Univerzitet u Beogradu
Fakultet organizacionih nauka

Diplomski rad

Beograd, 2004

Univerzitet u Beogradu
Fakultet organizacionih nauka

Aplikacioni okvir za razvoj neuronskih mreža

Diplomski rad

Profesor
Vladan Devedžić

Student
Zoran Ševarac

Beograd 2004

SADRŽAJ

UVOD

Cilj diplomskog rada je razvoj aplikacionog okvira (framework), koji će omogučiti jednostavno kreiranje i simulaciju raznih modela neuronskih mreža (NM). Na taj način stvoriće se osnova za razvoj NM u obliku softverskih komponenti, eksperimentalno istraživanje i praktičnu primenu ove tehnologije.
Ideja je da se definiše skup osnovnih softverskih klasa iz kojih će se izvoditi svi modeli. Pri tom treba uzeti u obzir sve strukturne i funkcionalne specifičnosti koje odlikuju različite modele. Da bi se to postiglo potrebno je izvršiti analizu glavnih predstavnika različitih modela, definisati zahteve za svaki od njih, a potom projektovati odgovarajuće rešenje.
Rad je organizovan u šest celina:

1. U prvom delu su ukratko izložene definicije i osnovni pojmovi iz ove oblasti.
2. U drugom delu su analizirani razni modeli neurona i arhitekture NM, a kao rezultat analize dobijeni su matematički i konceptualni model.
3. U trećem delu su isprojektovane klase aplikacionog okvira.
4. U četvrtom delu je izvršena je analiza i poređenje sa nekim postojećim aplikacionoim okvirima iste namene.
5. U petom delu prikazana je simulacija Kohonen-ove samoorganizujuće mreže, i dati su primeri primene iste.
6. U šestom delu data su zaključna razmatranja

U prilogu je data dokumentacija za najvažnije klase okvira.

1. NEURONSKE MREŽE – DEFINICIJE I OSNOVNI POJMOVI

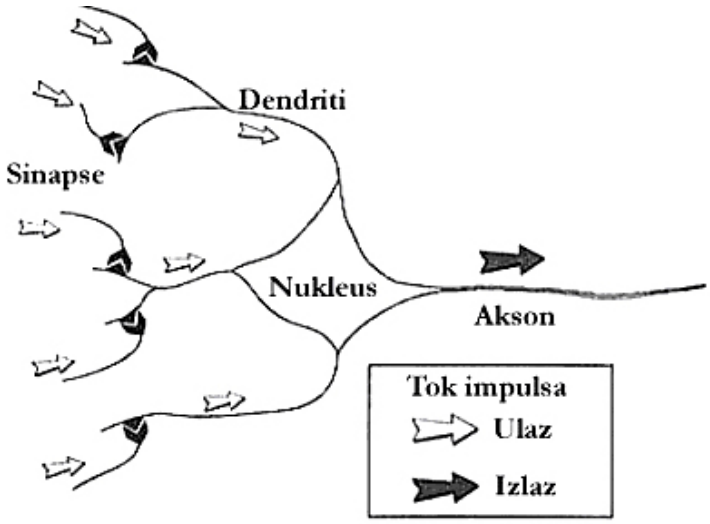
DARPA: Neuronska mreža je sistem koji se sastoji od velikog broja međusobno povezanih, jednostavnih elemenata procesiranja koji rade paralelno. Funkcija NM je određena strukturom mreže, težinom veza, i obradom u elementima procesiranja.

Haykin: Neuronska mreža je paralelni distribuirani procesor koji ima prirodnu sposobnost čuvanja i korišćenja iskustvenog znanja. Sličnost sa mozgom se ogleda kroz dve osobine:
- mreža stiče znanje kroz proces učenja
- znanje se čuva u vezama između neurona (sinaptičkim težinama)

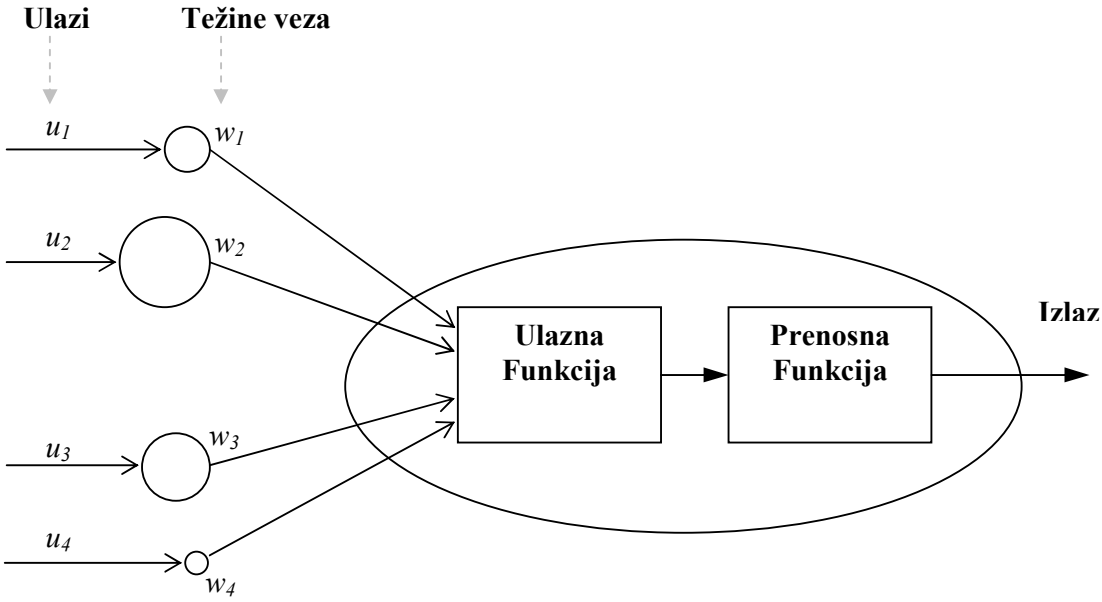
Zurada: Veštački neuro sistemi ili neuronske mreže, su ćelijski sistemi koji mogu da stiču, čuvaju i koriste iskustveno znanje.

U navedenim definicijama data su osnovna strukturna i funkcionalna svojstva NM, a to je da:
1) se sastoje od međusobno povezanih osnovnih jedinica (elemenata, ćelija) procesiranja, koje vrše neku jednostavnu, elementarnu obradu
2) jedinice procesiranja rade paralelno
3) imaju sposobnost učenja, čuvanja i korišćenja znanja

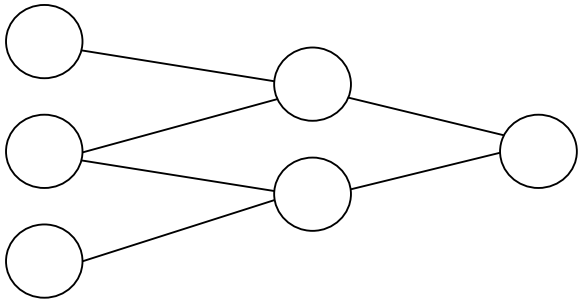
Veštačke neuronske mreže inspirisane su biološkim neuronskim mrežama, i predstavljaju njihov matematički odnosno računarski model. Neuroni su predstavljeni elementima procesiranja, a sinapse težinom veze. Dendriti su ulazi a akson je izlaz elementa procesiranja. Elementi procesiranja povezani su u mrežu tako što je izlaz svakog vezan na ulaz bar jednog od ostalih. Obrada koja se vrši u telu neurona predstavljena je funkcijama ulaza i prenosa.
Analogija između biološkog i veštačkog neurona prikazana je na slikama 1a. i 1b.



