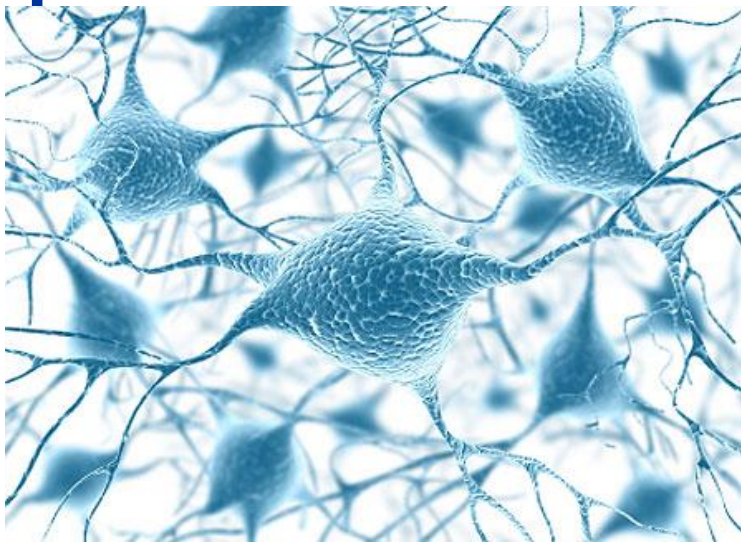
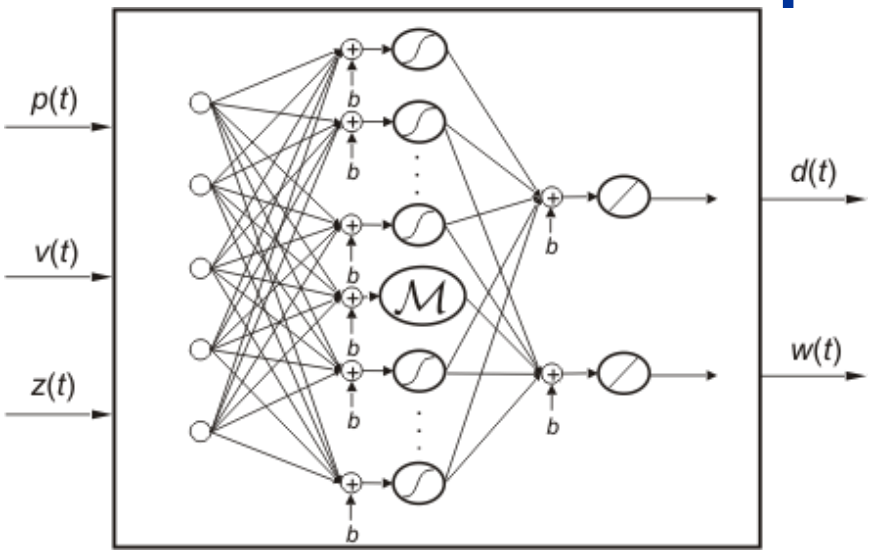
Neuronske mreže

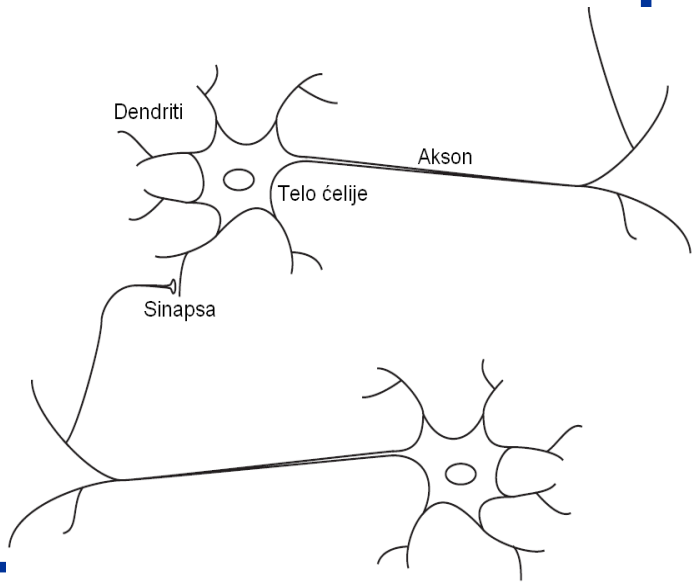
Miloš Mojović, v. prof.





Šta su neuronske mreže?

- Računarske neuronske mreže inspirisane su biološkim nervnim sistemom. Kao i u prirodi, veze između elemenata u velikoj meri određuju funkciju mreže. Neuronska mreža se može naučiti da obavlja određenu funkciju podešavanjem vrednosti veze (tzv. *težine*) između elemenata.
- U pravim biološkim sistemima, neke od neuronskih struktura sa nama su od rođenja, druge su pak, uspostavljene iskustvom. Opšte je prihvaćeno da se nervne funkcije, uključujući pamćenje, obavljaju u neuronima i vezama između njih. Učenje se posmatra kao uspostavljane novih veza i modifikacija starih.
- Neuroni koje ćemo ovde razmatrati nisu biološki. Oni su veoma prosta apstrakcija bioloških neurona, realizovani kao elementi u programu ili možda kao kola od silicijuma.
- Mreže ovakvih veštačkih neurona nemaju ni delić snage ljudskog mozga, ali mogu biti naučene da obavljaju razne korisne funkcije. Veštački neuron je zapravo računarski model inspirisan prirodnim neuronom.



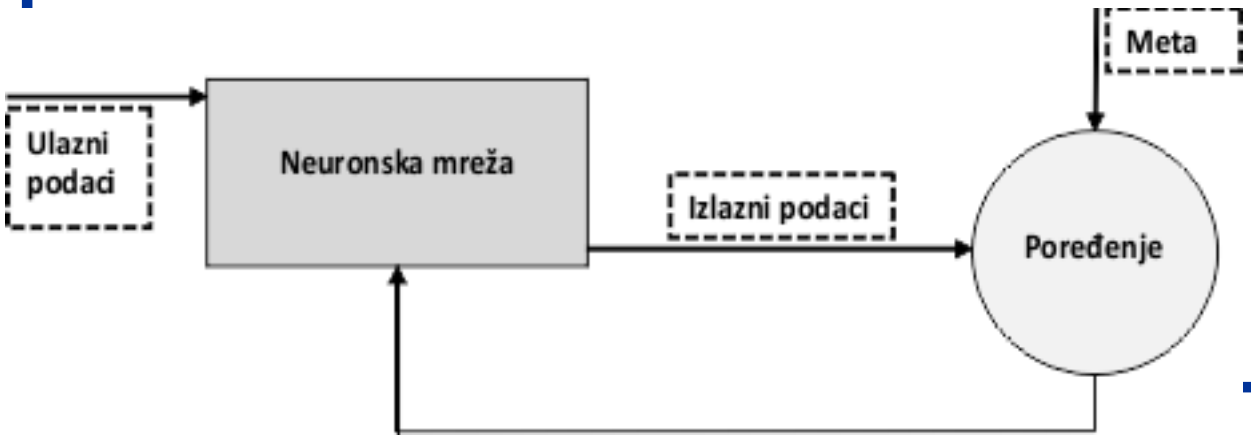
Šematski prikaz dva biološka neurona.

