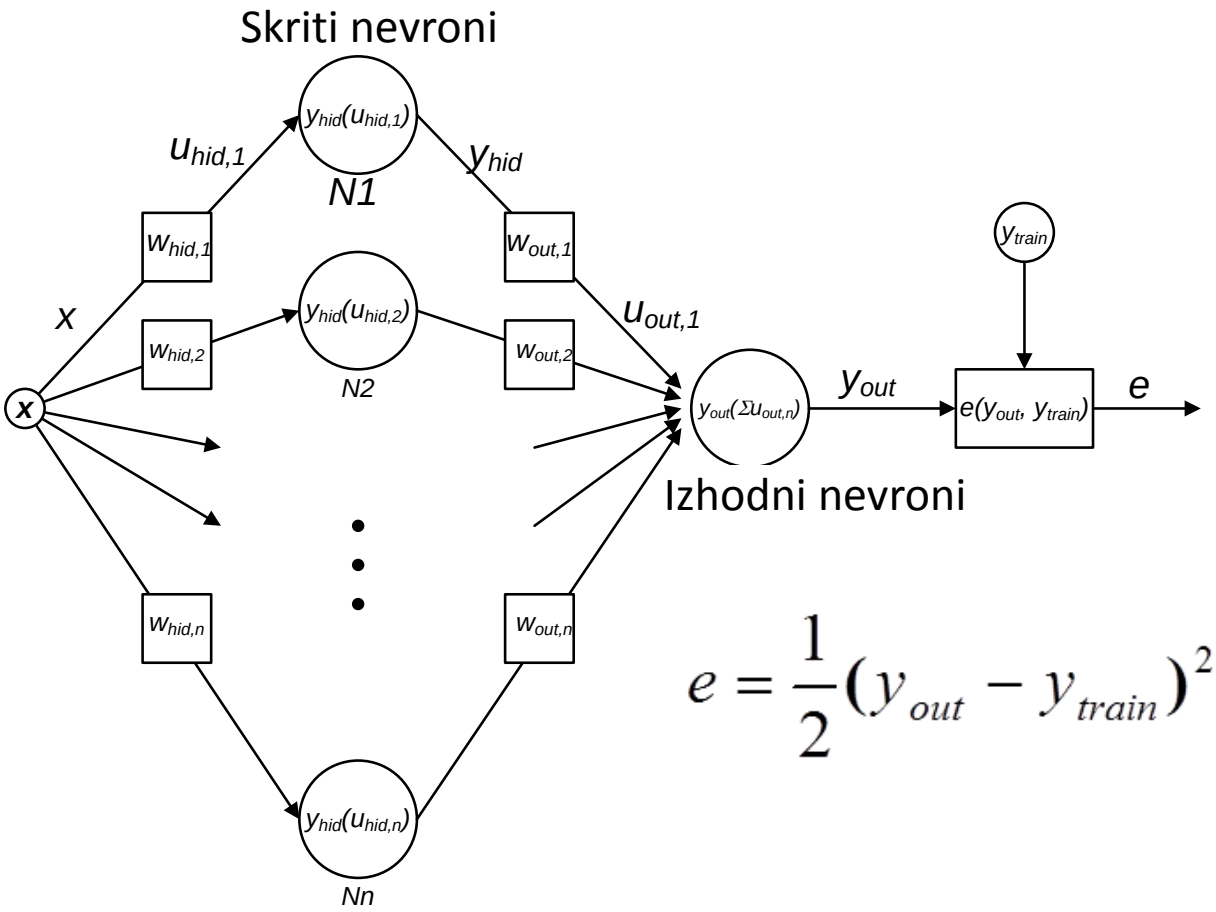
kjer je α učna hitrost

Prilagajanje uteži je torej:

$$w(k+1) = w(k) + \Delta w$$

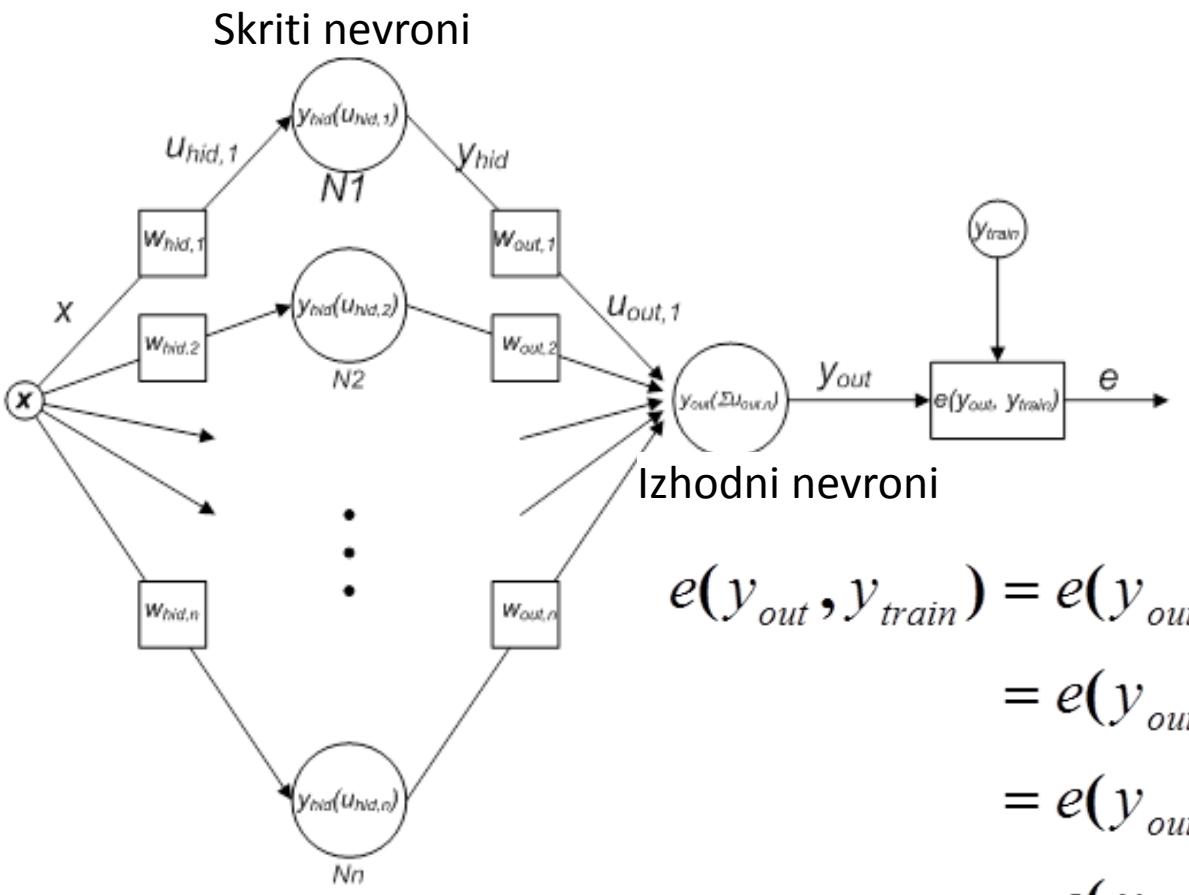
Vzvratno razširjanje – učna napaka

Učna napaka: e



$$e = \frac{1}{2}(y_{out} - y_{train})^2$$

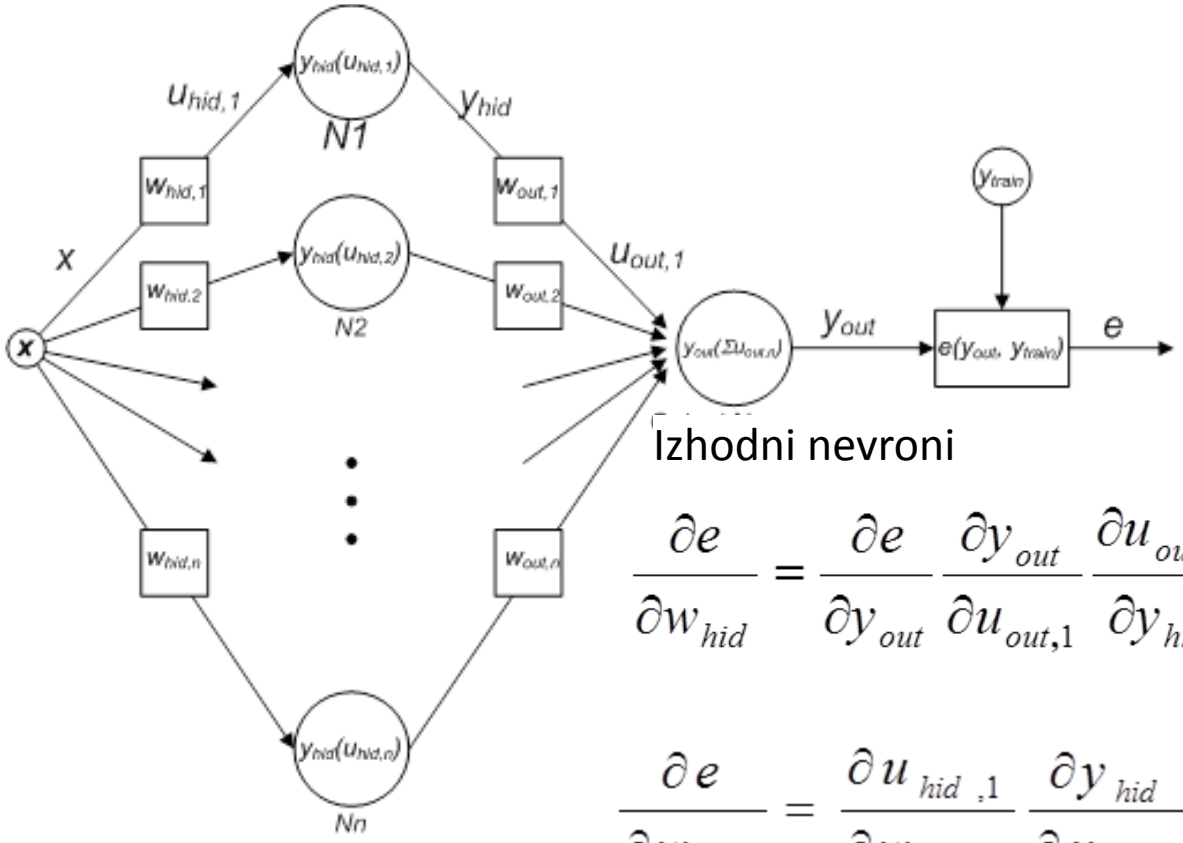
Formulacija vzvratnega razširjanja



$$e(y_{out}, y_{train}) = e(y_{out}(u_{out,1}), y_{train})$$
$$= e(y_{out}(w_{out,1} y_{hid,1}), y_{train})$$
$$= e(y_{out}(w_{out,1} y_{hid}(u_{hid,1})), y_{train})$$
$$= e(y_{out}(w_{out,1} y_{hid}(w_{hid,1} x)), y_{train})$$