Slika 1a.Biološki neuron



Slika 1.b.Veštački neuron



Slika 2.Veštačka neuronska mreža

Algoritam za trening (učenje) mreže je proces u kome se vrši podešavanje težina ulaznih veza kako bi mreža imala željeno ponašanje. Podešavanje težina veza se vrši na osnovu određenog skupa podataka - uzoraka odnosno primera. NM dakle uče na osnovu primera, a ispoljavaju sposobnost generalizacije i van podataka koji su korišćeni za učenje.

2. OSNOVNI MODELI

2.1. MODEL NEURONA

Osnovna komponenta neuronskih mreža je neuron (element procesiranja). U skladu sa slikom 1.b. imamo sledeći matematički model:

u – ulazni vektor $[u_1, u_2, \ldots u_n]^T$
w – vektor težina $[w_1, w_2 \ldots w_n]^T$
net – ukupni ulaz iz mreže
g – ulazna funkcija
f – prenosna funkcija
y – izlaz

Izlaz neurona definisan je jednačinom

$$y = f(net) \qquad (1)$$

Ukupni ulaz iz mreže za pojedinačni neuron je vektorska funkcija težina veza i ulaza

$$net = g(u, w) \qquad (2)$$

Najčešće se računa se kao suma ulaza pomnoženih odgovarajućom težinom

$$net = \sum u_i w_i \qquad (3)$$

i predstavlja skalarni proizvod vektora ulaza i vektora težina.

Za funkciju prenosa se biraju funkcije ograničene na intervalima [0, 1] , [-1, 1]
U tabeli 1 prikazane su često korišćene funkcije prenosa.

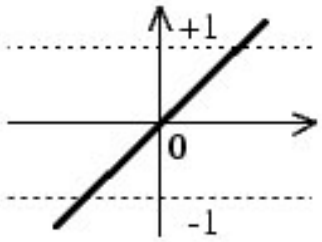
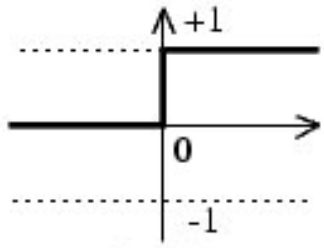
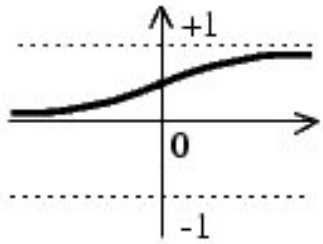
Linearna	$y=ax$	
Odskočna	$y=\begin{cases}1, x>0\\0, x\le 0\end{cases}$	
Sigmoidna	$y=\frac{1}{1+e^{-x}}$	

Tabela 1. Prenosne funkcije

Ovako definisan model neurona funkcioniše na sledeći način:

1. Ulazna funkcija nalazi sumu proizvoda ulaznih signala i težina odgovarajućih ulaznih veza
2. Ova suma je ulaz za funkciju prenosa, čiji izlaz predstavlja izlaz iz neurona

U osnovnom modelu često se pominje i prag *(threshold)* koji predstavlja minimalnu ili maksimalnu vrednost ulazne sume da bi neuron bio aktivan. Ovakvo ponašanje je karakteristično za odskočnu funkciju Ako je

T – prag

tada je

$$y=\begin{cases}a, net\ge T\\b, net<T\end{cases} \text{ - ekscitorni neuron} \qquad (4)$$

$$y=\begin{cases}a, net<T\\b, net\ge T\end{cases} \text{ - inhibitorni neuron} \qquad (5)$$

Uobičajni parametri su $a=1$ i $b=0$

Kod nekih modela imamo i unutrašnju pobudu *(bias)*, koja predstavlja konstantnu vrednost koja se dodaje ulaznoj sumi. Ako sa B označimo bias, tada je za neuron sa bias-om:

$$y=f(net+B) \qquad (6)$$

Negativan *bias* ima istu ulogu kao prag u nekim slučajevima (ekscitorni neuron sa odskočnom funkcijom prenosa kod koga je T=0).

Navedeni model naziva se McCulloch-Pitts (MP) model neurona. Pored njega postoje i razne varijante tzv. modifikovanog MP modela. Originalni MP model ima odskočnu funkciju prenosa, a modeli sa sigmoidnom funkcijom su tzv. modifikovani MP neuroni.

Kod kohonenove samoorganizujuće mreže neuroni umesto funkcije sume imaju funkciju rastojanja - euklidske metrike. Ova funkcija predstavlja intenzitet razlike vektora ulaza i vektora težina.

$$d = |u - w| = \sqrt{\sum_i (u_i - w_i)^2} \qquad (7)$$

Još i da napomenemo da ukoliko uvedemo vremensku dimenziju, tada jednačine (1) i (3) postaju

$$y(t)=f(net(t)) \qquad (8)$$
$$net(t) = \sum u_i(t) w_i \qquad (9)$$

Na izlazu neurona se može naći element za kašnjenje, kojim se odlaže uticaj izlaza na ostale neurone u mreži.

Pošto je računarska simulacija NM na jednoprocesorskim sistemima sekvencijalna, koristićemo modele u diskretnom vremenu i umesto *t* imati *k* koje označava tekuću iteraciju. Na taj način će se rešiti problem sinhronizacije izračunavanja.

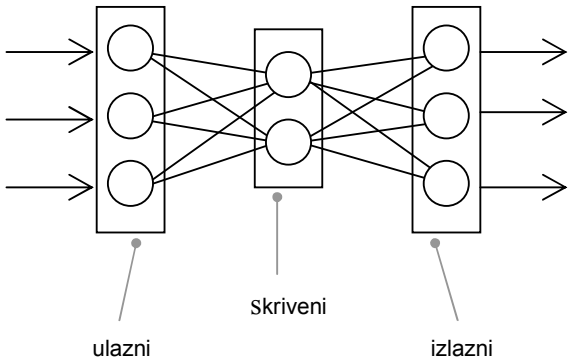
Dati matematički model i uvedeni pojmovi biće osnov za dalju analizu i formiranje konceptualnog modela. Pre toga biće analizirane osnovne arhitekture i algoritmi za učenje na konkretnim modelima.