Opšti pregled neuronskih mreža

- Iako veštačke neuronske mreže ne prilaze kompleksnosti moždane strukture, postoje dve ključne sličnosti između bioloških i veštačkih neuronskih mreža.

  (1) Izgrađivački blokovi i jedne i druge mreže su prosti računarski uređaji (mada su veštački neuroni mnogo jednostavniji od bioloških) koji su izuzetno međusobno povezani.

  (2) Veze između neurona određuju funkciju mreže.
- Cilj koji se postavlja pred nama je da odredimo odgovarajuće veze da bi smo rešili određeni problem.
- Obično, neuronske mreže se usklađuju, ili treniraju, tako da određeni ulazni podaci vode do specifičnih izlaznih podataka – mete (očekivane vrednosti izlaznih podataka za date ulazne podatke).

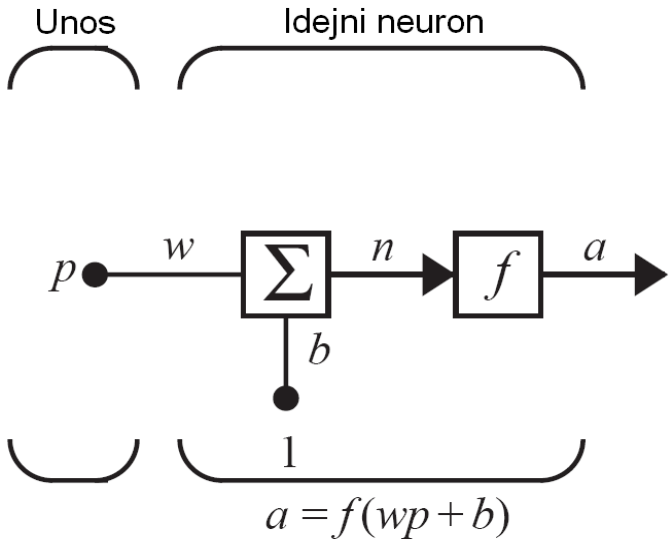


Treniranje mreže je zasnovano na poređenju izlaznih podataka i podataka mete.

Modeli neurona i mrežne strukture

Jednoulazni neuron

- Model jednoulaznog neurona izgrađen je iz više prostih delova od kojih je svaki prikazan i obeležen na slici.
- Skalarni *ulazni podatak* ili *unos p* množi se skalarnom *težinom w* (weight, eng. težina) da bi se formirao proizvod *wp* koji se dalje šalje na sumu.
- Dobijenom proizvodu *wp*, dodaje se *pomeraj b* (bias, eng. pomeraj) i prosleđuje se do sume.
- Sumiranjem dobijamo tzv. *mrežni unos n*, koji ide do *funkcije prenosa f* koja proizvodi skalarni *izlazni podatak* ili *iznos a* neurona.



Jednoulazni neuron

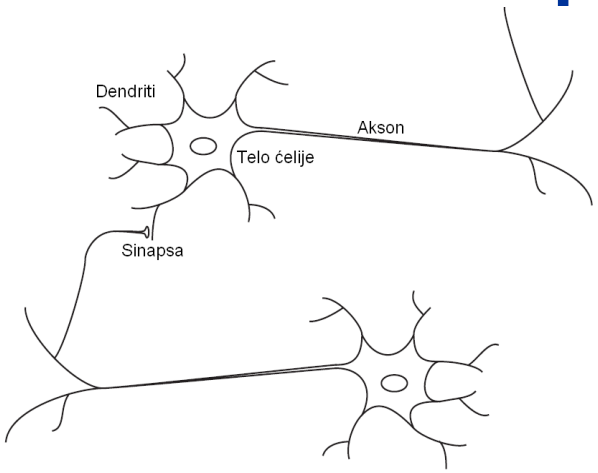
Modeli neurona i mrežne strukture

- Ako povežemo ovaj prosti model sa biološkim neuronom videćemo da težina *w* odgovara jačini sinapse, telo ćelije predstavljeno je sumom i funkcijom prenosa, a izlazni podatak *a* signalom na aksonu.
- Izlazni podatak računa se po formuli:

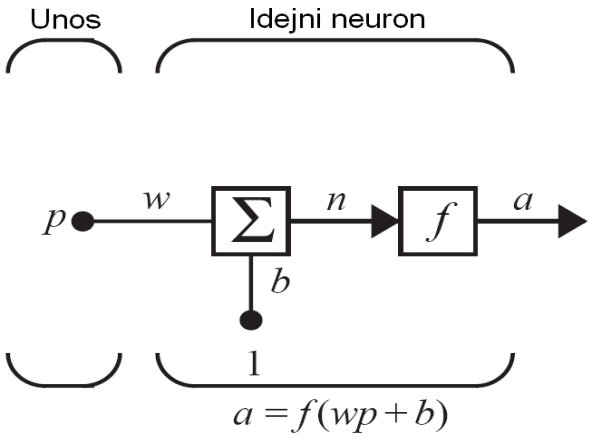
$$a = f(n) = f(wp + b)$$

(Ako je na primer: $w = 3$, $p = 2$ i $b = -1.5$, tada je $a = f(3 \cdot 2 - 1.5) = f(4.5)$

- Pomeraj *b* je vrlo sličan težini, osim što ima konstantan unos, 1. Međutim, ako ne želite da imate pomeraj u određenom neuronu, on može biti izostavljen.
- Treba imati na umu da su i *w* i *b* podesivi skalarni parametri neurona. Obično funkciju prenosa bira sam dizajner, a parametri *w* i *b* podešavaju se nekim *pravilom učenja* tako da odnos unos/iznos ispunjava neki specifičan cilj.
- Za različite svrhe koriste se različite prenosne funkcije.



Biološki neuroni



Jednoulazni neuron

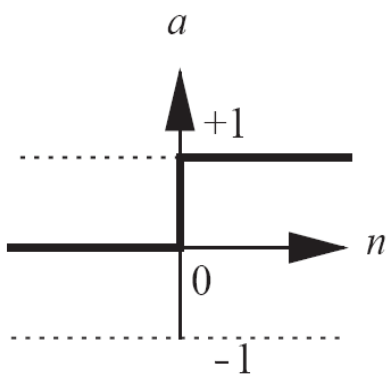
Prenosne funkcije

- Prenosna funkcija može biti linearna ili nelinearna funkcija od n. Određena prenosna funkcija bira se tako da udovolji zahtevima problema koji neuron pokušava da reši. Postoji mnoštvo ovih funkcija od kojih ćemo navesti dve.
- ***Hard limit*** prenosna funkcija (ili *funkcija praga*) prikazana na slici levo data je sa:

$$a = 0 \text{ za } n < 0,\ a = 1 \text{ za } n \geq 0$$

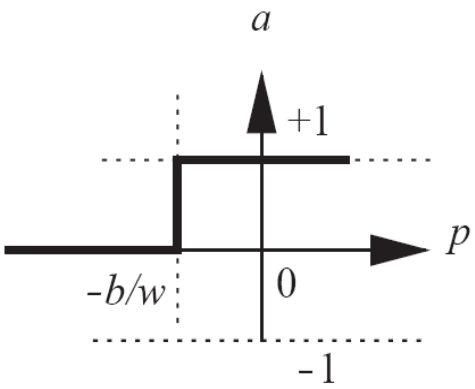
Ova funkcija za izlazni podatak neurona postavlja 0 ako je argument manji od 0, ili 1 ako je argument veći ili jednak 0. Ova funkcija se koristi kada želimo da kreiramo neuron koji klasifikuje ulazne podatke u dve odvojene kategorije.