Formulacija vzvratnega razširjanja

Skriti nevroni

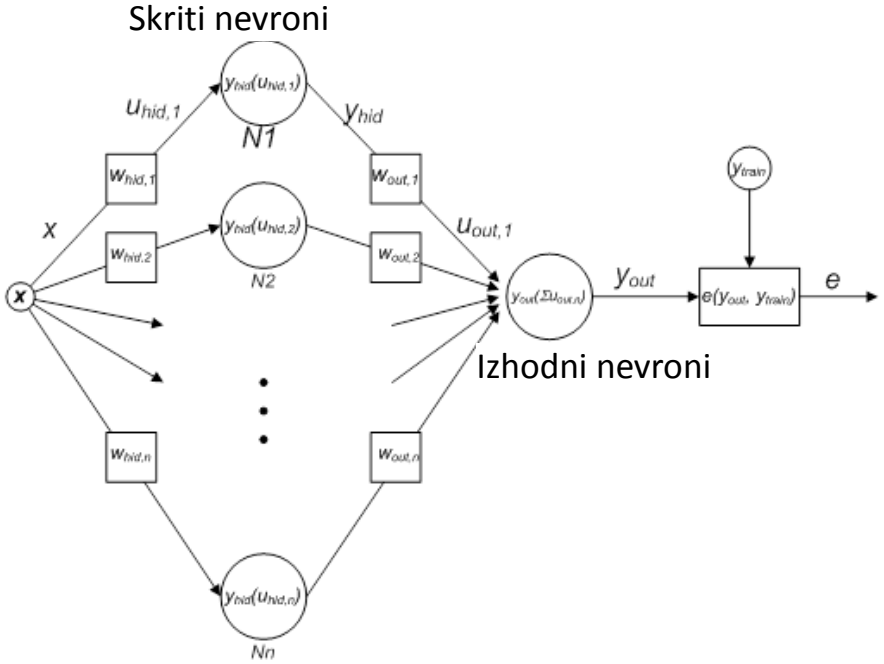


Izhodni nevroni

$$\frac{\partial e}{\partial w_{hid}} = \frac{\partial e}{\partial y_{out}} \frac{\partial y_{out}}{\partial u_{out,1}} \frac{\partial u_{out,1}}{\partial y_{hid}} \frac{\partial y_{hid}}{\partial u_{hid,1}} \frac{\partial u_{hid,1}}{\partial w_{hid,1}}$$

$$\frac{\partial e}{\partial w_{hid}} = \frac{\partial u_{hid,1}}{\partial w_{hid,1}} \frac{\partial y_{hid}}{\partial u_{hid,1}} \frac{\partial u_{out,1}}{\partial y_{hid}} \frac{\partial y_{out}}{\partial u_{out,1}} \frac{\partial e}{\partial y_{out}}$$

Formulacija vzvratnega razširjanja



$$\frac{\partial u_{hid,1}}{\partial w_{hid,1}} = \frac{\partial}{\partial w_{hid,1}} w_{hid,1} x$$

$$= x$$

$$\frac{dy_{hid}(u)}{du} = \frac{d}{du}\left[\frac{1}{1+e^{-u}}\right]$$

$$= (1+e^{-u})^{-2}(-e^{-u})$$

$$= \frac{1+e^{-u}-1}{(1+e^{-u})^2}$$

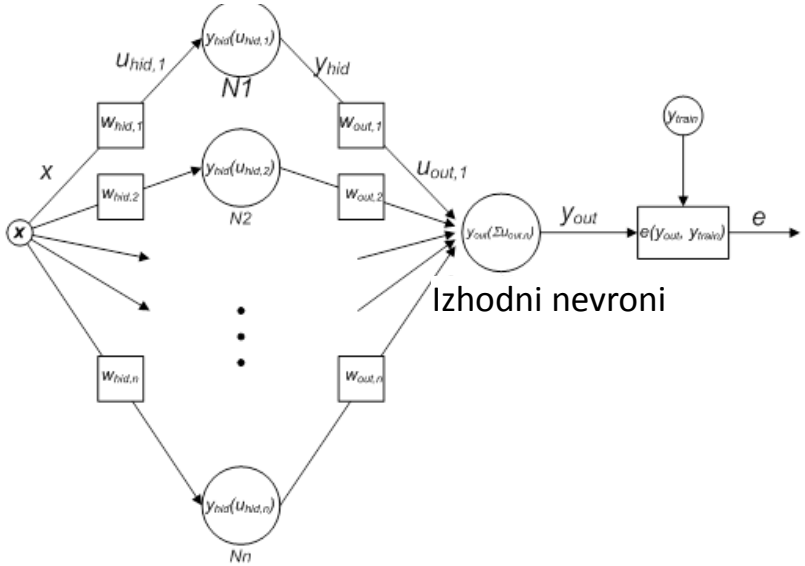
$$= \frac{1+e^{-u}}{(1+e^{-u})^2} - \frac{1}{(1+e^{-u})^2}$$

$$= \frac{1}{(1+e^{-u})} - \frac{1}{(1+e^{-u})^2}$$

$$= \frac{1}{(1+e^{-u})}\left[1 - \frac{1}{(1+e^{-u})}\right]$$

$$= y_{hid}(u_{hid,1})\left[1 - y_{hid}(u_{hid,1})\right]$$

Formulacija vzvratnega razširjanja



$$\frac{\partial u_{out,1}}{\partial y_{hid}} = \frac{\partial}{\partial y_{hid}} w_{out,1} y_{hid,1}$$

$$= w_{out,1}$$

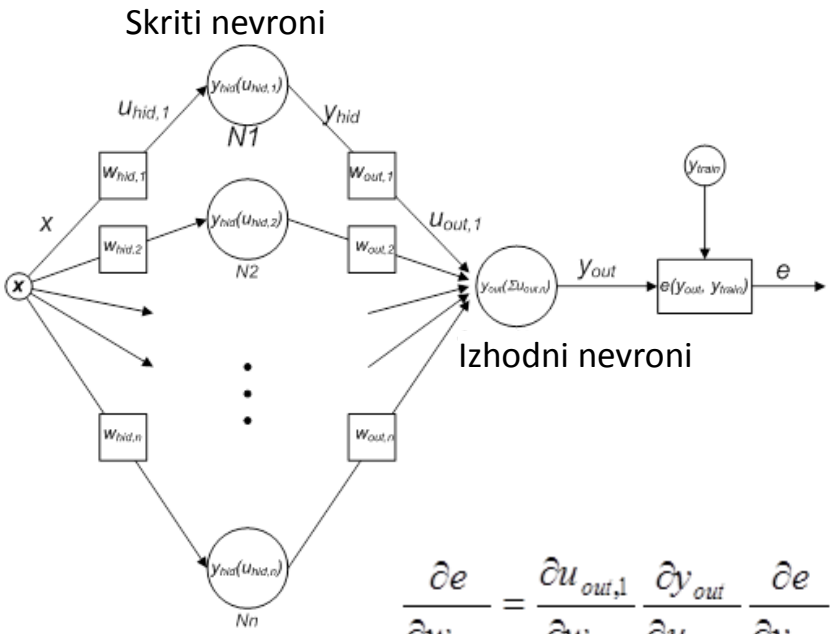
$$\frac{\partial e}{\partial y_{out}} = \frac{\partial}{\partial y_{out}} \frac{1}{2}(y_{out} - y_{train})^2$$

$$= (y_{out} - y_{train})$$

$$\frac{\partial e}{\partial w_{hid}} = \frac{\partial u_{hid,1}}{\partial w_{hid,1}} \frac{\partial y_{hid}}{\partial u_{hid,1}} \frac{\partial u_{out,1}}{\partial y_{hid}} \frac{\partial y_{out}}{\partial u_{out,1}} \frac{\partial e}{\partial y_{out}}$$

$$= (x)\left(y_{hid}(u_{hid,1})\left[1 - y_{hid}(u_{hid,1})\right]\right)(w_{out,1})(1)(y_{out} - y_{train})$$

Formulacija vzvratnega razširjanja



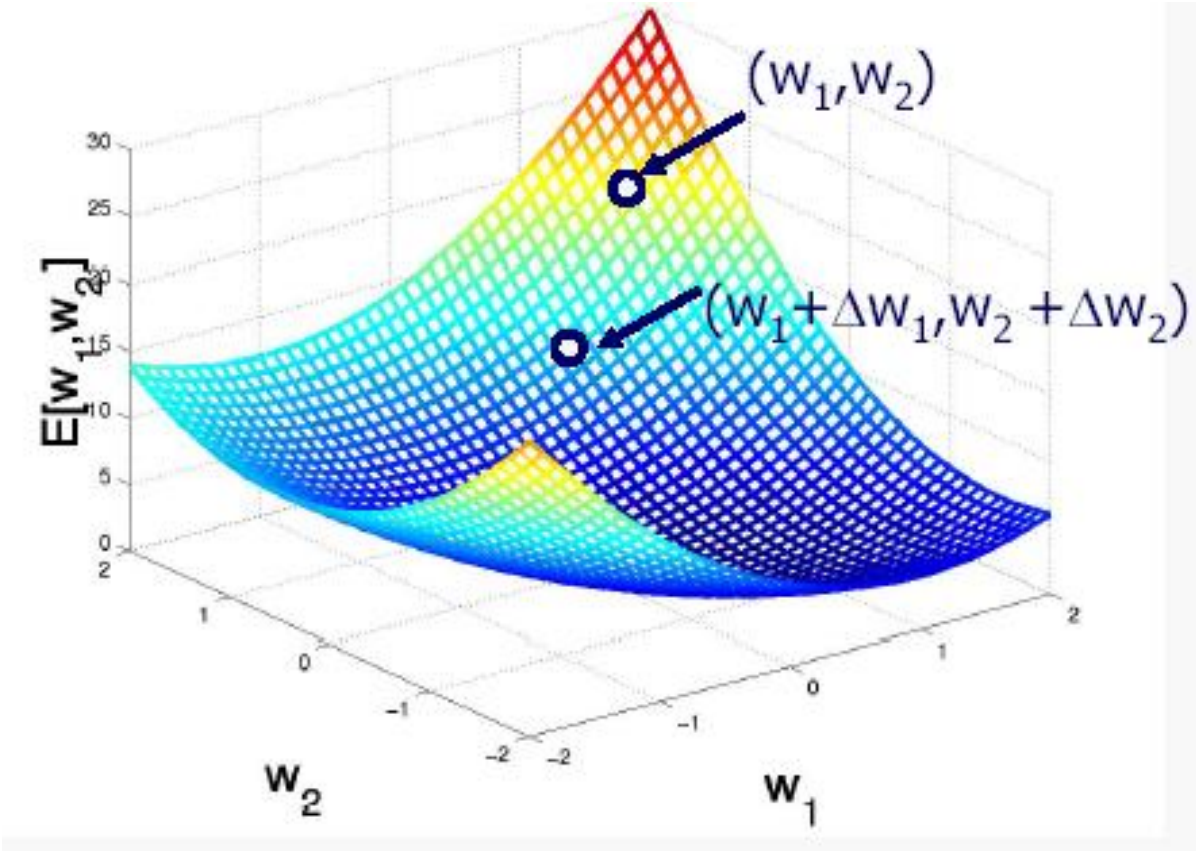
$$\frac{\partial e}{\partial w_{out}} = \frac{\partial u_{out,1}}{\partial w_{out}} \frac{\partial y_{out}}{\partial u_{out,1}} \frac{\partial e}{\partial y_{out}}$$

$$= \left(\frac{\partial}{\partial w_{out}} w_{out,1} y_{hid,1} \right) \left(\frac{\partial}{\partial u_{out,1}} \left[u_{out,1} + u_{out,2} + \ldots + u_{out,N} \right] \right) \left(\frac{\partial}{\partial y_{out}} \frac{1}{2} (y_{out} - y_{train})^2 \right)$$

$$= (y_{hid})(1)(y_{out} - y_{train})$$

Gradientno zmanjševanje

Definirajmo funkcijo napake ciljne koncepte in izhod nevronske mreže.
Cilje je spremeniti uteži tako, da bo napaka minimalna.