2.2. MODELI NEURONSKIH MREŽA

2.2.1. ARHITEKTURE

Neuroni u mreži grupisani su u slojevima *(layers*). U zavisnosti od uloge u mreži, slojevi se mogu biti:

1) ***Ulazni*** – prima podatke iz okoline
2) ***Izlazni*** – daje rezultate obrade
3) ***Skriveni*** – nalaze se između ulaznog i izlaznog sloja. Nazivaju se skriveni jer njihovi ulazi i izlazi nisu dostupni iz 'spoljnog sveta', već se koriste za interne veze.



Slika 2.Višeslojna mreža sa prostiranjem signala unapred

Veze u mreži mogu da budu između neurona iz različitih slojeva, ili između neurona iz istog sloja.

Veze između slojeva mogu biti:

1) ***Jednosmerna veza unapred - feedforward***
Izlazi neurona u svim slojevima šalju signal (vezani su) isključivo na neurone u narednim slojevima.
2) ***Povratna sprega – feedback***
Izlazi neurona u nekim slojevima šalju signal unazad, na neurone iz prethodnih slojeva.

Za dva sloja kaže se da su

- ***potpuno povezana*** - ako je svaka ćelije prvog sloja, povezana na sve ćelije drugog sloja.
- ***delimično povezana*** – ako svaka ćelija prvog sloja ne mora obavezno da bude povezana na sve ćelije drugog sloja
- ***hijerarhijski povezana*** – ako su ćelije svakog sloja povezane samo na ćelije iz sledećeg susednog sloja

Pored navedenih postoji i tzv. ***rezonantna veza***, koja predstavlja dvosmernu vezu kod koje ćelije šalju signale sve dok nije zadovoljen određeni uslov.

Veze između ćelija u istom sloju mogu biti:

1) ***Rekurentne veze*** su realizovane tako da neuroni prvo prime pobudu iz drugog sloja, a zatim komuniciraju međusobno dok ne dostignu stabilno stanje. Kada dostignu stabilno stanje šalju signal sledećem sloju.
2) ***Inhibitorne veze ka okolini (on center/off soround)*** - kada neuron ima ekscitornu vezu ka samom sebi i neuronima iz susednog sloja, a inhibitornu vezu ka neuronima iz istog sloja.

2.2.2. ALGORITMI ZA UČENJE

Postoje dva osnovna tipa algoritma za učenje NM:

1) ***Učenje sa nadgledanjem (supervised learning)*** - zahteva učitelja. Učitelj je spoljni kontroler koji vrši podešavanje težina na osnovu razlike stvarnog i željenog izlaza mreže. Ova razlika predstavlja grešku i algoritam teži da je minimizuje.

2) ***Učenje bez nadgledanja (unsupervised learning)*** – mreža samostalno bez spoljnog uticaja kreira internu reprezentaciju ulaznih podataka po nekom pravilu koje je ugrađeno u mrežu. Ovaj oblik učenja zasniva se na ideji *samoorganizacije*.

Pored ova dva osnovna postoje i mnogi drugi algoritmi koje različiti autori tretiraju kao jedan od navedenih, ili kao posebnu vrstu. Postoji osnov za oba pristupa. Pomenućemo neke:

- ***Učenje pojačavanjem željenog ponašanja (reinforcement learning)*** – zahteva učitelja. Učitelj ocenjuje rezultate (ponašanje) mreže. Kod ovih algoritama se ponašanje koje je bliže željenom ocenjuje višom ocenom. Ovakav oblik učenja zasnovan je na *genetskim algoritmima,* pomoću kojih se često i realizuje. Zbog postojanja učitelja svrstavaju ga u učenje sa nadgledanjem.

- ***Kompetitivno učenje (competitive learning)*** se zasniva na principu da se grupe neurona međusobno 'nadmeću' *(compete)* da postanu aktivne. Kada jedan od neurona prevlada, on deaktivira ostale oko sebe. Zatim se vrši pojačavanje težina veza za čvor koji je prevladao i slabljenje težina za ostale čvorove u istoj grupi. Na taj način stanja izazvana zadavanjem ulaza postaju dominirajuća i stabilna. Zbog nepostojanja učitelja svrstavaju ga u algoritme bez nadgledanja.

Opšti matematički model za učenje sa nadgledanjem glasi:

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}(k) = w_{ji}(k) + \mu E(y_j(k), d_j(k)) \quad (1)$$

gde je:

$w_{ji}(k)$ - težina veze od *i*-og do *j*-og elementa, u *k*-oj iteraciji

$\Delta w_{ji}(k)$ - promena težine veze u *k*-oj iteraciji

u_{ji} – ulaz *j*-og elementa, $u_{ji}=y_i$

$y_j(k)$ - stvarni izlaz j-og elementa
$d_j(k)$ - željeni izlaz j-og elementa
μ - mala pozitivna konstanta koja se naziva koeficijent učenja
$E(y_j(k), d_j(k))$ - funkcija greške koja obično sadrži i u_{ji}

Učenje je završeno kada korekcija težine $\Delta w_{ji}(k)$, nakon konačnog broja iteracija k teži nuli. To se dešava kada se svi podaci za trening propuste nekoliko puta u slučajnom redosledu.

Opšte pravilo za učenje bez nadgledanja matematički se izražava na sledeći način:

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}(k) = w_{ji}(k) + \mu y_j(k) u_{ji}(k) \qquad (2)$$

Ova formula predstavlja tzv. Hebovo pravilo za učenje bez nadgledanja. Prema ovom pravilu težina se ne menja na osnovu željenog izlaza, već srazmerno prema stvarnom izlazu i ulazu. Proces se ponavlja za isti ulaz sve dok se težine menjaju. Kada težine prestanu da se menjaju znači da je mreža postala stabilna, tj. da je zapamtila zadati ulaz.

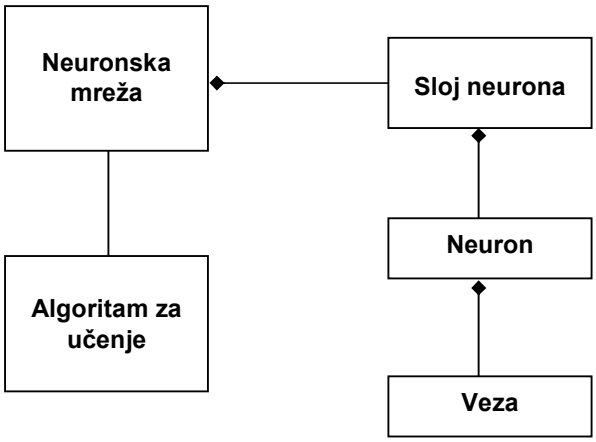
Pomenućemo još i opšte Hebovo pravilo *(Hebbian learning)* koje ima veliki uticaj u neuro-računarstvu. Prema ovom pravilu, težina veze se povećava ukoliko prva ćelija ima ekscitorno dejstvo na drugu, i ako je stalno (ili dovoljno često) aktivna. Time se pojačavaju veze grupe aktivnih neurona.