- Grafik na desnoj strani ilustruje ulazno/izlazne karakteristike jednoulaznog neurona koji koristi hard limit prenosnu funkciju (može se primetiti uticaj težine i pomeraja).



$a = hardlim(n)$

Hard limit prenosna funkcija



$a = hardlim(wp + b)$

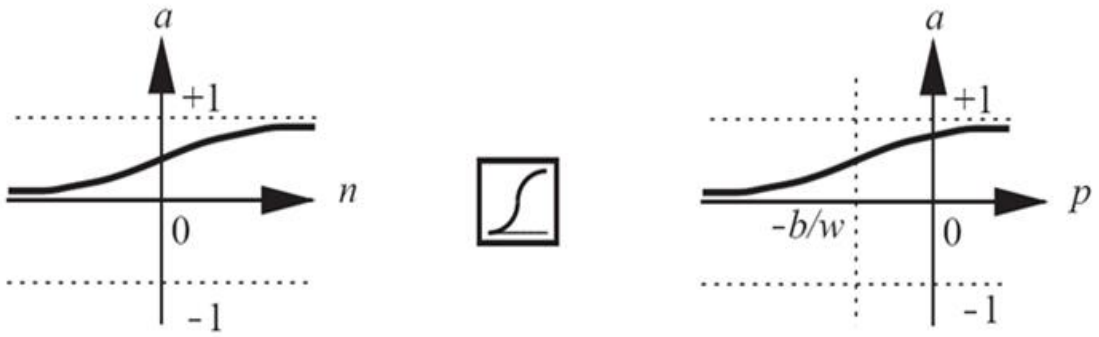
Jednoulazni *hardlim* Neuron

Prenosne funkcije

- ***Log-sigmoid*** (ili *sigmoidalna*) prenosna funkcija vrlo često je u upotrebi i data je sledećim izrazom:

$$a = \frac{1}{1 + e^{-n}}$$

- Ova funkcija svoj unos (koji može imati vrednosti između plus i minus beskonačno) pretvara u iznos sa vrednošću u intervalu od 0 do 1 (slika levo).
- Na desnoj strani slike opet se vidi kako odabir odgovarajućih vrednosti za težinu i pomeraj može uticati na vrednost izlaznog podatka *a*.
- Log-sigmoid prenosna funkcija najčešće se koristi u višeslojnim mrežama, a sa sobom nosi posebne pogodnosti zato što je diferencijabilna na čitavom svom domenu (od minus do plus beskonačno).



$a = logsig(n)$

Log-Sigmoid prenosna funkcija

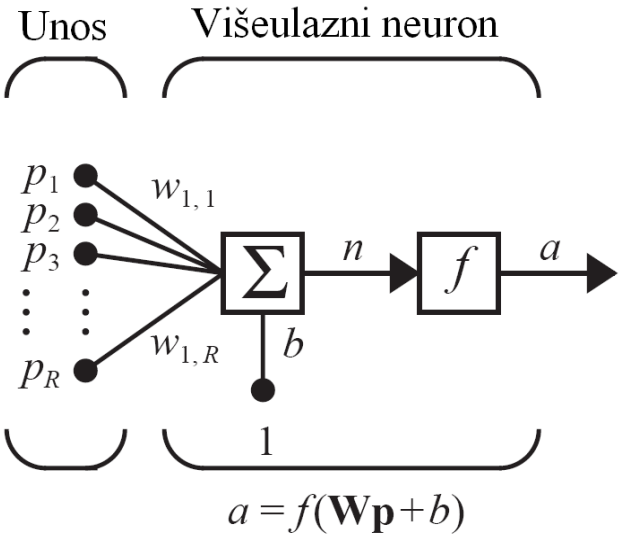
$a = logsig(wp + b)$

Jednoulazni *logsig* Neuron

Modeli neurona i mrežne strukture

Višeulazni neuron

- Neuron obično poseduje više od jednog unosa. Na slici je prikazan neuron sa R unosa.
- Pojedinačni ulazni podaci p_1, p_2,…, p_R umnoženi su odgovarajućim težinama koji predstavljaju elemente $w_{1,1}$, $w_{1,2}$,…, $w_{1,R}$ težinske matrice **W**.
- Neuron ima pomeraj b, koji se dodaje sumi unosa pomnoženih odgovarajućim težinama tako da formira mrežni unos n:



Višeulazni neuron

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

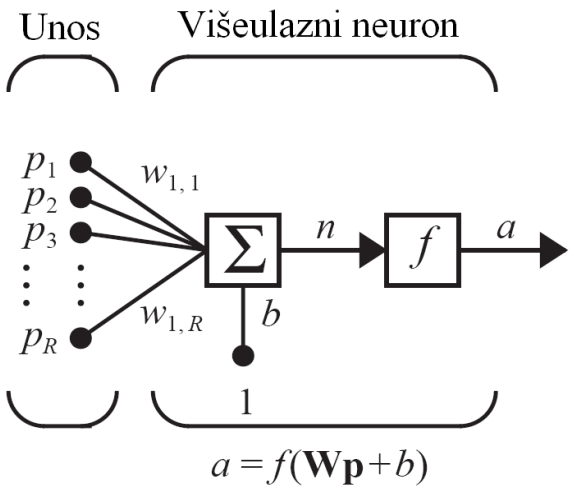
Ili u matričnoj formi: $n = \mathbf{W}\mathbf{p} + b$

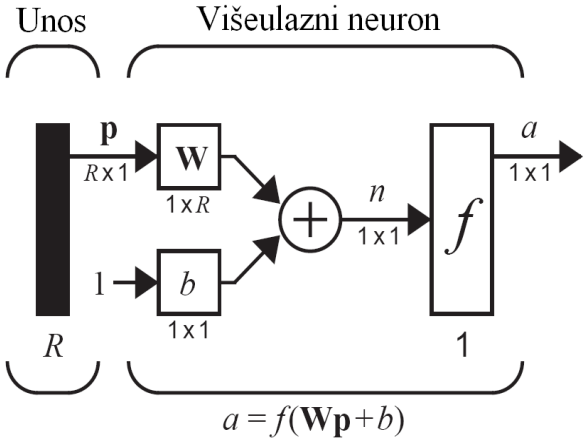
gde matrica **W** za slučaj jednog neurona ima samo jedan red. Sada iznos iz neurona može biti napisan kao:

$$a = f(\mathbf{W}\mathbf{p} + b)$$

Prikaz neuronske mreže – skraćena notacija

- Ukoliko želimo da nacrtamo mreže od po nekoliko slojeva pri čemu svaki ima nekoliko unosa to bi bilo veoma kompleksno i konfuzno. Da bi smo to izbegli, koristi se skraćena notacija (slika desno).
- Kao što se vidi, *ulazni vektor* **p** predstavljen je punom vertikalnom trakom na levoj strani. Dimenzije vektora **p** prikazane su ispod njegove oznake kao R x 1, ukazujući da se radi o matrici koloni sa R elemenata.
- Ulazni vektor **p** množi se sa težinskom matricom **W**, koja u slučaju jednog neurona, ima R kolona, ali samo jedan red. Konstanta 1 ulazi u neuron kao unos i biva umnožena skalarnim pomerajem *b*.
- Mrežni unos prenosne funkcije *f* je *n*, što predstavlja sumu pomeraja *b* i proizvoda **Wp**.
- U ovom slučaju, iznos *a* celokupnog neurona je skalar. Kada bi imali više od jednog neurona mrežni iznos bio bi vektor.
- Na šemama skraćene notacije uvek su uključene i dimenzije promenljivih koje se u njima pojavljuju, tako da odmah možemo prepoznati da li je reč o skalaru, vektoru ili matrici.