Algoritem gradientnega zmanjševanja

- Vsak učni primer je par oblike <***x***,*t*>, kjer je ***x*** vektor vhodnih vrednosti, *t* pa je ciljna izhodna vrednost η pa je učna hitrost (npr. 0,5)
- Na začetku določimo majhne naključne vrednosti uteži w_i
- Dokler ni izpolnjen končni pogoj:
  - enačimo vsak Δw_i z 0
  - Za vsako linearno enoto uteži w_i določimo

    $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$
  - Za vsako linearno enoto uteži w_i določimo

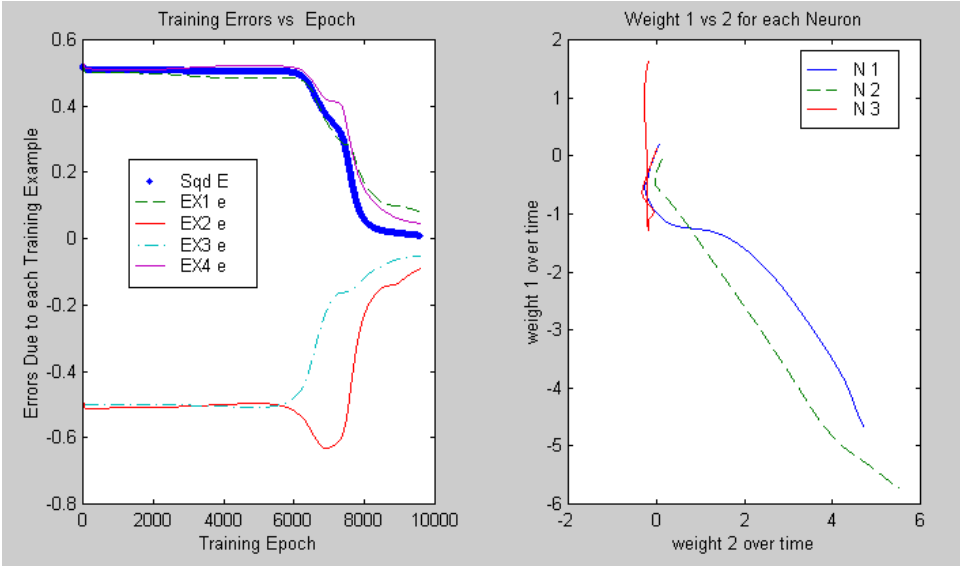
    $w_i \leftarrow w_i + \Delta w_i$

Primer: XOR problem

- En skriti sloj: 3 Sigmoidni nevroni
- 2 vhoda, 1 izhod

	x1	x2	y
Primer 1	0	0	**0**
Primer 2	0	1	**1**
Primer 3	1	0	**1**
Primer 4	1	1	**0**

Učna napaka



Primer: XOR problem

Zacetne_utezi =

0.0654 0.2017 0.0769 0.1782 0.0243 0.0806 0.0174
0.1270 0.0599 0.1184 0.1335 0.0737 0.1511

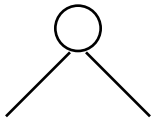
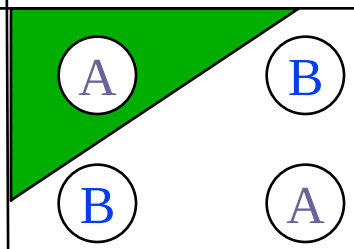
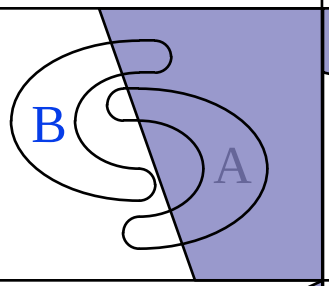
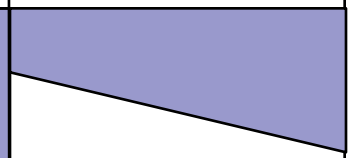
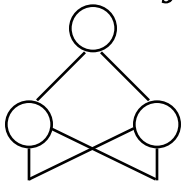
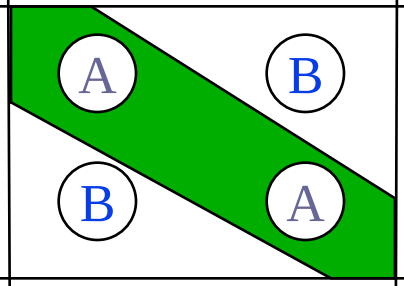
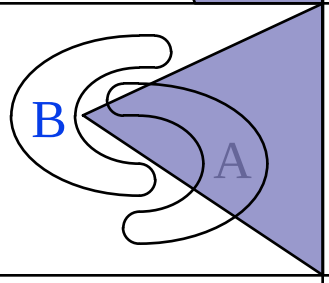
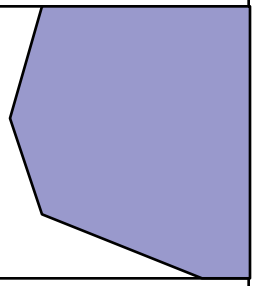
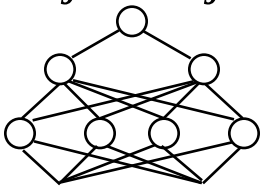
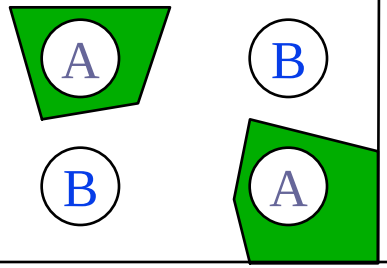
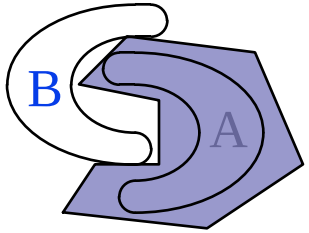
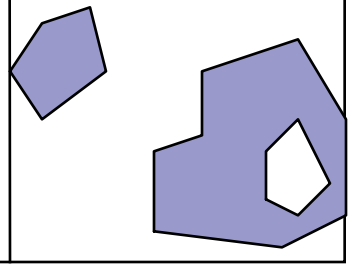
Koncne_utezi =

4.6970 -4.6585 2.0932 5.5168 -5.7073 -3.2338 -0.1886
1.6164 -0.1929 -6.8066 6.8477 -1.6886 4.1531

Preslikava, dobljena z naučeno nevronsko mrežo

	x1	**x2**	**y**
Primer 1	0	0	0.0824
Primer 2	0	1	0.9095
Primer 3	1	0	0.9470
Primer 4	1	1	0.0464

Različni nelinearno ločljivi problemi

Struktura	*Tip odločitvenega območja*	*XOR Problem*	*Razredi z mrežnimi očesi*	*Splošne oblike območij*
En nivo 	*Polravnina omejena s hiperravnino*			
Dva nivoja 	*Konveksna odprta ali zaprta območja*			
Trije nivoji 	Samovoljno (kompleksnost je omejena s številom nevronov)			

Izbira števila skritih slojev

Število vhodnih in izhodnih nevronov se ujema s številom vhodov mreže in s številom želenih izhodov. Izbira števila vmesnih skritih slojev in nevronov v njih pa je odvisno od uporabe mreže.

Matematični pristop k določitvi optimalnega števila skritih slojev zato ne obstaja.

Število skritih slojev se lahko določi tako, da ob učenju mreže uporabimo različne konfiguracije in nato izberemo tisto, ki ima ob čim manjšem številu slojev in nevronov minimalno napako (npr. koren povprečne vrednosti kvadratov - RMS) in je čim hitrejša (bolj efektivna). V splošnem velja, da z drugim skritim slojem izboljšamo predikcijsko zmožnost mreže zaradi lastnosti nelinearne ločljivosti mreže. Z vsakim naslednjim dodatnim slojem pa na predikcijski zmožnosti mreža ne pridobiva več toliko (je podobna, kot pri dveh skritih slojih), vendar pa se začne podaljševati čas učenja zaradi bolj kompleksne strukture.

Normalizacija vhodne in izhodne podatkovne zbirke

Nevronske mreže zahtevajo normalizacijo vhodnih in izhodnih podatkov na enako magnitudno stopnjo. Če podatki niso normalizirani, potem lahko imajo posamezne spremenljivke neupravičeno večji vpliv oziroma so bolj signifikantne kot dejansko so.
Na primer, če ima vhodna spremenljivka i_1 vrednost 50.000, i_2 pa 5, potem mora biti utež za drugo spremenljivko veliko večja od uteži za prvo spremenljivko, če želimo, da ima druga spremenljivka sploh kakšen pomen.

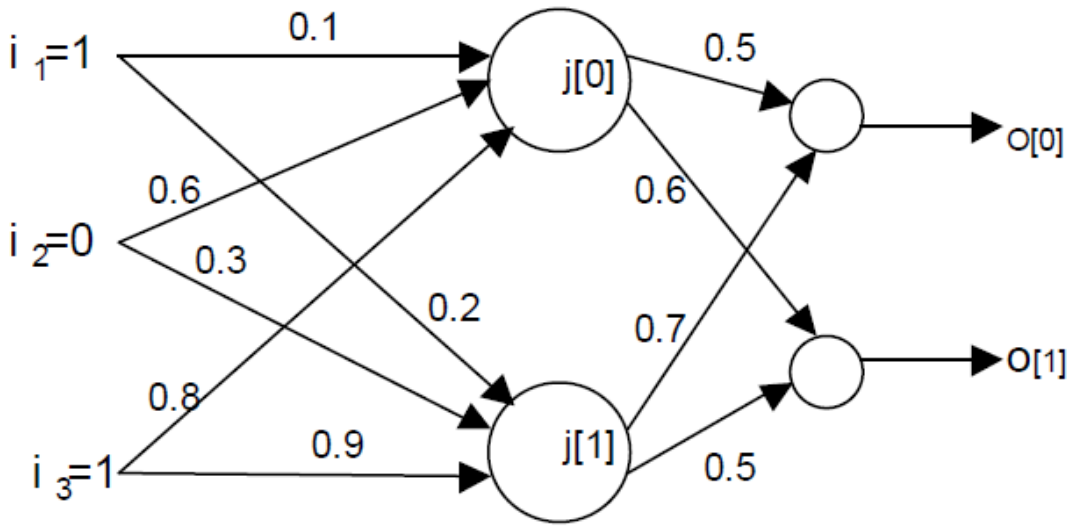
Povzetek učne faze algoritma vzvratnega razširjanja

1. Določimo začetne vrednosti uteži mreže.
2. Skaliramo vhodne/izhodne podatke.
3. Izberemo ustrezno strukturo mreže (tudi število skritih slojev in število nevronov v vsakem sloju).
4. Določimo aktivacijske funkcije nevronov, ki so lahko enake za vso mrežo ali pa se razlikujejo med različnimi sloji.
5. V učni množici določimo učni par in pripeljemo vhodni vektor na vhod mreže.
6. Izračunamo izhod mreže glede na začetne vrednosti uteži in na vhodne podatke.
7. Izračunamo napako med dejanskim izhodom mreže in med želenim izhodom, ki je določen z izhodnim vektorjem v učnem paru.
8. Napako razširimo nazaj po nevronih mreže in prilagodimo uteži tako, da jo minimiziramo. Začnemo pri izhodnem sloju in se pomikamo nazaj proti izhodu.
9. Ponovimo korake 5 – 8 za vsak vektor učne množice dokler ni napaka manjša od vnaprej določene/izbrane vrednosti.