Nakon uvođenja osnovnih pojmova definisaćemo opšti konceptualni model, a zatim u definisanom okviru analizirati konkretne modele. Modeli koji će biti analizirani izabrani su tako da obuhvate prethodno navedene slučajeve u pogledu arhitekture i algoritama za učenje. Svaki model biće analiziran kroz dva aspekta:

1) Arhitektura
2) Algoritam za učenje

2. 3. KONCEPTUALNI MODEL

Na osnovu dosadašnje analize možemo izvesti sledeći konceptualni model:



Slika 3.Konceptualni model neuronske mreže

1) **Neuronska mreža** – sastoji se od jednog ili više slojeva međusobno povezanih neurona

2) **Sloj neurona –** skup neurona koji se u simulaciji tretira kao celina

3) **Neuron –** osnovni element obrade u NM. Karakterišu ga ulazna i prenosna funkcija

4) **Veza izmedju neurona –** svaki neuron ima skup ulaza koji su povezani na izlaze drugih, ili sopstveni izlaz. Svaku vezu karakteriše *težina*, koja predstavlja faktor koji množi ulazni signal.

5) **Algoritam za učenje –** pravilo po kome se vrši podešavanje težina kako bi mreža imala željeno ponašanje

2. 4. 1. ADALINE (ADAptive LInear NEuron)

Arhitektura

Ukoliko je funkcija prenosa *y=net* tj. izlaz neurona je jednak ukupnom ulazu, takva komponenta se naziva *adaptivni linearni kombinator (adaptive linear combiner - ALC)*. Jednačina ovog elementa je

$y = wu + B$

gde je

$w = [w_0, w_1 \ldots w_m]$ – vektor težina
$u = [u_0, u_1, \ldots u_m]^T$ – vektor ulaza
B – unutrašanja pobuda (bias)

Radi jednostavnosti se uzima da je $u_0=1$ *a* $w_0=B$ tako da je $y=wu$

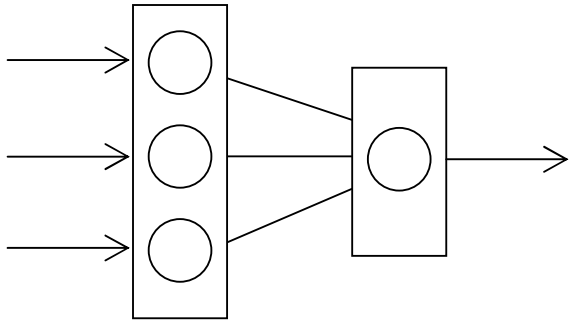
Ukoliko mu na izlazu stavimo prag ili signum funkciju imaćemo:

$$y = \begin{cases} wu, wu \geq T \\ 0, wu < T \end{cases}$$

odnosno

$y = sgn(wu)$

Adaline ima dva sloja ovakvih neurona, pri čemu ima samo jedan neuron u izlaznom sloju. Prvi sloj je ulazni i njegova uloga je samo da prosledi ulaze do izlaznog neurona.



Slika 4. Adaline

Adaline se može proširiti u *Madaline (Many Adalines)* kombinovanjem nekoliko *adaline* komponenti.

Algoritam za učenje: Widrow-Hoff (LMS) rule

Algoritam se zasniva na metodi najmanjih kvadrata *(Least Mean Squares)* pa se naziva i *MNK metoda.* Matematička osnova je metoda opadanja gradijenta kojom se vrši iterativno pretraživanje i nalazi minimum funkcije srednje-kvadratnog odstupanja.

Razmotrićemo prvo trening jednog elementa sa linearnom funkcijom prelaza, a zatim dobijeni model uopštiti za više elemenata i nelinearne izlaze.

LMS pravilo se može izraziti kroz sledeće jednačine:

$\varepsilon_p = d_p - y_p$ (1) greška izlaznog neurona za *p*-ti uzorak iz skupa za trening

$$E = \frac{1}{2}\sum_{p=1}^{n} \varepsilon_p^{\,2}$$ (2) ukupna greška za sve uzorke iz skupa za trening

$$w_{ji}(k+1) = w_{ji}(k) + \mu\varepsilon(k)u_{ji}(k)$$ (3) promena težine

$$\mu(k+1) = \mu(k) - \beta$$ (4) promena koeficijenta učenja

Jednačina (1) predstavlja grešku – razlika između željenog i stvarnog izlaza.
Jednačina (2) je izraz sa srednje kvadratno odstupanje i izračunava ukupnu grešku svih uzoraka za trening. Ukoliko je greška nula ili manja od dozvoljene trening je završen.
Jednačina (3) izračunava i vrši promenu težine veze.
Jednačina (4) postepeno smanjuje koeficijent učenja i na taj način obezbeđuje bolju konvergenciju treninga. Koeficijent učenja može biti i konstantan, ali se ovako postižu bolji rezultati.

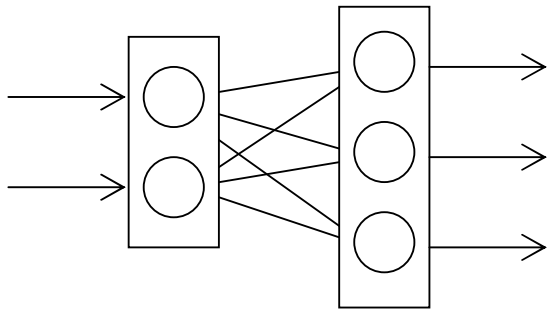
Ukoliko imamo više elemenata u izlaznom sloju tada jednačini (2) dodajemo sumu po elementima izlaznog sloja, a jednačina (3) prelazi u vektorski oblik.

Za elemente sa nelinearnim izlazom, pravilo se primenjuje pre nelinearne funkcije prenosa. U slučaju kada imamo prag i/ili odskočnu funkciju, greška se izražava kao razlika nivoa aktivacije i praga.

2. 4. 2. PERCEPTRON

Arhitektura

Perceptron se sastoji od dva potpuno povezana sloja neurona – ulaznog i izlaznog. Veza između slojeva je jednosmerna unapred (slika 5).



Slika 5. Perceptron

Elementi procesiranja u perceptronu su ekscitorni neuroni sa pragom, ulaznom funkcijom sume i odskočnom funkcijom prenosa.

Ponašanje neurona u perceptronu opisuje se sledećim jednačinama:

$net = \sum u_i w_i$ - ulazna suma

$y = \begin{cases} 1, net \geq T \\ 0, net < T \end{cases}$ - izlaz neurona

Iz matematičkog modela se vidi da je izlaz neurona jednak 1 ako je ulazna suma veća od praga, a 0 u suprotnom.

Algoritmi za učenje: Delta pravilo

Prvobitni algoritam za trening perceptrona koji je predložio Rosenblatt - *perceptron learning* glasi ovako:
Težina aktivne veze se povećava kada neuron nije aktivan a treba da bude, a smanjuje kada je aktivan a ne treba da bude. Matematički oblik je:

$$w(k+1) = w(k) + \mu(d(k) - y(k))u(k)$$

Algoritam je skoro identičan LMS algoritmu, ali bitna razlika je što se pri izračunavanju greške uzima izlaz nelinearne funkcije prenosa a ne ukupni ulaz.