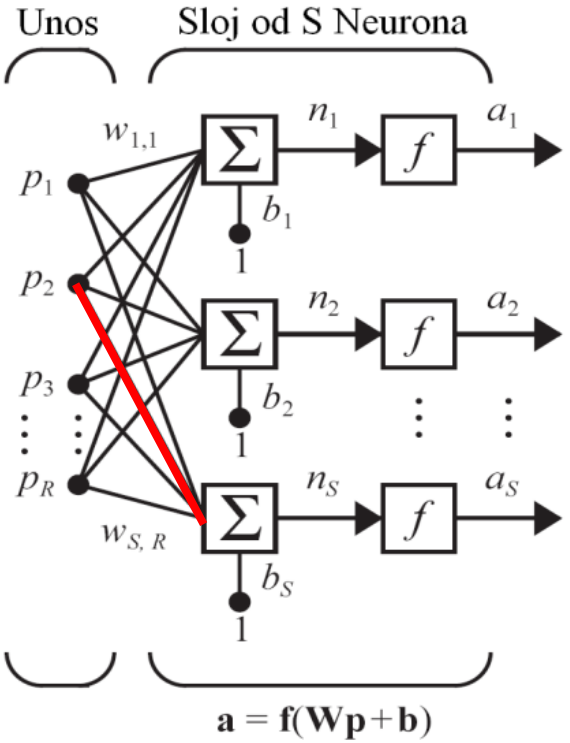


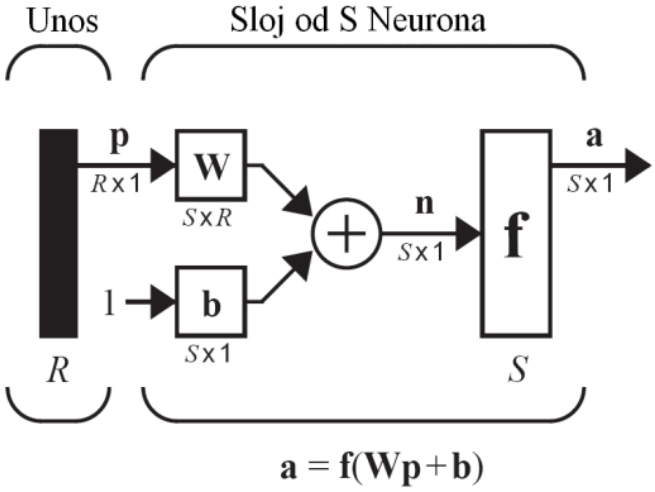
Neuron sa R unosa prikazan u skraćenoj notaciji

Mrežne strukture

Neuronski slojevi

- Jedan neuron, čak i sa više unosa, obično nije dovoljan. Zato koristimo više neurona koji rade paralelno u onome što se zove **“sloj”.**
- Jednoslojna mreža od S neurona data je na slici. Svaki od R unosa povezan je sa svakim od neurona, tako da težinska matrica **W** sada ima S redova.
- Sloj sadrži težinsku matricu, sume, vektor pomeraja **b**, prenosne funkcije i *izlazni vektor* **a**.
- Svaki element ulaznog vektora **p** povezan je sa svakim neuronom preko težinske matrice **W**. Svaki neuron poseduje pomeraj b_i, sumu, prenosnu funkciju f i iznos a_i. Svi iznosi zajedno formiraju izlazni vektor **a.**
- Broj unosa u sloj se obično razlikuje od broja neurona u sloju ($R \neq S$). Takođe, prenosne funkcije u sloju ne moraju biti iste za sve neurone. Može se definisati sloj koji će imati različite prenosne funkcije za različite neurone.





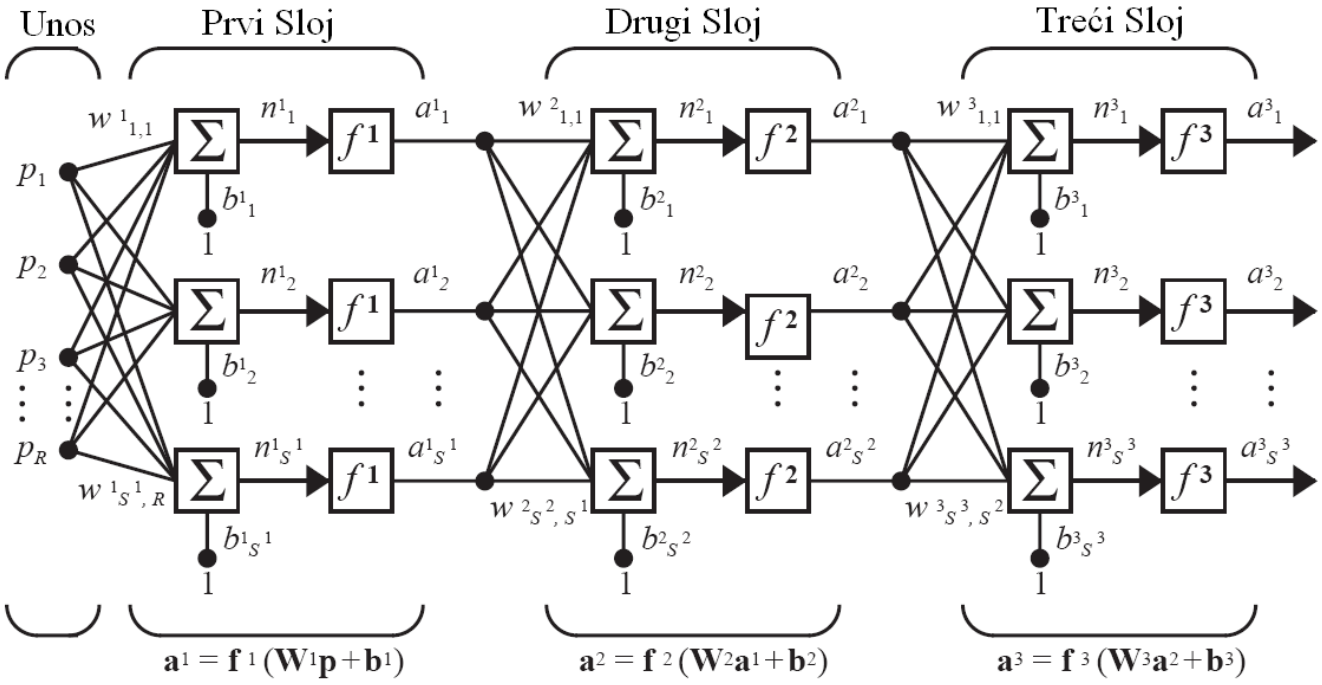
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

Elementi ulaznog vektora ulaze u mrežu preko težinske matrice **W**.