Primer

Vzemimo mrežo na sliki

$$f(x) = \frac{1}{1+e^{-x}} \Rightarrow f'(x) = f(x)[1-f(x)]$$



Primer 1

Iteracija št. 1

Korak 1: inicializacija

Korak 2: Izračun izhodnih vrednosti

$$J[0] = f(W_j[0].I) = f(0.1*1+0.6*0+0.8*1 = f(0.9) = 0.7109$$

$$J[1] = f(W_j[1].I) = f(1.1) = 0.7503$$

$$O[0] = f(W_k[0].J) = 0.5*0.7109+0.7*0.7503 = 0.88066$$

$$O[1] = f(W_k[1].J) = 0.6*0.7109+0.5*0.7503 = 0.80169$$

Korak 3: izračun napake

$$\Delta k[0] = d_0 - k[0] = 0 - 0.88066 = -0.88066$$

$$\Delta k[1] = d_1 - k[1] = 1 - 0.80169 = 0.19831$$

Primer 1

Korak 4: Uteži so popravljene/prilagojene z enačbami:

$$\Delta w_j = -\eta(d-o)f'(a)I_j \qquad j = 0,1,2,\ldots,n$$
$$w_j(new) = w_j(old) + \Delta w_j \qquad j = 0,1,2,\ldots,n$$

$$W_{k00}(new) = W_{k00}(old) + n * \Delta k[0] * f(k[0]) * j[0] =$$
$$0.5 + 1 * (-0.88066) + 0.2072 * 0.7109 = 0.3694$$
$$W_{k01}(new) = 0.56309 \qquad W_{k10}(new) = 0.6301 \qquad W_{k11}(new) = 0.5138$$
$$W_{j00}(new) = W_{j00}(old) + n * I[0] * \Sigma w\Delta =$$
$$0.1 + 1 * 1 * (0.5 * -0.88066 + 0.6 * 0.19831) = -0.2213$$
$$W_{j01}(new) = 0.6 \qquad W_{j02}(new) = 0.4787 \qquad W_{j10}(new) = -0.3173$$
$$W_{j11}(new) = 0.3 \qquad W_{j12}(new) = 0.3827$$

Primer 1

Iteracija št. 2: v tej iteraciji so za izračun uporabljene popravljene uteži iz prejšnje iteracije. Ponovijo se koraki 2 – 4 iz prejšnje iteracije

Korak 2:

$$J[0] = 0.5640 \qquad J[1] = 0.5163 \qquad O[0] = 0.4991 \qquad O[1] = 0.6299$$

Korak 3:

$$\Delta k[0] = -0.4991 \qquad \Delta k[1] = 0.3701$$

Korak 4:

$$W_{k00}(new) = 0.3032 \quad W_{k01}(new) = 0.5025 \quad W_{k10}(new) = 0.6774$$
$$W_{k11}(new) = 0.5751 \quad W_{j00}(new) = -0.17248 \quad W_{j01}(new) = 0.6$$
$$W_{j02}(new) = 0.5275 \quad W_{j10}(new) = -0.4015 \quad W_{j11}(new) = 0.3$$
$$W_{j12}(new) = 0.2985$$

Uteži imajo po dveh iteracijah učenja naslendnje vrednosti:

$$J[0] = 0.5878 \quad J[1] = 0.5257 \quad O[0] = 0.4424 \quad O[1] = 0.7005$$

Slabosti mrež z vzvratnim razširjanjem

Učenje je pogosto dolgotrajen postopek

– Kompleksne funkcije zahtevajo več sto ali tisoč iteracij

Mreža je dejansko "črna škatla"

– med vhodnimi in izhodnimi vektorji lahko ustvari želeno preslikavo (***x, y***), vendar pa nima informacije, zakaj je določen ***x*** preslikan v določen ***y***.

– zato ne more ponuditi intuitivne (npr. kavzalne) razlage rezultata.

– vzrok je v tem, da skriti sloji in naučene uteži nimajo semantike. Naučiti je mogoče izvedbene parametre in ne splošnega, abstraktnega znanja nekega področja.

Pristop gradientnega zmanjševanja lahko zagotovi le zmanjšanje napake v lokalni minimum.

(***E*** se lahko zmanjša na 0)

– iz lokalnega minimuma napake ni mogoče pobegniti – vsaka funkcija se ne da naučiti.

Slabosti mrež z vzvratnim razširjanjem

Nevarnost lokalnega minimuma je odvisna od oblike funkcije napake: veliko dolin in oz. lokalnih minimumov seveda povečuje nevarnost ujetja.

Mogoče rešitve :
– Uporaba mrež z različnim številom skritih slojev in nevronov, ki lahko povzročajo različne oblike funkcij napake.
– Uporaba različnih začetnih uteži in s tem različnih začetnih vrednosti v funkciji napake. Prisilni izstop iz lokalnega minimuma z naključno perturbacijo.

- Posplošitev ni zagotovljena tudi, če se napaka zmanjša na nič

– Problem preučenja: naučena mreža izvede nad učnimi podatki popolno preslikavo z napako 0, vendar pa za druge vhodne podatke ne vrne točnega rezultata.

- Za razliko od statističnih metod, za učenje z vzvratnim razširjanjem ne obstaja predpisan način za oceno kvalitete

– Kakšen je nivo zaupanja za mrežo naučeno z vzvratnim učenjem in s končno napako E (ki je lahko, ali pa tudi ne, blizu nič)

Mreže z radialnimi baznimi aktivacijskimi funkcijami

Mreže z radialnimi baznimi aktivacijskimi funkcijami (ang. *Radial Basis Function Networks* – RBFN) so alternativa mrežam z vzvratnim razširjanjem. Izvajajo aproksimacije za probleme prilagajanje krivulj in jih je mogoče hitro in enostavno učiti.
Ena od prednosti RBFN je, da lahko linearne uteži povezane z izhodnim slojem obravnavamo ločeno od nevronov skritih slojev. Uteži izhodnega sloja se prilagajajo z linearno optimizacijo, medtem ko se uteži skritih slojev prilagajajo z nelinearno optimizacijo.

Osnovna ideja RBFN je Cover-jev teorem.

Coverjev teorem: Za kompleksen problem razvrščanja vzorcev je bolj verjetno, da bo linearno ločljiv v več-dimenzionalnem prostoru kot v prostoru nižjih dimenzij.

Uvod v radialne bazne funkcije

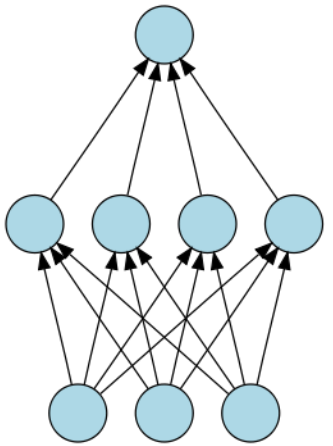
Ideja ***mrež z radialnimi baznimi funkcijami (ang. Radial Basis Function – RBF)*** izvira iz teorije funkcijske aproksimacije. Ogledali smo si že delovanje mrež z več-slojnimi perceptroni (Multi-Layer Perceptron – MLP) in s skritim slojem sigmoidnih enot, ki se lahko naučijo aproksimiranja funkcij. Mreže RBF uporabljajo nekoliko drugačen pristop. Njihove glavne značilnosti so:

1. So dvo-slojne mreže brez povratne vezave.
2. V skritem sloju so implementirane radialne bazne funkcije (npr. Gaussove funkcije)
3. V izhodnem sloju so implementirane linearne seštevalne funkcije kot v MLP.
4. Učenje mreže je razdeljeno v dve stopnji: najprej se določijo uteži od vhoda proti skritemu sloju in potem uteži od skritega proti izhodnemu sloju.
5. Učenje je zelo hitro.
6. Mreže so zelo dobre za interpolacijske probleme.

Mreže z radialnimi baznimi aktivacijskimi funkcijami

Mreže z radialnimi baznimi funkcijami (RBF) imajo običajno 3 sloje:
vhodni sloj, skriti sloj z nelinearno RBF aktivacijsko funkcijo
In izhodni linearni sloj. Izhodni sloj mreže je torej

$$\varphi(\mathbf{x}) = \sum_{i=1}^{N} a_i \rho(||\mathbf{x} - \mathbf{c}_i||)$$



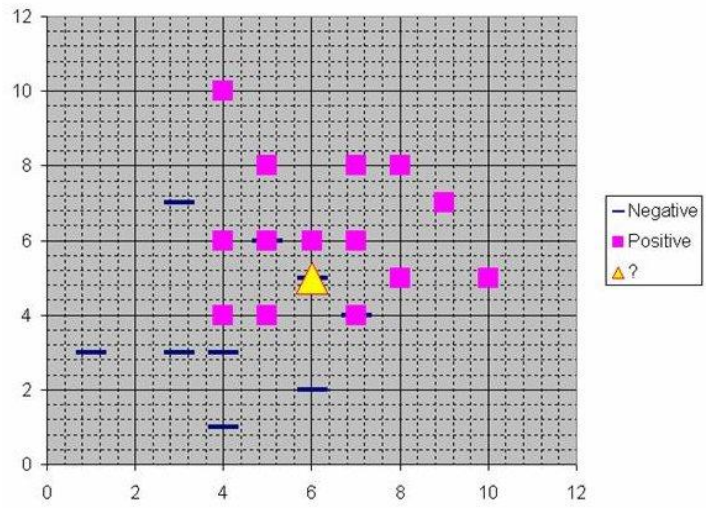
Kjer je *N* število nevronov v skritem sloju, C_i je centralni vektor nevrona *i* in α_i so uteži linearnega izhodnega nevrona. V osnovni obliki so vsi vhodni nevroni povezani z vsakim skritim nevronom. Običajno je uporabljena Evklidova norma (razdalja), čeprav je boljša Mahalanobisova razdalja, ter Gaussova bazna funkcija

$$\rho\left(\|\mathbf{x} - \mathbf{c}_i\|\right) = \exp\left[-\beta \|\mathbf{x} - \mathbf{c}_i\|^2\right]$$

Gaussova funkcija je lokalna, za njo velja $\lim_{\|x\| \to \infty} \rho(\|\mathbf{x} - \mathbf{c}_i\|) = 0$
Kar pomeni, da spreminjanje parametrov enega nevrona vpliva samo v manjši meri na vhodne vrednosti, ki so daleč od centra tega nevrona. Mreže RBF so univerzalni aproksimatorji. To pomeni, da mreža RBF z dovolj skritimi nevroni lahko aproksimira katero koi zvezno funkcijo s poljubno natančnostjo.
Uteži α_i, c_i, β , so določene na način, ki optimizira prilagajanje med φ in med podatki

Delovanje mrež RBF

Čeprav je implementacija zahtevna, so mreže RBF konceptualno podobne modelom K-Nearest Neighbor (k-NN). Osnovna ideja je, da je napovedana ciljna vrednost približno enaka vrednostim drugih elementov s približno enakimi vrednostmi kot prediktorji. Vzemimo za primer sliko:

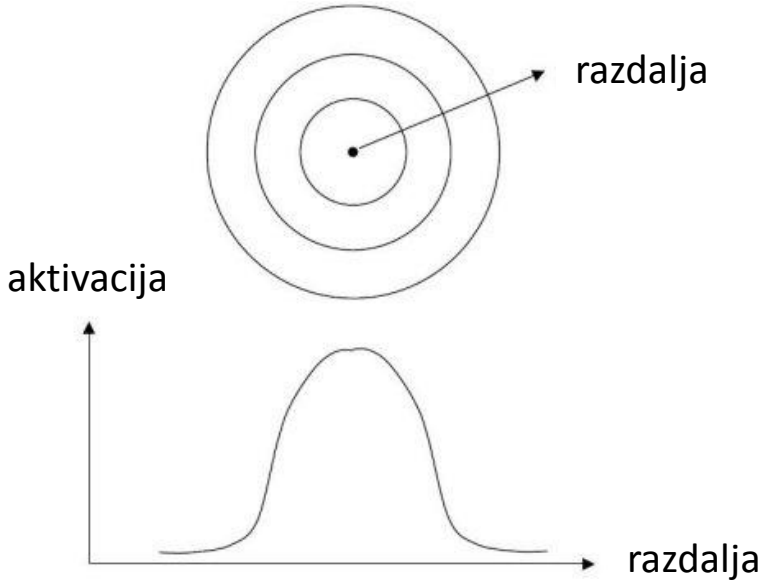


Privzemimo, da ima vsak primer v učni množici dve prediktivni spremenljivki, x in y. Primeri na sliki so narisani z uporabo njihovih x,y koordinat. Podobno privzemimo, da ima ciljna spremenljivka dve kategoriji: pozitivno, označeno s kvadratkom in negativno, označeno s črtico. Sedaj skušajmo napovedati vrednost novega primera, predstavljenega s trikotnikom s prediktivnimi vrednostmi x=6, y=5.1. Ali naj napovemo pozitivno ali negativno vrednost?

Položaj trikotnika se skoraj povsem prekriva črtico, ki predstavlja negativno vrednost. Vendar pa je njen položaj precej nenavaden v preimerjavi z ostalimi črticami, ki so zgoščene pod kvadrati v levo od sredine. Tako je možno, da je negativna vrednost v tem primeru napaka ali nenavadnost.

Delovanje mrež RBF

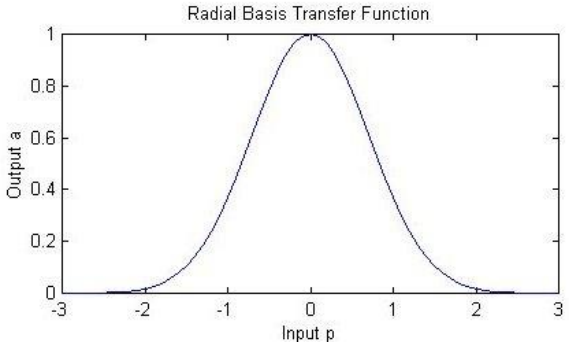
Klasifikacija najbližjih sosedov je v tem primeru odvisna od tega, koliko sosednjih točk upoštevamo. Če je uporabljen 1-NN in se upošteva samo ena najbližja točka, potem mora biti nova točka razvrščena kot negativna, ker leži nad poznano negativno točko. Nasprotno, če je uporabljena 9-NN klasifikacija in se upošteva 9 najbližjih točk, potem bo nad najbližjo negativno točko prevladal učinek ostalih 8 pozitivnih točk v okolici.
Mreža RBF postavi enega ali več RBF nevronov v prostor, ki ga opisujejo prediktivne spremenljivke (x,y v tem primeru). Ta prostor ima toliko dimenzij kot je prediktivnih spremenljivk. Evklidska razdalja se izračuna med točko, ki jo ocenjujemo (npr. trikotnik na prejšnji sliki), in centrom vsakega nevrona. Z uporabo RBF in Evklidove razdalje nato izračunamo utež (vpliv) vsakega nevrona. Od tod tudi izvor imena RBF: radialna razdalja je namreč argument funkcije.
Utež = RBF(razdalja)
Bolj kot je nevron oddaljen od točke, ki jo obravnavamo, manjši je njegov vpliv.



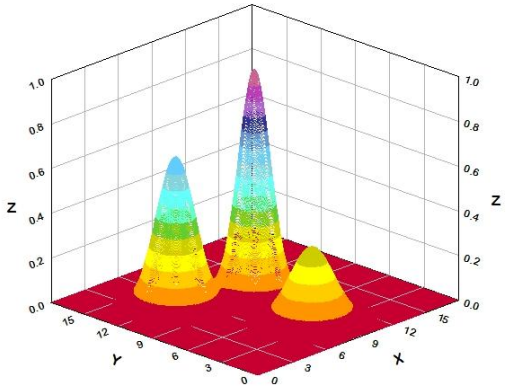
Delovanje mrež RBF

Radialna bazna funkcija (RBF)

Upoabljajo se lahko različni tipi RBF, najpogosteje pa se uporablja Gaussova funkcija:

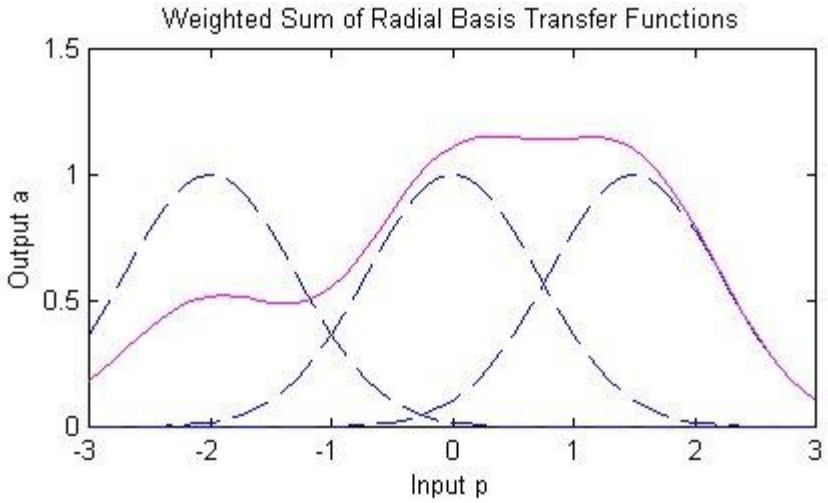


Če imamo več kot eno prediktivno spremenljivko, potem ima RBF toliko dimenzij, kot je spremenljivk. Spodnja slika predstavlja tri nevrone v prostoru dveh prediktivnih spremenljivk X in Y. Dimenzija Z predstavlja vrednost RBF funkcije.

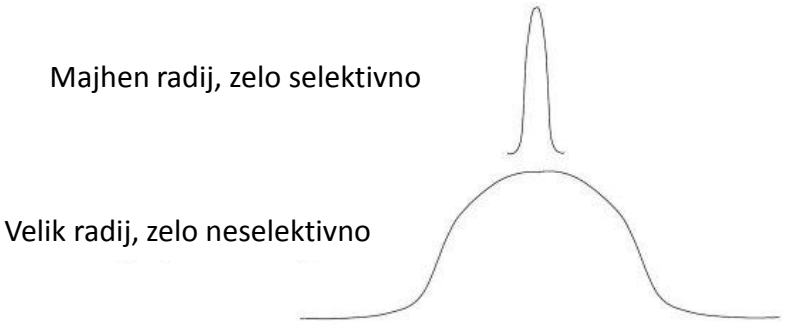


Najboljša prediktivna vrednost za novo točko je rezultat vsote izhodnih vrednosti RBF funkcij pomnožene z utežmi nevronov.

Delovanje mrež RBF

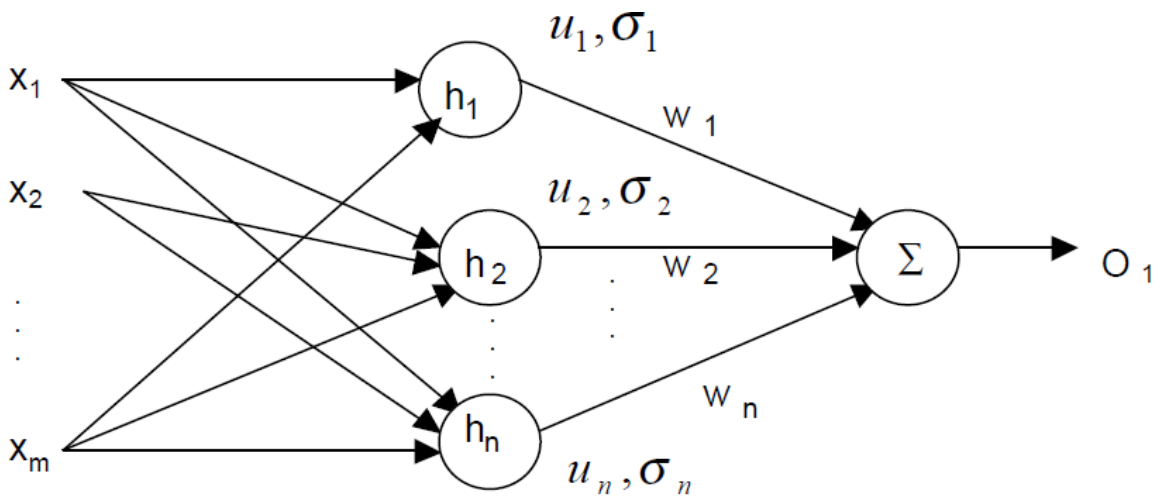


RBF nevrona ima središče in radij. Radij je lahko različen za vsak nevron. Pri mrežah RBF ustvarjenih DTREG ima vsaka dimenzija različen radij.



Nevroni z večjim radijem imajo na večjih razdaljah večji vpliv.

Preprosta RBFN mreža



XOR problem v prostoru h_1, h_2 je preslikan na nov problem, ki je linearno ločljiv. Gaussove funkcije lahko torej uporabimo za rešitev omenjenega problema interpolacije z enonivojsko mrežo. Zgornjo interpolacijo lahko posplošimo: *Vzemimo N točk (X_1,…,X_N) in pripadajočo zbirko N realnih vrednosti (d_1, d_2, d_3, ., d_1); poiščimo funkcijo, ki zadostuje naslednjemu interpolacijskemu pogoju:*

$$F(x_i) = d_i \qquad i = 1,2,\ldots, N$$

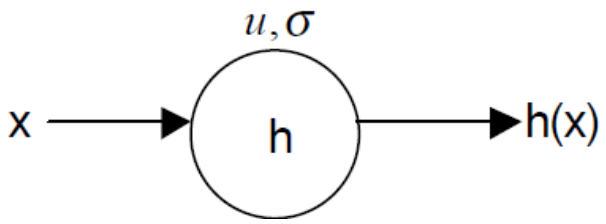
Razdalja

Vrednosti uteži vsakega nevrona v skritem sloju so izbrane tako, da proizvajajo želeni odziv: mreža ima maksimalen odziv takrat, ko so vhodi enaki njenim utežem. Aktivacijska funkcija h_i je definirana kot

$$h_i = e^{-D_i^2 / 2\sigma^2}$$

Kjer je *Di* razdalja med vhodom in središčem nevrona, določenim z vektorjem uteži skritega nevrona *i*.