Zbog problema i ograničenja koji su uočeni, iz ovoga je proisteklo delta pravilo koje predstavlja proširenje LMS pravila za nelinearne sisteme sa glatkim diferencijabilnim funkcijama prenosa.

$$w(k+1) = w(k) + \mu\varepsilon(k)u(k)f'(net(k))$$

Delta pravilo dobijeno je primenom pravila za izvod složene funkcije pri izračunavanju parcijalnih izvoda funkcije srednje-kvadratnog odstupanja po težinama. Time se zapravo dobija osetljivost funkcije greške na promenu težina.
Za primenu ovog algoritma potrebno je odabrati pogodnu funkciju prenosa, kojoj se lako (sa malo izračunavanja) može odrediti izvod. Zato se koriste sigmoidne funkcije:

$$y = \frac{1}{1+e^{-x}} \qquad y' = y(1-y)$$

$$y = \frac{1-e^{-x}}{1+e^{-x}} = \tanh(x) \qquad y' = 0.5(1-y^2)$$

Ukupna greška mreže, jednaka je sumi srednje-kvadratnih grešaka pojedinačnih neurona iz izlaznog sloja za sve elemente iz skupa za trening:

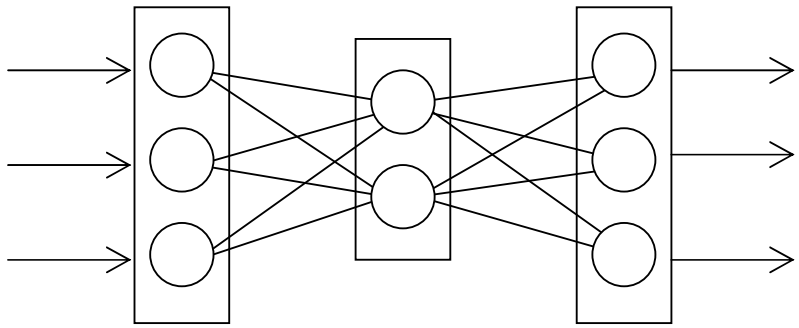
$$E = \frac{1}{2}\sum_{p=1}^{n}\sum_{j=1}^{m} E_{pj}{}^{2}$$

pri čemu je :
p - indeks elementa za trening
j – indeks neurona u izlaznom sloju

2. 4. 3. VIŠESLOJNI PERCEPTRON

Arhitektura

Višeslojni perceptron *(Multi layer perceptron)* pored ulaznog i izlaznog ima i najmanje jedan skriveni sloj neurona. U praksi se koriste mreže sa najviše 3 skrivena sloja.



Slika 6. Višeslojni perceptron

Algoritam za učenje: Prostiranje greške unazad (Backpropagation of error)

Backpropagation algoritam predstavlja generalizaciju delta pravila i odnosi se na mreže sa skrivenim slojevima.
Promena težina u izlaznom sloju vrši se isto kao kod delta pravila, i koristi se isti izraz za ukupnu grešku. Promena težina u skrivenom sloju (pre izlaznog) izračunava se po sledećem obrascu:

$$w_{ji}(k+1) = w_{ji}(k) + \mu f'(net_j(k))\left(\sum_a \varepsilon_a(k) f'(net_a(k)) w_{aj}(k)\right) u_{ji}$$

Izraz u zagradi je suma lokalnih grešaka δ pomnoženih težinom veze, od svih izlaznih neurona. Izraz predstavlja grešku koja se sa izlaznog prenosi na skriveni sloj. To se radi zato jer ne postoji željena vrednost za skriveni sloj, pa nije moguče neposredno izračunati grešku za neurone iz ovog sloja.

Analiza do sada predstavljenih algoritama za učenje

Svi do sada navedeni algoritmi zasnivaju se na metodi opadanja gradijenta i promena težine ima istu strukturu:

$\Delta w_{ji}(k) = \mu\delta_j(k)u_{ji}(k)$

pri čemu je δ *lokalna greška*

U zavisnosti da li je prenosna funkcija linearna ili nelinearna, i da li je neuron u izlaznom ili skrivenom sloju imamo imamo sledeće slučajeve:

1) Funkcija prenosa je linearna i neuron je u izlaznom sloju

$\delta_j(k) = \varepsilon_j(k)$

2) Funkcija prenosa je nelinearna i neuron je u izlaznom sloju

$\delta_j(k) = \varepsilon_j(k)f'(net_j(k))$

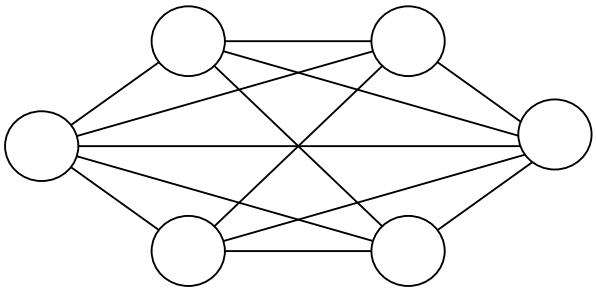
3) Funkcija prenosa je nelinearna i neuron je u skrivenom sloju

$$\delta_j(k) = f'(net_j(k))\sum_a \delta_a w_{aj}(k)$$

2. 4. 4. HOPFIELD-OVA MREŽA

Arhitektura

Hopfield-ova mreža je rekurentna mreža u kojoj je svaki neuron povezan sa svim ostalim neuronima u mreži sem sa samim sobom. Mreža nema slojevitu strukturu, odnosno svi neuroni se nalaze u istom sloju.



Slika 7. Hopfield-ova mreža

Neuroni imaju prag (negativan bias) i signum funkciju prenosa, a moguće je i korišćenje sigmoidnih funkcija.
Broj uzoraka koje mreža može da zapamti je 15% od broja neurona.
Osnovni princip funkcionisanja je da mreža kroz niz interakcija teži stabilnom stanju.
Uvodi se pojam energije mreže, i mreža teži stanju sa minimumom energije. Stanja sa minimumom energije su određena težinama, i to su stanja u kojima mreža 'raspoznaje' ulaz.

Učenje

Učenje može da se realizuje iterativno preko Hebbian-ovog pravila, ili se težine mogu izračunati unapred na osnovu podataka koje će skladištiti.

Težine veza su simetrične i mogu se izračunati na sledeći način:

$$w_{ji} = \sum_{a=1}^{M} u_j^{\,a} u_i^{\,a} \ , \ i \neq j$$

$$w_{ji} = w_{ij}$$

pri čemu je

$u=[u_1, \ldots u_n]$ – ulazni vektor odnosno uzorak
M – broj uzoraka

Ukoliko se koristi Hebbian-ovo pravilo tada je:

$$w_{ji}(k+1) = w_{ji}(k)+\Delta w_{ji}(k)$$

$$\Delta w_{ji}(k+1) = \mu y_j u_{ji} = \mu y_j y_i$$

Jednačina neurona u Hopfield-ovoj mreži je

$$y_j(k+1) = f(\sum_{i=1, i\neq j}^{N} w_{ji} y_i(k) - b_j + u_j(k))$$

Ulaz u_j se pojavljuje u jednačini samo prilikom zadavanja ulaza (u k-oj iteraciji), nakon čega nestaje i pušta mrežu da pređe u stabilno stanje.