Indeks reda elementa matrice označava neuron kome taj element (tj. težina) pripada, a indeks kolone označava izvor signala za tu težinu. Tako, indeksi elementa $w_{3,2}$ govore nam da ova težina predstavlja vezu između trećeg neurona i drugog izvora (unosa).

Višestruki neuronski slojevi

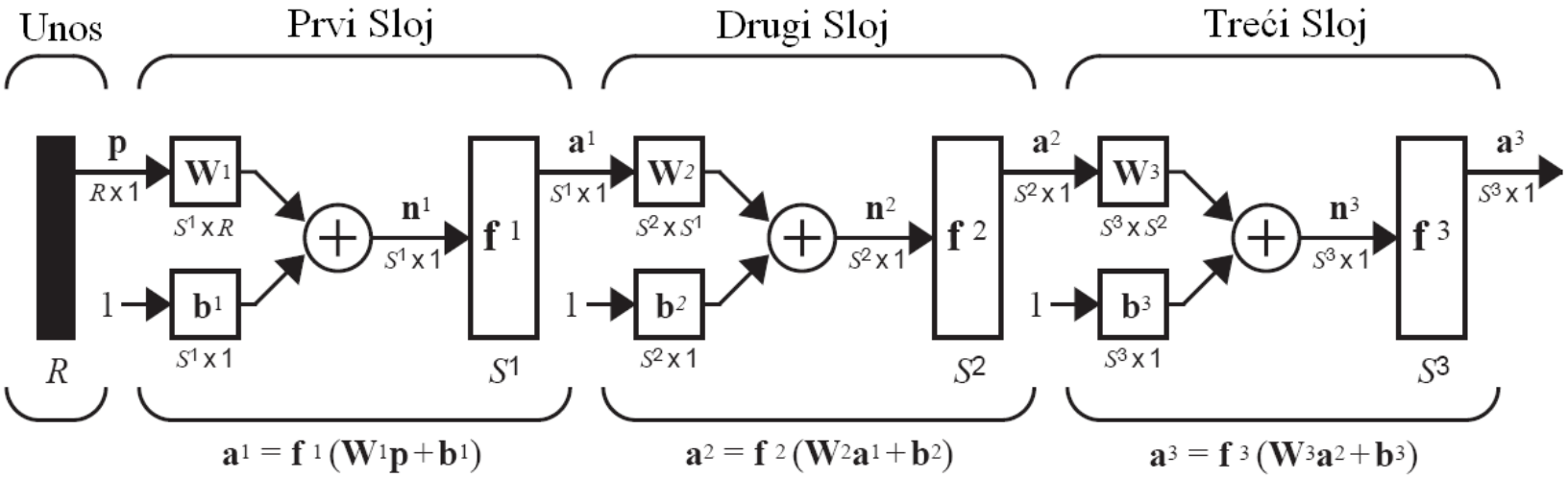
- Razmotrimo sada mrežu koja se sastoji od nekoliko slojeva. Svaki sloj ima svoju sopstvenu težinsku matricu **W**, svoj vektor pomeraja **b**, vektor mrežnog unosa **n**, kao i izlazni vektor **a**.
- Sada treba uvesti dodatne oznake kako bi razlikovali ove slojeve. Koristićemo desne gornje indekse za identifikaciju slojeva. Konkretno, priključićemo broj sloja kao desni gornji indeks oznaci svake promenljive koja pripada tom sloju.
- Tako, težinska matrica za prvi sloj biće napisana kao $\mathbf{W}^1$, a za drugi $\mathbf{W}^2$. Ovakva notacija korišćena je na slici, koja prikazuje mrežu od tri sloja.



$$a^3 = f^3(W^3 f^2(W^2 f^1(W^1 p + b^1) + b^2) + b^3)$$

Višestruki neuronski slojevi

- Vidimo da postoji R unosa u mrežu, S^1 neurona u prvom sloju, S^2 neurona u drugom, itd. Različiti slojevi mogu imati različit broj neurona. Izlazni podaci iz prvog sloja su ulazni podaci za drugi sloj, a izlazni podaci iz drugog su ulazni za treći sloj.
- Tako, drugi sloj može biti posmatran kao jednoslojna mreža koja ima $R = S^1$ unosa, $S = S^2$ neurona, i $S^1 \times S^2$ težinsku matricu $\mathbf{W}^2$.
- Unos u drugi sloj je $\mathbf{a}^1$, a iznos $\mathbf{a}^2$.
- Sloj čiji iznos je izlaz za čitavu mrežu zove se ***izlazni sloj***. Ostali slojevi naziviaju se ***skriveni slojevi.***
- Na dole prikazanoj mreži (u skraćenoj notaciji) postoji jedan izlazni sloj (treći sloj) i dva skrivena (prvi i drugi).



Višestruki neuronski slojevi

- Višeslojne mreže moćnije su od monoslojnih (ali su zato komplikovanije).
- U praksi, ako imamo četiri spoljašnje promenljive, postavićemo četiri unosa u mrežu. Slično, ako želimo da mreža ima sedam iznosa, onda izlazni sloj mora imati sedam neurona.
- Željene karakteristike izlaznog signala pomoćiće nam da odaberemo prenosnu funkciju za izlazni sloj (npr. ako želimo da izlazni podaci budu 0 ili 1, u izlaznom sloju koristićemo *hard limit* prenosnu funkciju).
- Šta ako imamo više od dva sloja?

  Predviđanje optimalnog broja neurona u skrivenim slojevima nije nimalo lak zadatak i aktivna je oblast istraživanja. Većina praktičnih neuronskih mreža ima samo dva ili tri sloja. Četiri sloja ili više u upotrebi su vrlo retko.
- Na kraju, potrebno je odabrati da li će neuroni imati pomeraj ili ne. Pomeraj dodaje mreži jednu promenljivu više, tako da možemo očekivati da mreže sa pomerajem budu moćnije od mreža bez njega.

Učenje neuronske mreže

- Pod ***pravilom učenja*** podrazumevamo proceduru za modifikaciju težina i pomeraja mreže. Ovakva procedura takođe se može zvati i ***algoritam treninga***.
- Svrha pravila učenja je obučavanje mreže da izvede određeni zadatak. Postoji mnogo tipova pravila učenja koja se svrstavaju u tri široke kategorije: učenje sa nadzorom, učenje bez nadzora i učenje sa ocenjivanjem.