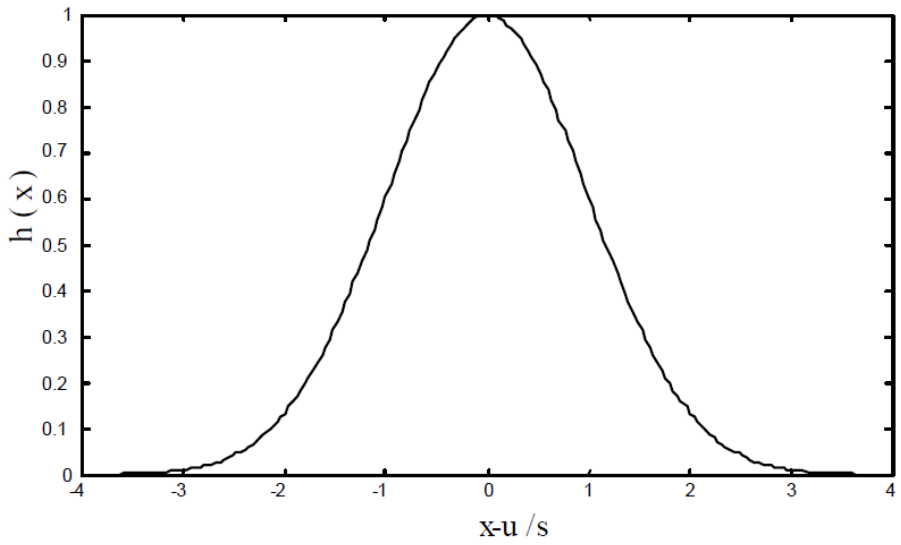
$$\begin{cases} D_i^{\,2} = (x - u_i)^T (x - u_i) \\ x : \text{Vhodni vektor} \\ u_i : \text{Vektor uteži skritega nevrona i} \end{cases}$$



Funkcija h(x)

Definicija funkcije h(x) :

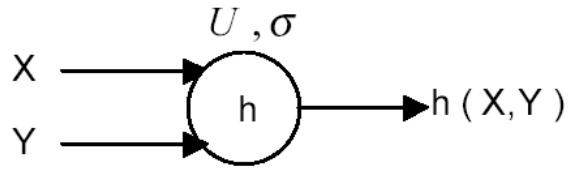
$$h(x) = e^{-\frac{(x-u)^2}{2\sigma^2}}$$



Preprost RBFN nevron

$$\begin{cases} h(x)=1 & x=u \\ h(x)=0 & |x-u|>3\sigma \\ 0<h(x)<1 & |x-u|<3\sigma \end{cases}$$

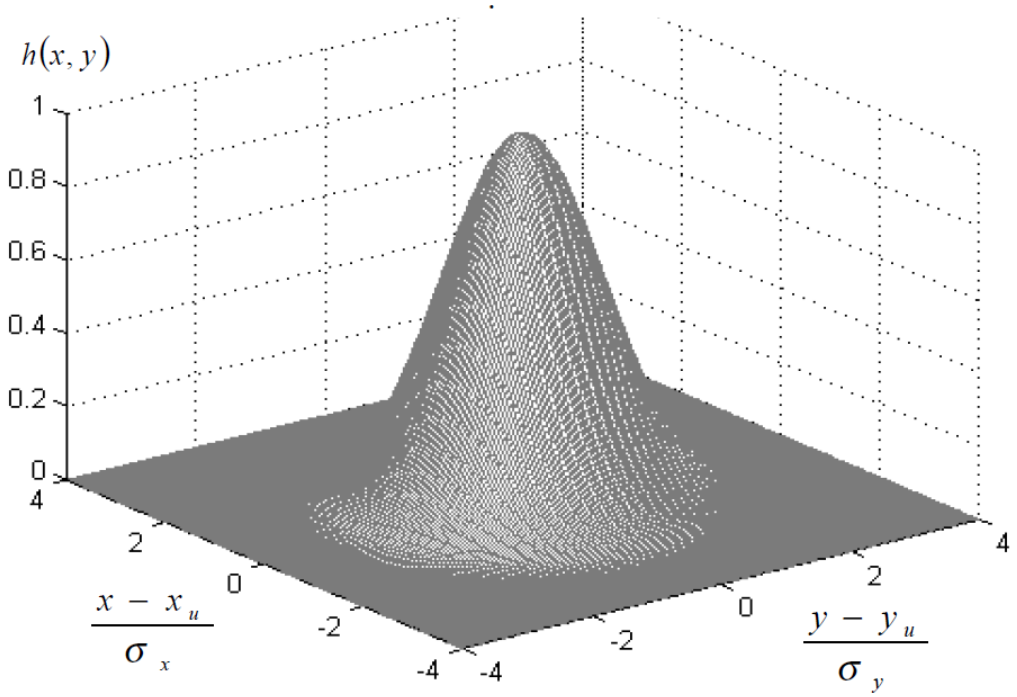
Enačba nakazuje, da vsebuje vsak nevron prispevke vhodov, ki so blizu središču utežne funkcije. Za druge vrednosti x, bo imel nevron ničelni odziv in ne bo prispeval h končnemu odzivu mreže. Na sliki je prikazan RBFN nevron z dvema vhodoma X in Y.



Učenje

Učenje RBFN mreže poteka v dveh fazah. V prvi fazi se določijo središče U_i in premer dovzetnosti σ_i za vsak nevron. V drugi fazi se ustrezno prilagodi vektor uteži W.

Po končanem učenju nastopi faza spomina, v kateri se določijo dejanski izhodi mreže.



Določanje središča U_i in premera σ_i nevronov

Določanje središča U_i: Eden od možnih pristopov določanja središča Ui je delitev vhodnega vektorja v gruče, ki jim nato določimo središča in lokacijo nevrona skritega sloja v tej točki.

Določanje premera dovzetnega območja: Vrednost σ lahko pomembno vpliva na zmogljivost mreže. Ena od priljubljenih metod je osnovana na podobnosti grozdenja vhodnih podatkov. Za vsak nevron skritega sloja se izračuna RMS razdalja do njegovih najbližjih sosedov, ki predstavljajo vrednosti σ. Učno fazo RBFN mreže lahko povzamemo:

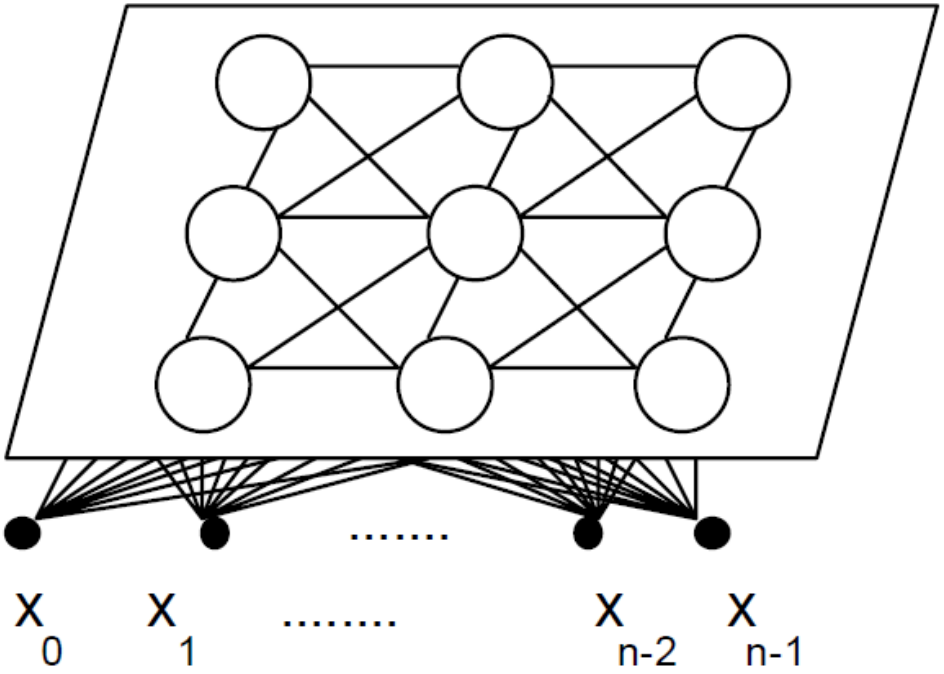
1. Vzemimo vhodni vektor X učne množice.
2. Izračunamo izhod skritega sloja.
3. Izračunani izhod Y primerjamo z želeno/ciljno vrednostjo in ustrezno popravimo uteži W:

$$w_{ij}(n+1) = w_{ij}(n) + \eta.(x_j - y_j).x_i$$

4. Ponovimo korake 1 – 3 za vsak vektor učne množice.
5. Ponovimo korake 1 – 4 dokler ni napaka manjša od določene največje sprejemljive vrednosti.

Prednost mreže z radialnimi osnovnimi aktivacijskimi funkcijami v primerjavi z mrežami z vzvratnim razširjanjem je hitrejše učenje.

KOHONENOVA SAMOORGANIZIRAJOČA SE MREŽA (SOM)



KOHONENOVA SAMOORGANIZIRAJOČA SE MREŽA

Kohononenov SOM je vrsta nenadzorovanega učenja.Cilj tega je odkriti nekakšno temeljno strukturo podatkov. Kohonenova samoorganizirajoča se mreža se organizira glede na topološke lastnosti vhodnih podatkov.

Kohonen SOM je preslikava, ki ohranja topologijo, ker je topološka struktura naložena v vozliščih omrežja.Topološka karta je torej le preslikava, ki ohranja sosedske odnose. Vsako vozlišče v določenem sloju je enako v tem, da je vsako povezano z vsemi vozlišči v zgornji in / ali spodnji plasti. Vozlišča, ki so "tesno", skupaj drugače sodelujejo kot vozlišča, ki so "daleč" narazen.