Jednačina za ukupnu energiju mreže je:

$$E(k) = -\frac{1}{2}\sum_j \sum_{i\neq j} w_{ji} y_j(k) y_i(k) + \sum_j y_j(k) b_j$$

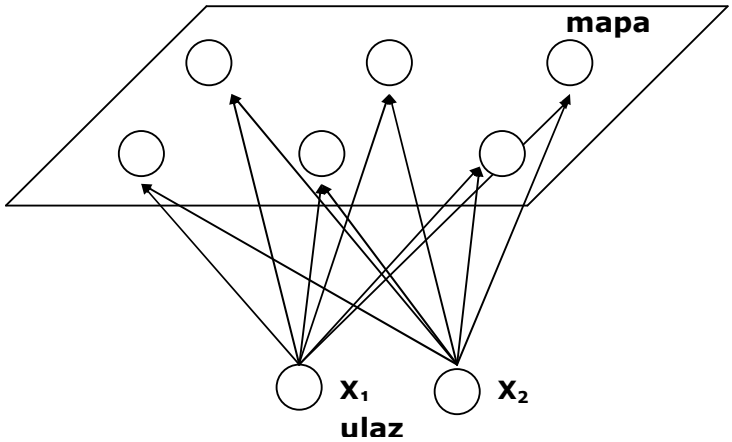
Može se dokazati da se posle svake iteracije ukupna energija mreže se smanjuje:

$E(k+1)<E(k)$

2. 4. 5. KOHONENOVA SAMO-ORGANIZUJUĆA MAPA

Arhitektura

Kohenenove SOM su zasnovane na kompetitivnom učenju, uređuju i prepoznaju skup ulaznih vektora. Mreža ima samo dva sloja neurona - ulazni sloj i mapu. Broj ulaza odnosno neurona u ulaznom sloju predstavlja dimenziju mreže. Sloj mape je najčešće 1D ili 2D, a može biti i veći. Na slici 8 je data struktura ovog modela sa šest ćelija u sloju mape.



Slika 8. Kohenenova samo-organizujuća mapa

Algoritam za učenje

Kohenenova mapa koristi euklidsku metriku odnosno funkciju rastojanja, za izbor pobednika tj. dominirajuće ćelije:

$$d_j = \left| x - w_j \right| = \sqrt{\sum_i (x_i - w_{ji})^2} \qquad (1)$$

U nekim implementacijama se radi pogodnosti u izračunavanju koristi kvadrat funkcije rastojanja, što je jednačina (1) bez korena.

Dominirajuća ćelija *c* je ona sa najmanjim rastojanjem d, od ulaznog vektora.

$$d_c = \min d_j \qquad (2)$$

Podešavanje težina se vrši po formuli

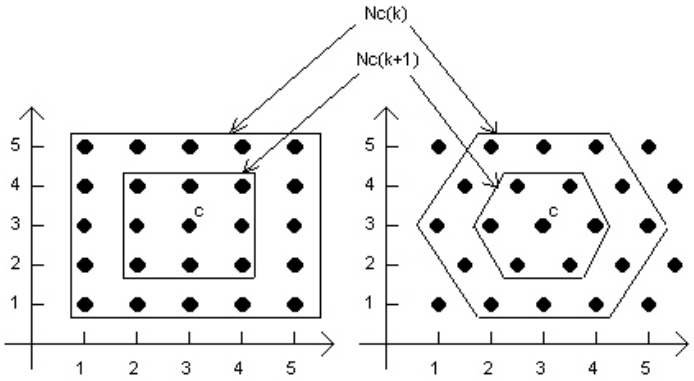
$$w_j(k+1) = w_j(k) + h_{cj}(k)\left[x(k) - w_j(k)\right] \qquad (3)$$

gde je
k – diskretna celobrojna vremenska koordinata odnosno iteracija treninga
h_{cj} – funkcija susedstva koja se definiše kao

$$h_{cj}(k) = \begin{cases} \alpha(k), j \in N_c(k) \\ 0, drugacije \end{cases} \qquad (4)$$

gde je $\alpha(k)$ koeficijent učenja, linearna monotono opadajuća funkcija.

Kroz trening pored podešavanja težina dominirajućeg neurona vrši se i podešavanje težina njemu susednih neurona.Uobičajna susedstava su pravougaona ili šestougaona (slika 9)



Slika 9. Pravougaono i šestougaono susedstvo

Početne vrednost za koeficijent učenja je obično 0.9 i vremenom postepeno opada, a veličina susedstva odgovara poluprečniku mape i takođe se smanjuje vremenom.

U sloju mape se koristi princip kompetitivnog učenja (lateralna inhibicija), međutim zbog brzine se u praksi koristi spoljni kontroler koji određuje 'najbližu' ćeliju.

Umesto funkcije rastojanja moguće je koristiti i skalarni proizvod vektora, ali potrebno je prethodno izvršiti normalizaciju vektora težina i ulaza.

3. DIZAJN I IMPLEMENTACIJA FRAMEWORK-A ZA RAZVOJ NM

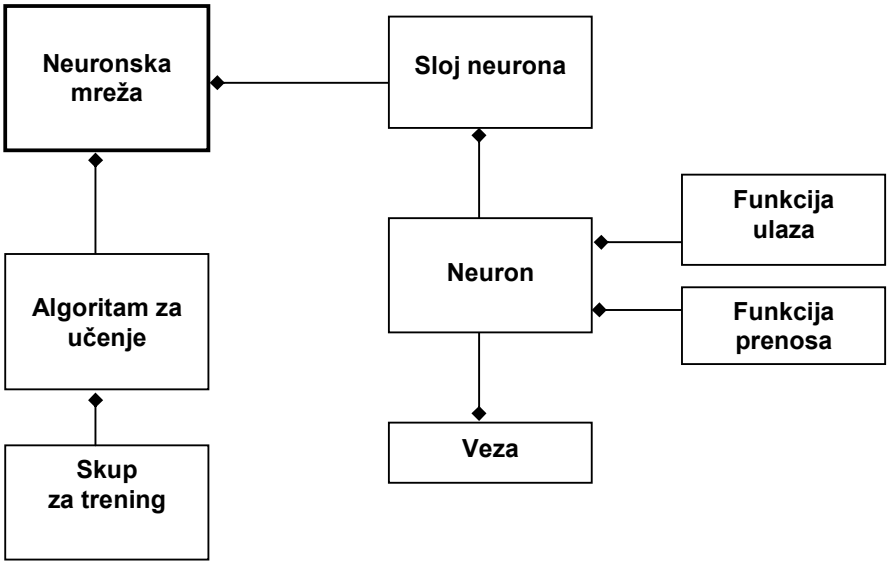
3. 1. Definisanje funkcionalnih i strukturnih zahteva

Na osnovu date strukturne i funkcionalne specifikacije glavnih predstavnika različitih NM, definisaćemo opšti model koji će poslužiti kao osnova za njihovo izvođenje. Neke stavke iz prethodnih analiza biće ponovljene kako bi na jednom mestu imali kompletan zahtev.