  (1) Učenje sa nadzorom, pravilo učenja snabdeveno je skupom primera (tzv. trenažnim setom) pravilnog ponašanja mreže:

  $$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

  gde je $\mathbf{p}_q$ unos u mrežu, a $\mathbf{t}_q$ odgovarajući tačni iznos koji se zove *meta* (q = 1, 2,…,Q). Kada neki unos primenimo na mrežu iznos se poredi sa metom, a pravilo učenja se onda koristi za podešavanje težina i pomeraja mreže tako da iznos bude što približniji meti.

  (2) Učenje sa ocenjivanjem slično je učenju sa nadzorom, osim što umesto tačnih iznosa za svaki unos, algoritam koristi sistem ocenjivanja (bodovanja) rada mreže. Ocena je mera učinka mreže za određenu sekvencu ulaznih podataka. Ovakav način učenja je znatno manje uobičajen od učenja sa nadzorom.

  (3) Učenje bez nadzora, težine i pomeraji modifikuju se samo u odnosu na ulazne podatke. Ne postoje očekivani izlazni podaci – mete. Ali kako trenirati mrežu, a da ne znamo šta će ona da radi? Većina ovih algoritama vrši određenu vrstu grupisanja ulaznih podataka. Oni se, zapravo, uče da kategorizuju ulazne podatke u konačan broj klasa. Ovaj vid učenja je manje zastupljen od učenja sa nadzorom, ali ipak pronalazi mnogobrojnu primenu.

Algoritam propagacije unazad

- Algoritam propagacije unazad uveden je od strane Rumelharta i Meklelanda 1986. i predstavlja jedno od najznačajnijuh otkrića u razvoju veštačkih neuronskih mreža.
- Ovaj algoritam spada u klasu učenja sa nadzorom i koristi se za treniranje višeslojnih neuronskih mreža.
- U procesu učenja, mreži se predstavlja skup primera (1) koji se sastoje od serije ulaznih podataka i njima pridruženih (očekivanih) izlaznih podataka – meta.

$$\{\mathbf{p}1, \mathbf{t}1\}, \{\mathbf{p}2, \mathbf{t}2\}, \ldots, \{\mathbf{p}Q, \mathbf{t}Q\} \qquad (1)$$

- Na samom početku treninga, mreži se zadaju početne vrednosti težina i pomeraja (najčešće male pozitivne vrednosti), a zatim se primenjuju ulazni podaci iz trenažnog skupa.
- Kada se primene svi ulazni podaci, za dobijene izlazne podatke računa se greška E(**x**) kao razlika između stvarnih izlaznih podataka i očekivanih izlaznih podataka – meta (2):

$$E(\mathbf{x}) = \sum_{q=1}^{Q} (t_q - a_q)^2, \qquad E(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q), \qquad (2)$$

x je vektor koji sadrži sve težine i pomeraje, a T se odnosi na operaciju transponovanja

Greška ako postoji samo jedan izlazni podatak

Greška ako postoji više izlaznih podataka

Algoritam propagacije unazad

- Nakon prvog kruga treninga, mreži se zadaju nove (korigovane) vrednosti težina i pomeraja i čitav proces se ponavlja u iteracijama.
- Cilj algoritma propagacije unazad je traženje takve kombinacije težina (i pomeraja) koji odgovara minimumu funkcije E(**x**).
- Okosnicu algoritma čini tzv. ***delta pravilo*** (ili pravilo *gradijentnog spusta*), na kome se bazira korekcija težina i pomeraja:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \qquad w_{ij}(\mathbf{k}+1) = w_{ij}(\mathbf{k}) - \eta \frac{\partial E(\mathbf{k})}{\partial w_{ij}(\mathbf{k})};$$

$$\Delta b_i = -\eta \frac{\partial E}{\partial b_i}, \qquad b_i(\mathbf{k}+1) = b_i(\mathbf{k}) - \eta \frac{\partial E(\mathbf{k})}{\partial b_i(\mathbf{k})};$$

- Δw_{ij} je korekcija te težine u sledećoj iteraciji
- w_{ij}(k) predstavlja težinu veze između neurona *i* i neurona *j* u k-toj iteraciji
- Δb_i predstavlja korekciju pomeraja
- b_i(k) predstavlja pomeraj neurona *i* u k-toj iteraciji
- η naziva se ***brzina učenja*** i predstavlja pozitivnu konstantu.

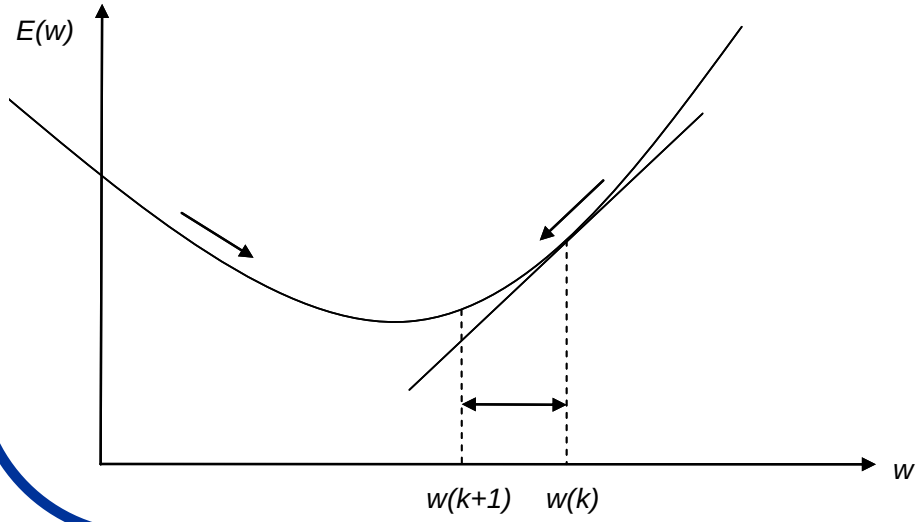
Vidimo da je korekcija težine jednaka negativnom gradijentu greške pomnoženom pozitivnom konstantom η.

Znak minus u relaciji govori nam da se korekcija težina odvija u smeru *gradijentnog spusta* ili niz gradijent greške

Algoritam propagacije unazad

- Na slici, u pojednostavljenoj formi, (2D slučaj) prikazana je korekcija težina koja se odvija u smeru ***gradijentnog spusta*** ili niz gradijent greške.
- Vrši se dakle, takva promena težina koja vodi do nižih vrednosti greške. Veličina korekcije zavisiće od vrednosti gradijenta ali i od vrednosti η, koja određuje hod (ili brzinu) učenja.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \qquad w_{ij}(\mathbf{k}+\mathbf{1}) = w_{ij}(\mathbf{k}) - \eta \frac{\partial E(\mathbf{k})}{\partial w_{ij}(\mathbf{k})};$$



Pravilo gradijentnog spusta primenjeno na imaginarnu mrežu koja se sastoji od jednoulaznog neurona. Težine se koriguju u smeru smanjenja vrednosti greške.

Vrednost greške u ovom uprošćenom primeru predstavljena je krivom E(w).

Ovaj algoritam pokušava da nađe apsolutni (globalni) minimum ove krive.