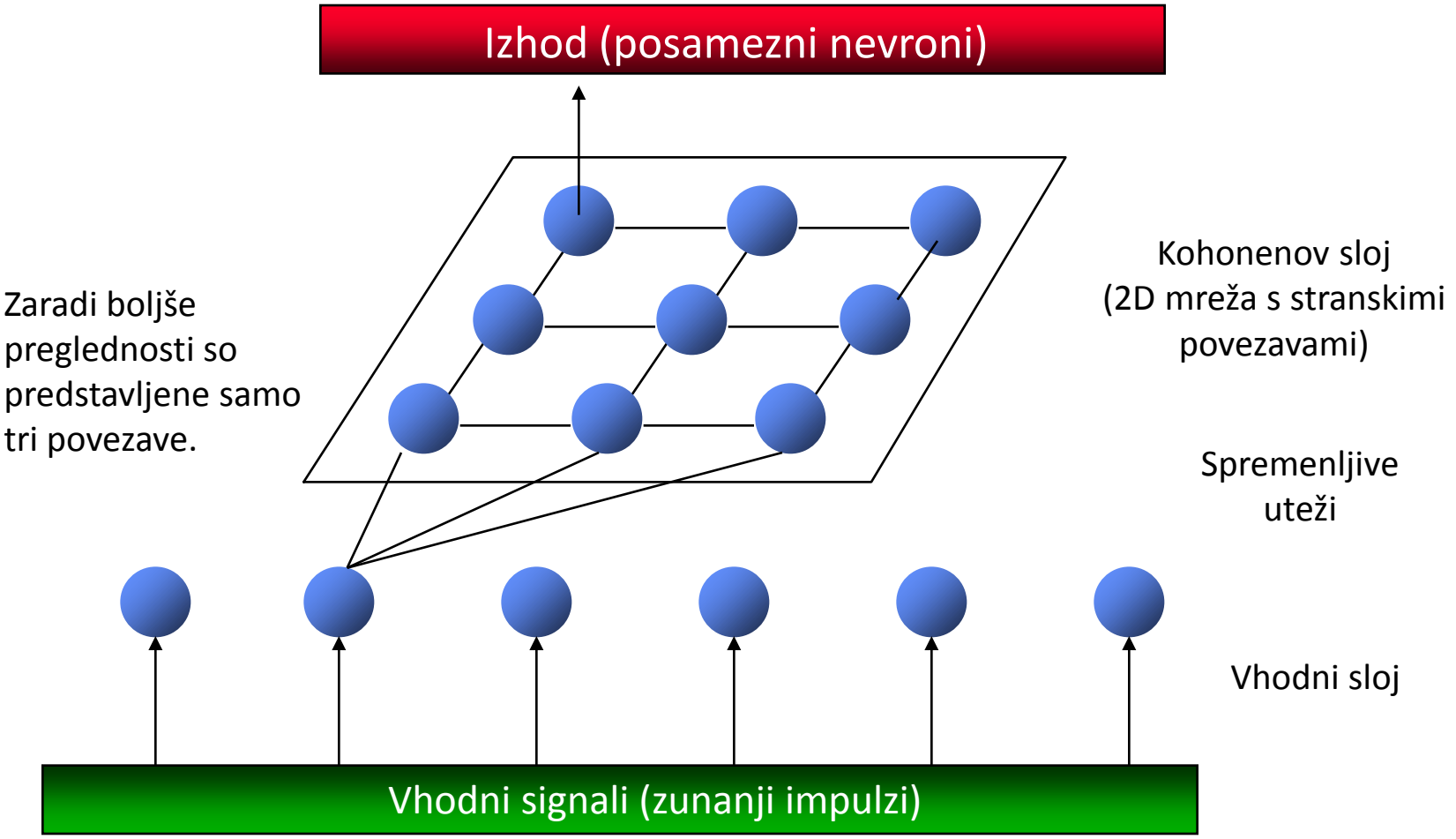
Kohonenova mreža ni hierarhični sistem, temveč je zgrajena iz polja popolnoma povezanih nevronov. Izhod nevrona predstavlja vhod vsem drugim nevronom v mreži vključno s samim sabo. Vsak nevron ima dve vrsti uteži:

- Prva je uporabljena za izračun vsote uteženih zunanjih vhodov,
- Druga pa je uporabljena za nadzor interakcij med nevroni v mreži.

Uteži vhodnega vzorca so spremenljive, medtem ko so uteži med nevroni fiksne.

KOHONENOVE SAMOORGANIZIRAJOČE SE PRESLIKAVE

- Nenadzorovano učenje
- Tesno povezano z grozdenjem
- Ciljni/želeni izhod ni potreben za vsak vhodni vektor učne množice
- Vhodi so povezani v dvodimenzionalno mrežo nevronov
  - Eksplicitno vzdrževanje sosedski odnosov ali
  - Vsak nevron lahko ima stranske povezave do svojih sosedov.
- Večdimenzionalni podatki se lahko preslikajo v dvodimenzionalno ravnino
  - Omogoča predstavitev gruč podatkov


Izhod (posamezni nevroni)
Zaradi boljše preglednosti so predstavljene samo tri povezave.
Kohonenov sloj (2D mreža s stranskimi povezavami)
Spremenljive uteži
Vhodni sloj
Vhodni signali (zunanji impulzi)

Naučena KOHONENOVA SOM

Nov vhodni vektor **X**

$\boldsymbol{X}=[x_1,x_2,....x_n]$

n elementov

Prehod skozi vhodne nevrone

Vektor uteži **w**,
Med vhodom
In vsakim
Kohonen nevronom

$\boldsymbol{W}_i=[w_{i1},w_{i2},...w_{in}]$

i je nevron
V Kohonen
sloju

Vsak nevron v Kohonen sloju vrne vrednost

Euclidova razdalja E_d, med nevronom in originalnim vektorjem

$E_d=|| X - W_i ||$

$$E_d=\sqrt{\sum_j (x_j - w_{ij})^2}$$

w_{ij} je utež med vhodom *j*
In Kohonen nevronom *i*

Učenje Kohonen SOM

Začne se z naključno inicializacijo uteži
med vhodom in Kohonen sloji

Obdelava učnih vektorjev

Določi se zmagovalni nevron

Določeni so tudi sosednji nevroni
zmagovalnega nevrona

Učenje Kohonen-ove SOM

Posodobljene so uteži zmagovalnega nevrona in njegovih sosedov tako, da se bolj približajo vhodnemu vektorju

Posodobitev uteži se izračuna:

$$\Delta w_{ij} = \alpha(x_j - w_{ij})$$

α – parameter stopnje učenja

Učenje Kohonen-ove SOM

Posodobljene so samo uteži povezav zmagovalnega nevrona in njegovih sosedov z naslednjim izračunom:

$$W_{ij}^{new} = W_{ij}^{old} + \Delta W_{ij}$$

Stopnja učenja in velikost soseščine se zmanjšujeta med učenjem.

Učenje Kohonen-ove SOM

Stopnja učenja je običajno nastavljena na relativno visoko vrednost, npr. 0.5 in se nato zmanjšuje po enačbi:

$$\alpha_t = \alpha_0 (1 - (t/T))$$

T – celotno število učnih iteracij

t – trenutno zaporedno število iteracije

α_t – stopnja učenja trenutne iteracije

α_0 – začetna vrednost stopnje učenja

Učenje Kohonen-ove SOM

Tudi velikost soseščine se iterativno zmanjšuje

Začetno vrednost nastavimo tako, da zajamemo približno polovico mreže

Velikost soseščine se v vsaki epohi zmanjša linearno

Učenje Kohonen-ove SOM

Kohonenov sloj z najmanjšo Evklidsko razdaljo,
tj. z nevronom najbližje vhodu,
je izbran na naslednji način:

$$|| X - W_c || = \min\{|| X_i - W_i ||\}$$

c označuje '**zmagovalni**' nevron v Kohonen sloju

'**Zmagovalni**' nevron je privzet
kot izhod mreže -"**zmagovalec pobere vse**"

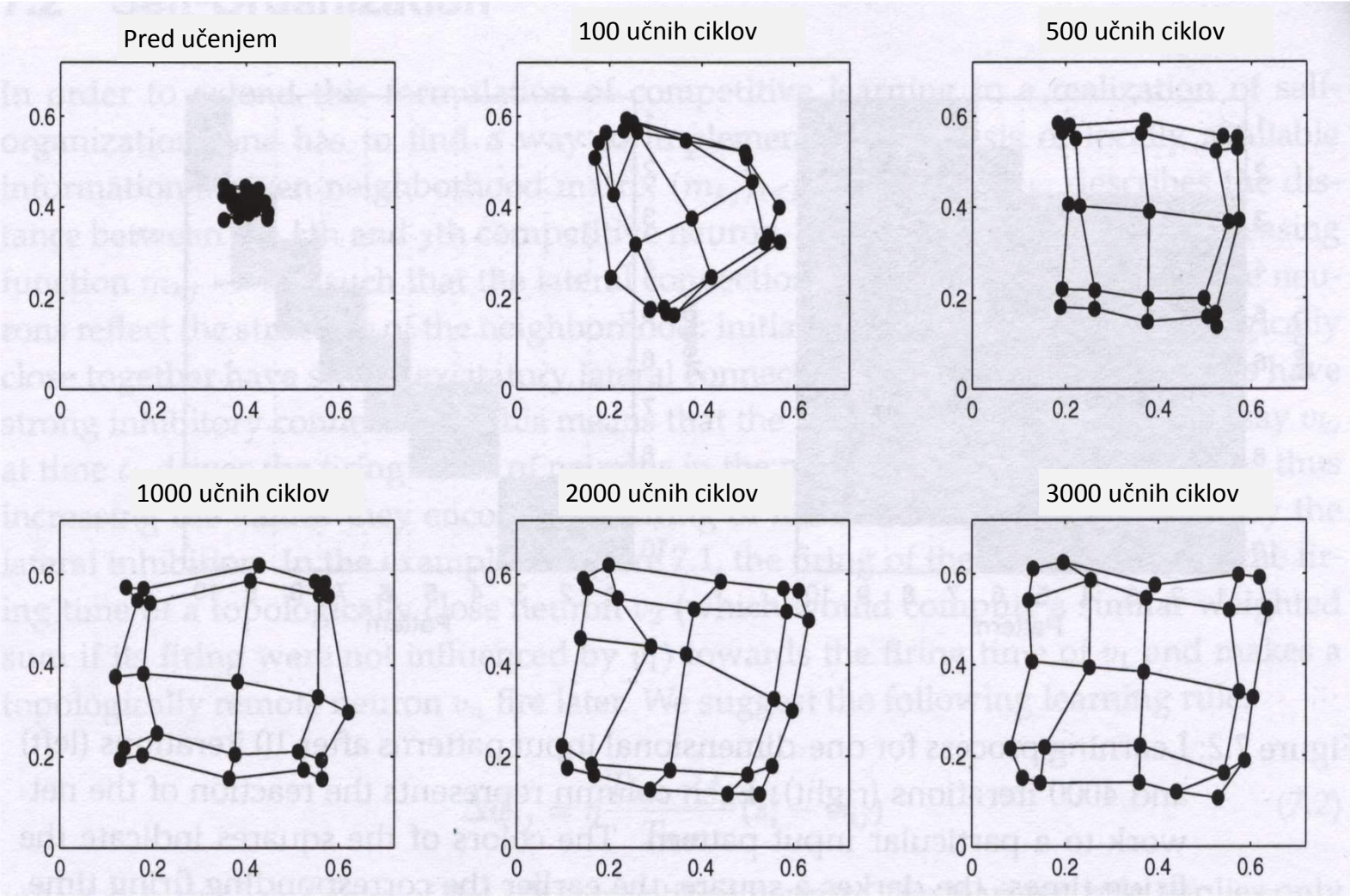
Prednosti Kohonen SOM

- Nenadzorovana arhitektura
- Ciljni izhodni vektorji niso potrebni
- Enostavna organizacija v najboljšo predstavitev za uporabljeno učno množico

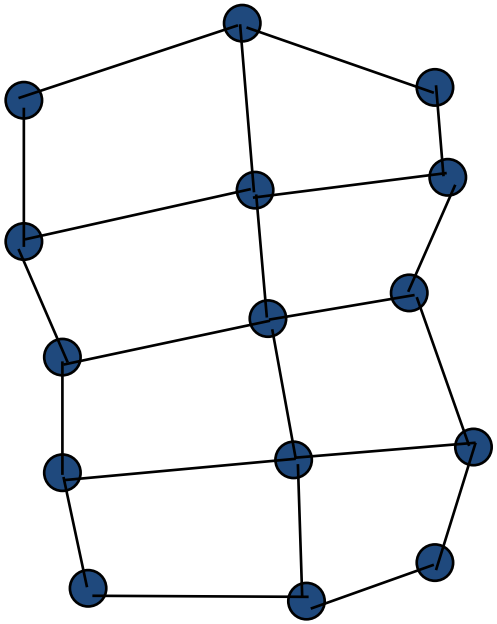
Omejitve Kohonen SOM

- Ponuja samo informacijo o lokaciji določenega vektorja v podatkovnem prostoru
- Zato je potrebna dodatna interpretacija te informacije
- Proces interpretacije je lahko dolgotrajen in zahteva podatke za katere je klasifikacija znana