1) Neuronska mreža se sastoji iz jednog ili više slojeva neurona
2) Sloj je strukturni element mreže i sastoji se od jednog ili više neurona
3) Neuron je osnovni element obrade.
4) Obrada koju neuron vrši određena je funkcijama ulaza i prenosa.
5) Neuroni u mreži povezani su težinskim vezama.
6) Svaka vrsta NM ima odgovarajući algoritam za učenje.
7) Algoritam za učenje vrši podešavanje težina veza kako bi mreža imala željeno ponašanje.
8) Algoritmi za učenje sa koriste skup elemenata za trening.
9) Skup elemenata za trening sa nadgledanjem je skup parova *(u, y)* koji predstavljaju ulaz i željeni izlaz, odnosno skup ulaznih vektora *U* kod treninga bez nadgledanja.
10) Prilikom simulacije potrebno je omogućiti sekvencijalno, slučajno, paralelno i predefinisano izračunavanje.

3. 2. Konceptualni model

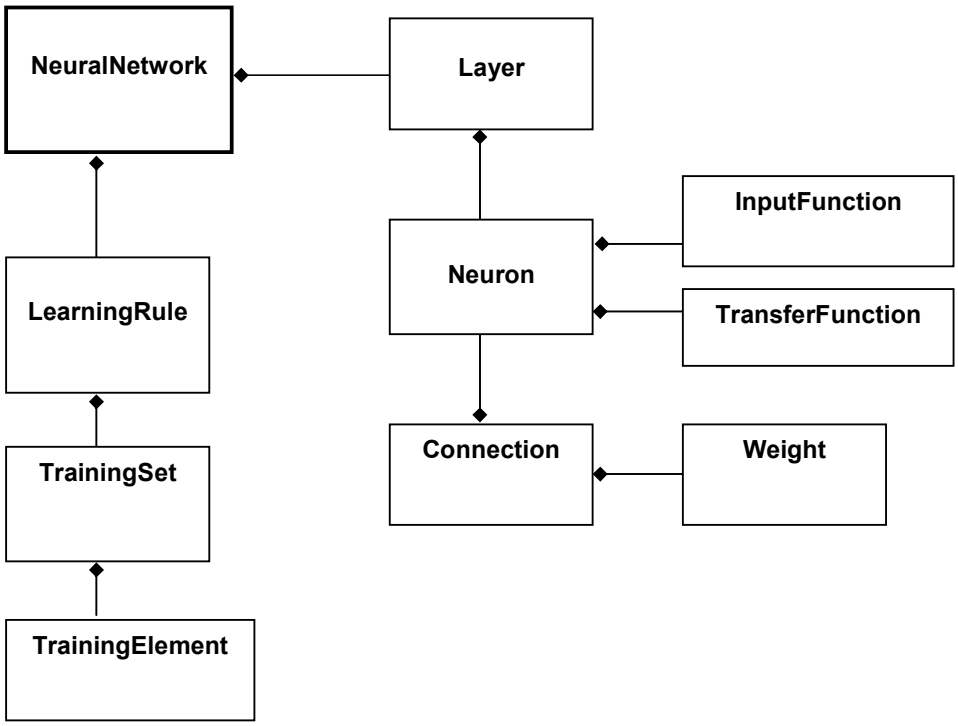
Iz prethodno navedene specifikacije izveden je konceptualni model koji predstavlja osnov za definisanje odgovarajućih klasa.



Slika 10. Konceptualni model

3. 3. Dijagram klasa

Na slici 11 dat je dijagram klasa neposredno proizašao iz konceptualnog modela.



Slika 11. Dijagram klasa

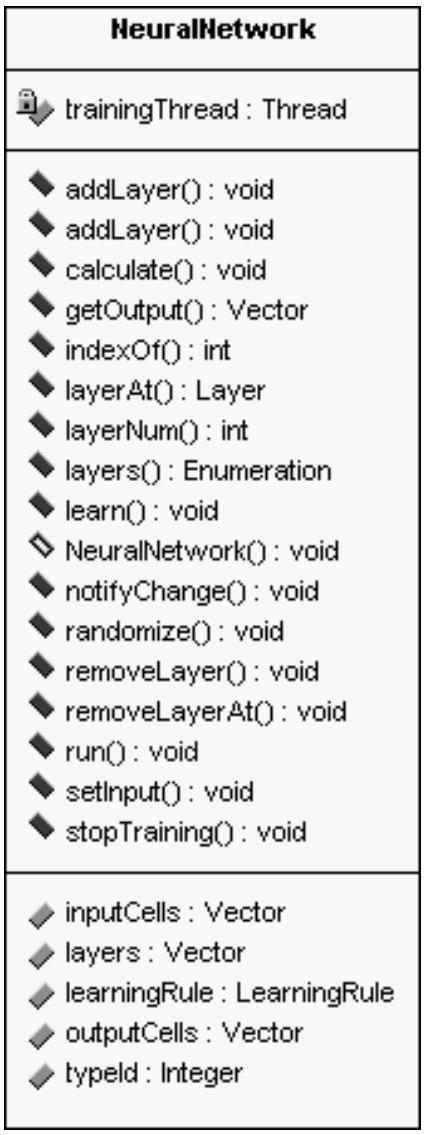
Klasa	Uloga
NeuralNetwork	Neuronska mreža
Layer	Sloj neurona
Neuron	Neuron
InputFunction	Ulazna funkcija
TransferFunction	Prenosna funkcija
Connection	Veza
Weight	Težina veze
LearningRule	Algoritam za učenje
TrainingSet	Skup za trening
TrainingElement	Primer za trening

3. 4. Klase aplikacionog okvira

Klasa *NeuralNetwork*

Neuronska mreža je kolekcija slojeva povezanih neurona.
Neuronsku mrežu možemo posmatrati kao adaptivni ulazno–izlazni sistem, koji uči pomoću odgovarajućeg algoritma koji je određen strukturom mreže.

Klasa *NeuralNetwork* je kontejner za slojeve neurona i agregira objekat koji realizuje algoritam za učenje. Takođe obezbeđuje interfejs za izvršenje osnovnih operacija nad mrežom. Realizuje osnovne funkcionalnosti zajedničke za sve modele i predstavlja baznu klasu za konkretne modele.