Algoritam propagacije unazad

- U opštem slučaju E(**x**) predstavlja površ (ili hiperpovrš) u prostoru težina. Tehnikom gradijentnog spusta ovaj algoritam pokušava da nađe apsolutni (globalni) minimum ove površi.
- Težinama se najpre daju male nasumično odabrane vrednosti (ovo je ekvivalentno nasumičnom odabiru tačaka na površi greške).
- Algoritam zatim računa lokalne gradijente na površi i menja težine u smeru negativnog gradijenta (tj. u smeru smanjenja vrednosti E(**x**)).
- Ako površ greške sadrži više od jednog mimimuma, važno je da algoritam ne ostane zarobljen u lokalnom minimumu. Ovo se prevazilazi uvođenjem tzv. ***konstante momenta*** i uopštavanjem delta pravila:

$$\Delta w_{ij}(\mathbf{k}) = \alpha \Delta w_{ij}(\mathbf{k}-1) - \eta \frac{\partial E(\mathbf{k})}{\partial w_{ij}(\mathbf{k})} .$$

  - Konstanta momenta predstavljena je sa ***α*** i ima vrednost između 0 i 1.

- Dodavanjem dela predhodne korekcije težine, tekućoj korekciji težine (koja je vrlo mala u lokalnom mimimumu) moguće je da algoritam pobegne iz lokalnog minimuma.
- Zbog izračunavanja izvoda greške po težinama, prenosne funkcije neurona u mreži koja se trenira ovim algoritmom moraju da budu glatke i diferencijabilne na čitavom svom domenu.

Algoritam propagacije unazad

Da rezimiramo:

Algoritam propagacije unazad odvija se u sledećih nekoliko faza:

(1) Inicijalizacija težina i pomeraja;

(2) Primena ulaznih vektora (iz trenažnog seta) na mrežu i njihova propagacija kroz slojeve mreže da bi dobili izlazni vektor;

(3) Računanje signala greške poređenjem stvarnih izlaznih vektora sa željenim izlaznim vektorima (metama);

(4) Propagacija signala greške nazad kroz mrežu (po čemu je algoritam dobio ime) i podešavanje težina tako da se umanji signal greške.

(5) Faze (2-4) se ponavljaju dok se ne postigne zadovoljavajuća vrednost signala greške

Primena NN

- **NN imaju široku primenu u različitim oblastima života:**

- Prepoznavanju rukopisa (sada je to popularno kod tablet - računara)
- Kompresiju slika
- Predviđanja berzanskih kretanja
- Medicinskoj dijagnostici
- Analizi spektara
- Ostalo

http://tralvex.com/pub/nap/

Kreiranje NN simulacije u programu MATLAB

- NN je nacin uspostavljanja relacije izmedju bilo kog seta ulaznih podataka sa nekim izlazom kada se nista ne zna o njihovoj medjusobnoj relaciji.
- Da bi mogli da koristimo NN, potrebno je da imamo veci set ulaznih i izlaznih podataka
- Napravićemo NN model koji ima tri ulazna podatka: a,b,c i pravi izlaz y.
- Da bi mogli da testiramo model, uzmimo da je zavisnost izmedju ulaza i izlaza:

$$y=5*a+b*c+7*c$$

- Uzecemo ovaj model da bi generisali ulazne i izlazne podatke koje nemamo
- U realnom slucaju, necemo imati matematicku relaciju izmedju ulaza i izlaza, već ćemo imati setove realnih ulaznih vrednosti (a,b,c) kao i izlaznih (y)
- Ovde, napravicemo setove ulaznih podataka koriscenjem komande “rand“
- Neka bude 1000 vrednosti za (a,b,c) i samim tim 1000 vrednosti izlaza (y)

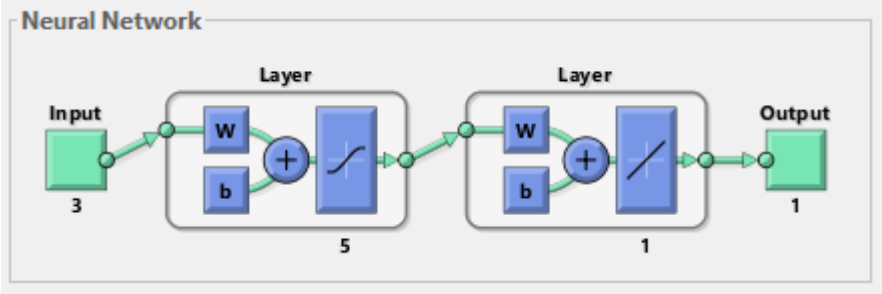
Kreiranje NN simulacije u programu MATLAB

```
% Generisanje matrice 1000 slucajnih vrednosti za a,b,c

a= rand(1,1000);      % Generisanje matrice a
b=rand(1,1000);       % Generisanje matrice b
c=rand(1,1000);       % Generisanje matrice c
n=rand(1,1000)*0.05; % Generisanje matrice za sum (n). Ovo je namerno
                        dodato da bi vrednosti za y izgledali kao
                        realni podaci
                      % Velicina suma je +-0.05 (tj. 0.1) i on je
                        uniforman

y=a*5+b.*c+7*c+n;     % Relacija koja racuna vrednosti y na osnovu
                        formule

I=[a; b; c];          % Pravimo jedinstvenu matricu redova i kolona
                      % ulaznih podataka I (I kao input)
O=y;                  % Pravimo matricu izlaznih podataka O (O kao
                        output i predstavlja zavisno promenljuvu "y")
```

Kreiranje NN simulacije u programu MATLAB

- NN treba da simulira mozak pun neurona koji su poredjani u vise slojeva.
- Prvi sloj uzima ulazne podatke koje sporvodi u medjuslojeve (skrivene slojeve).
- Poslednji (izlazni) sloj uzima izlaz iz skrivenih slojeva i prezentuje rezultat.
- Skrivenih slojeva moze biti proizvoljan broj.
- Svaki sloj je u stvari funkcija koja uzima neke promenljive (u obliku vektora u) i transformise ih u nove promenljive (vektor v) mnozeci ih sa nekim koeicijentima (matrica tezine w) i dodajuci im pomeraj b.
- Dimenzija vektora v poznata je kao v-dimenzija sloja: v=sum(w.*u)+b

- Napravimo NN sa 1 ulaznim i 1 izlaznim slojem.
- Dimenzija ulaznog sloja bice v=5 (a,b,c,w,b) tj. dimenzija ulaznog vektora koja je u=3 (tačke a,b,c) transformisaće se u vektor dimenzije 5 (v=5).
- Izlazni sloj imace dimenziju 1 i ima zadatak da ulazni vektor (u) dimenzije 5 transformise u vektor (v) dimenzije 1 (jer imamo samo jednu veličinu kao resenje)



Kreiranje NN simulacije u programu MATLAB

- Za kreiranje NN modela koristicemo komandu "newff"
- Najpre moramo napraviti matricu R za ulazne podatke. Ovde ce ona imati dimenzije 3x2 (3 je broj ulaznih promenljivih dok je 2 kolone potrebno za zadavanje min. i max. vrednosti ulaznih promenljivih.

```
R=[0 1; 0 1 ; 0 1]; % Prva kolona predstavlja minimalnu vrednost sva tri promenljive
                    a druga kolona njihovu maksimalnu vrednost. Posto su u pitanju
                    brojevi dobijeni rand komandom, opseg brojeva ce biti od 0 do 1
```