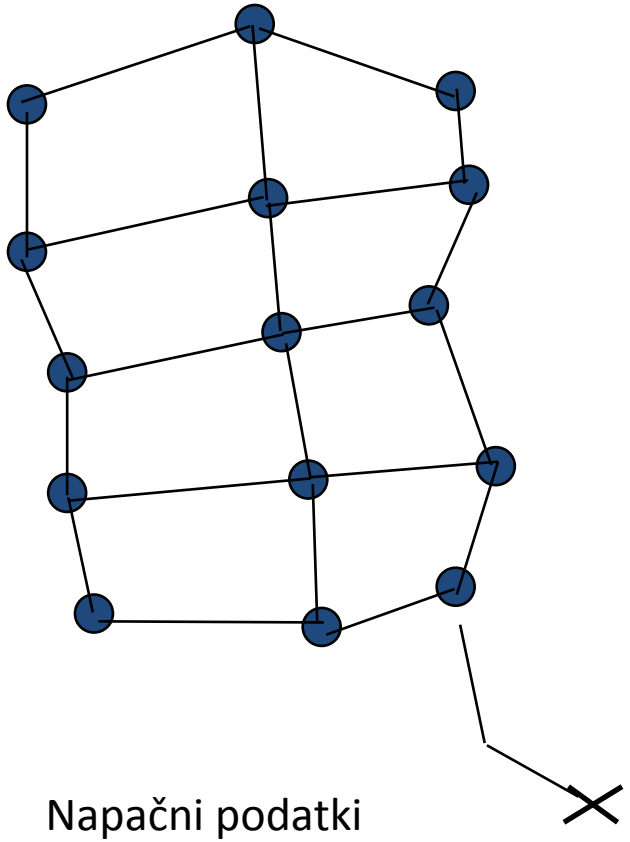
SOM za 2D kvadratno območje



Kohonen SOM – detekcija novosti



Kohonen mreža,
ki predstavlja
‘normalni’ prostor



Napačni podatki
padejo izven
‘normalnega’ prostora

Označevanje Kohonen mrež

- Oznake razredov so lahko uporabljene, če so označeni tudi podatki
- Uporaba strategij najbližji sosed (ang. nearest-neighbour) ali glasovanja
  - Najbližji sosed – Oznako razreda nastavimo na najpogostejšo oznako primera K najbližjih sosedov.
  - Glasovanje – določimo vse razrede, ki so pripisani nevronu in dodelimo najpogostejši razred.

Naučena vektorska kvantizacija

- Če so na voljo označeni podatki, potem lahko z vektorsko kvantizacijo izboljšamo porazdelitev nevronov.
- Nevrone premaknemo proti pravilno razvrščenim razredom.
- Oddaljevanje od nepravilnih klasifikacij.

- Nenadzorovano učenje zahteva podatke brez razrednih oznak
  - Odkrivanje gruč (in po možnosti tudi razrednih oznak)
  - Vizualizacija
  - Odkrivanje novosti

POVZETEK KOHONEN ALGORITMA

Algoritem poišče najbližji nevron, ki se ujema z učno množico in omogoča spreminjanje uteži samega nevrona in njegovih sosedov.

Naj bo w_{ij} utež med i-tim vhodnim nevronom in j-tim izhodnim nevronom.

Korak	Delovanje
0	Začetne vrednosti uteži. Največja vrednost naj bo R, določimo učno stopnjo α
1	Ponavljaj korake 2 – 8, dokler ni izpolnjen izstopni pogoj.
2	Za vsak vhodni vektor **x** ponovi korake 3 do 5
3	Za vsak j-ti nevron izračunaj Evkidsko razdaljo. $D(j) = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$
4	Poišči J tako da bo $D(J)$ minimum
5	Za vseh j nevronov v določeni soseščini J in za vse i $w_{ij}(new) = w_{ij}(old) + \alpha(x_i - w_{ij}(old))$ Opazimo, da je vektor uteži prisilen v pomikanje proti vhodnemu vektorju.
6	Posodobimo α, ki se zmanjšuje s številom epoh.
7	Zmanjšamo polmer topološke soseščine v vnaprej določenih trenutkh.
8	Izstopni pogoj. Običajno je to majhen del učne stopnje pri kateri so uteži nepomembne.

Kako deluje ?

- Naključno izberi vhodni vektor x
- Ugotovi "zmagovalno" izhodno vozlišče i, kjer je w_i utežni vektor, ki povezuje vhode z vsemi vozliščem i izhodnega nivoja.

Opomba: Enačba je enaka w_i . x>= x le, če so uteži normalizirane.

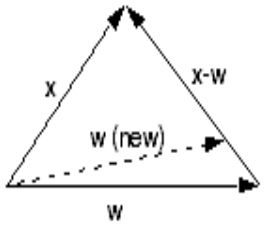
$$\left|\omega_i - x\right| \le \left|\omega_k - x\right| \qquad \forall k$$

Glede na zmagovalno vozlišče i, je posodobitev

$$w_k(new) = w_k(old) + \mu\, \aleph(i,k)\,(x - w_k)$$

kjer se $\aleph(i,k)$ imenuje sosedska funkcija, ki ima vrednost 1, če je i = k in pada z razdaljo | r_k - r_i | med vozli i in j v izhodni matriki (nivo). Tako se blizu vozla zmagovalca, kot tudi samega zmagovalca, njihove uteži znatno posodobijo. Uteži, povezane z oddaljenimi vozlišči izhodnega nivoja se bistveno ne spreminjajo. To je nek način pridobitve toploške informacije. Vse bližnje enote prejmejo podobne posodobitve in tako, da je na koncu odziv bližji vhodnemu vzorcu.

Zgornje pravilo potiska vektor uteži w_i uteži in bližnjih enot v smer vstopnega x.



KOHONEN SOM PRIMER

Vzemimo preprost primer, pri katerem imamo štiri učne vzorce.

x_1	x_2	x_3	x_4
1	1	0	0
0	0	0	1
1	0	0	0
0	0	1	1

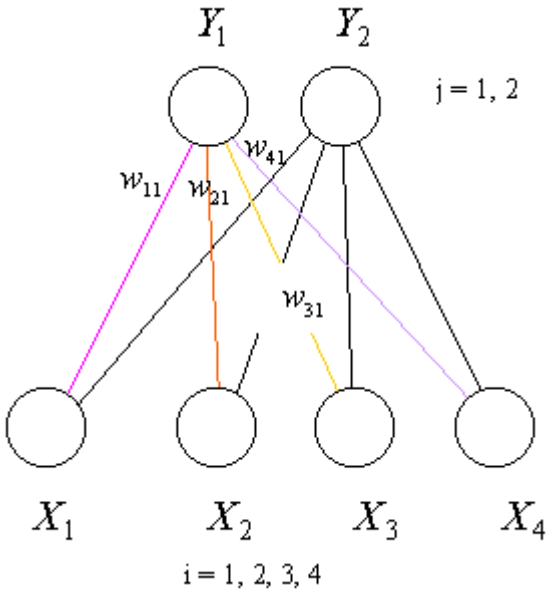
Naj bo stopnja učenja v trenutku t + 1 podana kot $\alpha(t+1) = \frac{\alpha(t)}{2}$

Začetna vrednost $\alpha(t=0) = 0.6$

Naj bo topološki polmer enak R = 0.

Za dodatno poenostavitev si oglejmo le dva izmed vseh nevronov v izhodnem sloju.

KOHONEN SOM primer nad.



Naj bo začetna matrika uteži

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

KOHONEN SOM primer nad.

Za vektor 1100

D(1) = 1.86, D(2) = 0.98 (Zaradi enostavnosti uporabimo kvadrat Evklidske razdalje.

Torej J = 2. Opazimo, da je R = 0, zato v tem primeru posodobitev uteži ni potrebna.

Z uporabo $w_{ij}(new) = w_{ij}(old) + \alpha\left(x_i - w_{ij}(old)\right)$, določimo novo matriko uteži

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$$

KOHONEN SOM primer nad.

Za vektor 0001
D(1) = 0.66, D(2) = 2.2768
Torej J = 1.

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}$$

Za vektor 1000
D(1) = 1.8656, D(2) = 0.6768
Torej J = 2

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.08 & 0.968 \\ 0.24 & 0.304 \\ 0.20 & 0.112 \\ 0.96 & 0.048 \end{bmatrix}$$

KOHONEN SOM primer nad.

Za vektor 0011

D(1) = 0.7056, D(2) = 2.724

Torej J = 1

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.032 & 0.968 \\ 0.096 & 0.304 \\ 0.680 & 0.112 \\ 0.984 & 0.048 \end{bmatrix}$$

Zmanjšamo stopnjo učenja (korak 6): $\alpha(1) = \frac{\alpha(0)}{2} = \frac{0.6}{2} = 0.3$

KOHONEN SOM primer nad.

Pokažemo lahko, da je po 100 obdelanih vhodnih vektorjev, končna matrika uteži enaka

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 6.7x10^{-17} & 1 \\ 2x10^{-16} & 0.49 \\ 0.51 & 2.3x10^{-16} \\ 1 & 1x10^{-16} \end{bmatrix}$$

Uporabimo lahko poenostavitev

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0.5 \\ 0.5 & 0 \\ 1 & 0 \end{bmatrix}$$

Razred 1 Razred 2

KOHONEN SOM primer nad.

TESTNA MREŽA

Vzemimo vhodni vzorec 1100.
Potem velja

$$D(j) = (w_{1j} - x_1)^2 + (w_{2j} - x_2)^2 + (w_{3j} - x_3)^2 + (w_{4j} - x_4)^2$$

$$D(1) = (0-1)^2 + (0-1)^2 + (0.5-0)^2 + (1-0)^2 = 3.25$$

$$D(2) = (1-1)^2 + (0.5-1)^2 + (0-0)^2 + (0-0)^2 = 0.25$$

Torej je nevron 2 “zmagovalec“ in je lokalno aktivno območje SOM. Opazimo, da lahko temu vhodnemu vzorcu dodelimo oznako razreda 2. Za vse ostale vzorce dobimo naslednje razrede.

x_1	x_2	x_3	x_4	Razred
1	1	0	0	2
0	0	0	1	1
1	0	0	0	2
0	0	1	1	1

Java Demo:
http://sund.de/netze/applets/som/som1/

Literatura

1. McCulloch, W.W. and Pitts, W., A Logical Calculus of Ideas Imminent in Nervous Activity. *Bull. Math. Biophys.*, 5, 115□133, 1943.
2. Pitts, W. and McCulloch, W.W., How we Know Universals, Bull. *Math.* 127□147, 1947.
3. McClelland, J.L. and Rumelhart, D.E., *Parallel Distributed Processing -Explorations in the Microstructure of Cognition*, Vol. 2, *Psychological and Biological Models*, MIT Press, Cambridge, MA, 1986.
4. Rosenblatt, F., *Principles of Neurodynamics*, Spartan Press, Washington, DC, 1961.
5 . Minsky, M. and Papert, S., *Perceptron*: *An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1969.
6 . Widrow, B. and Hoff, M.E, Adaptive Switching Circuits, IRE WESCON Convention Record, Part 4, NY, IRE, 96□104, 1960.
7 . Fausett, L., *Fundamentals of Neural Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
8. Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, 1999.