- Zatim pravimo matricu S koja odredjuje dimenzije ulazne matrice v tj. dimenzije svih slojeva. Rekli smo vec da je dim(v)=5

```
S=[5 1]; % Matrica S nam je potrebna da bi program znao koje su dimenzije slojeva
```

- Sada pravimo mrezu komandom "newff" sledecom sintaksom:

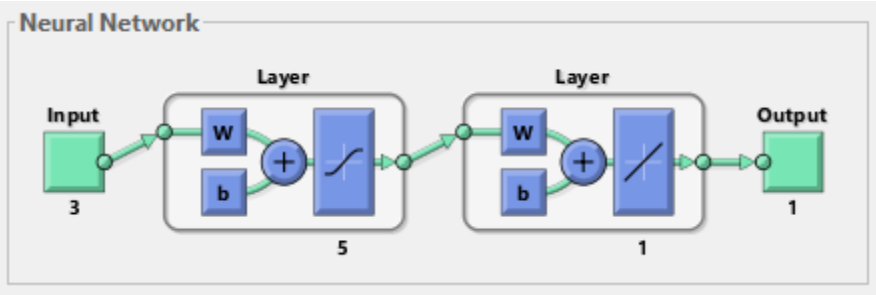
```
net = newff(R,S,{'tansig','purelin'}); % Promenljiva "net" je nas model NN
```

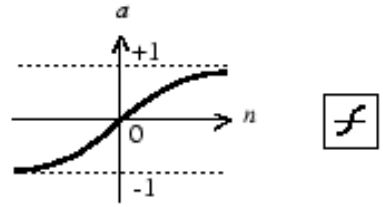
Kreiranje NN simulacije u programu MATLAB

- Zdržimo se na trenutak na prethodnom programskom redu. Tu se pominju dve nove komande: "tansig" i "purelin".

```
net = newff(R,S,{'tansig','purelin'});
```

- Prva komanda govori da nasa mreza za ulazni sloj koristi sigmoidalnu, a za izlazni sloj linearnu funkciju prelaza.



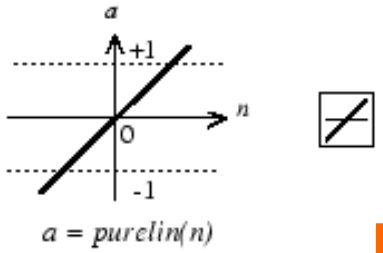


Tan-Sigmoid Transfer Function

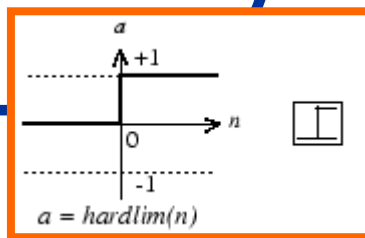
- Testirajmo kako ove dve funkcije izgledaju:

```
n = -5:0.1:5;
a = tansig(n);
plot(n,a)
```

```
n = -5:0.1:5;
a = purelin(n);
plot(n,a)
```



Testirajte kako izgleda i hard-limit prelazna funkcija (hardlim) koristeći sličnu sintaksu



Kreiranje NN simulacije u programu MATLAB

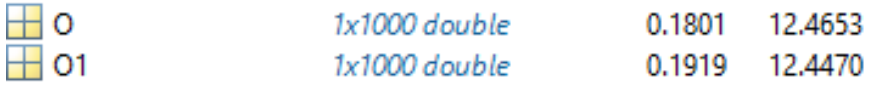
- Dobijena promenljiva "net" je naš model NN.
- Sada moramo da pustimo mrežu da uči. Mrežu ćemo trenirati podacima koje smo napravili korišćenjem "rand" komande. Koristimo sintaksu:

```
net=train(net,I,O);
```

- Pošto je mreža uspesno naučila, probajmo da simuliramo koju ćemo vrednost izlaza (O1) dobiti za "rand" generisan ulaz (I).

```
O1=sim(net,I);
```

- Ako uporedimo vrednost promenljive O i simulirane promenljive O1 videćemo da su praktično identične što znači da je naša mreža dobro naučila.



Kreiranje NN simulacije u programu MATLAB

- Ukoliko nacrtamo grafik vrednosti O i O1 za svih 1000 tacaka, možemo videti da su vrlo slične:

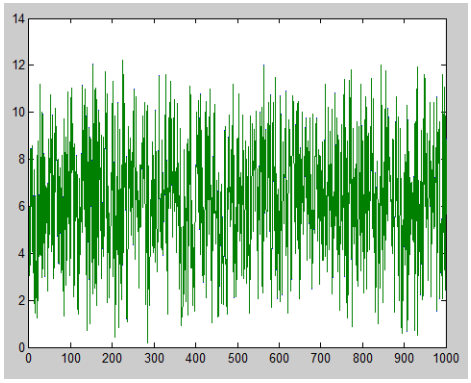
```
figure(1)
plot(1:1000,O,1:1000,O1);
```

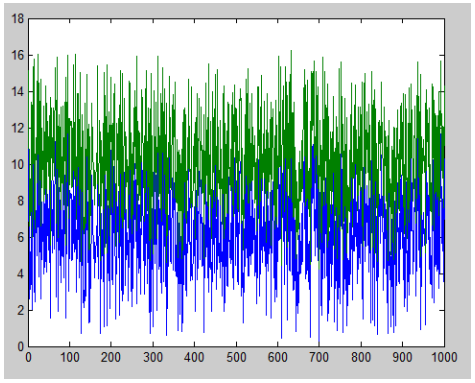
Da bi bolje videli, možemo vrednostima O1 dodati 4

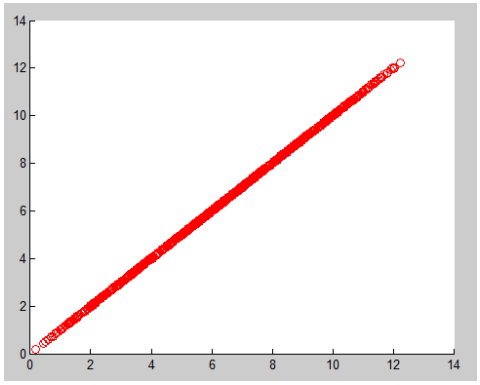
```
figure(1)
plot(1:1000,O,1:1000,O1+4);
```

Možemo nactati i uporediti realne (O) i simulirane (O1) tačke

```
figure(2)
scatter(O,O1);
```







Vidimo da su vrednosti O i O1 skoro identične

Kreiranje NN simulacije u programu MATLAB

- Sada bi mogli da testiramo našu naučenu mrežu za neke druge ulazne podatke.
- Recimo da su oni: a=1, b=1, c=1
- Za ove podatke, znamo da bi trebali da dobijemo vrednost y=13

```
y=a*5+b.*c+7*c
```

- Ulazna matrica će u ovom slučaju biti: I=[1 1 1]'
- Ovo " ' " je da bi dobili matricu sa 1 kolonom i 3 reda

```
y1=sim(net,[1 1 1]')
```

- Dobijamo broj koji je veoma blizak broju 13 (1*5+1*1+7*1)