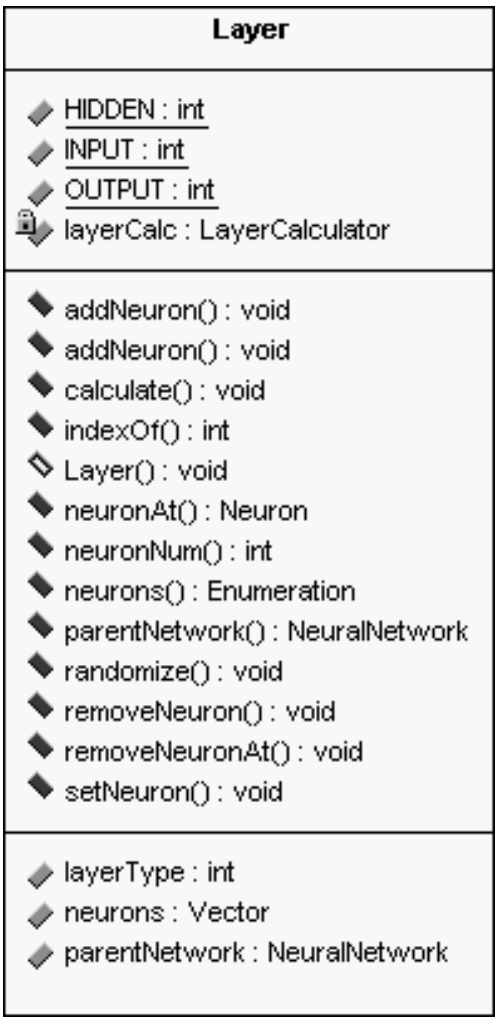


Slika 12. Klasa NeuralNetwork

Klasa *Layer*

Sloj neurona je kolekcija neurona.

Klasa Layer realizuje sloj neurona. Sadrži kolekciju neurona, metode za dodavanje, izbacivanje i pristup neuronima, i metode za inicijalizaciju i izračunavanje sloja.



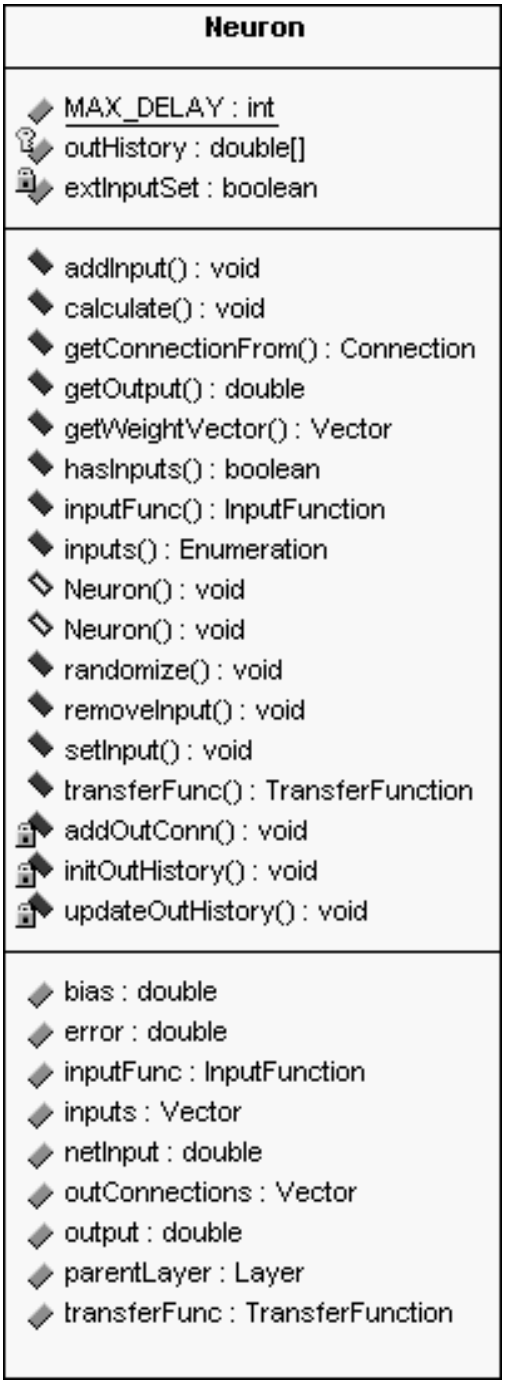
Slika 13. Klasa Layer

Klasa *Neuron*

Neuron je osnovni strukturni i funkcionalni element NM.

Svaki neuron sadrži:

- kolekciju ulaznih veza i metode za dodavanje i izbacivanje veza
- ulaznu i prenosnu funkciju i metode za pristup i postavljanje istih
- atribute koji predstavljaju ukupan mrežni ulaz, metodu za izračunavanje izlaza i metode za pristup navedenim atributima
- bias vrednost, bafer za grešku - error, baffer za kašnjenje



Slika 14. Klasa Neuron

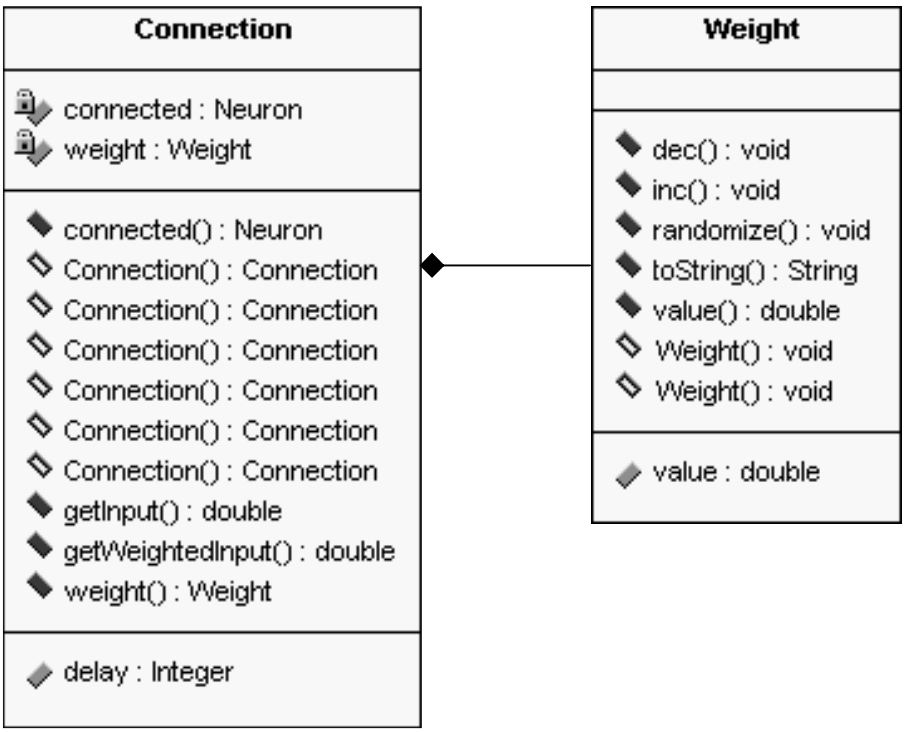
Klase *Connection* i *Weight*

Svaka ulazna veza ima težinu, i predstavlja spoj između izlaza i ulaza dva neurona.

Klasa *Connection* realizuje vezu između neurona. S obzirom na ulogu i značaj koji težina veze ima kod neuronskih mreža modelovaćemo je posebnom klasom.

Klasa *Connection* sadrži objekat klase *Weight* koji predstavlja težinu veze i referencu na neuron koji je sa druge strane veze.

Klasa *Weight* ima atribut koji predstavlja vrednost težine i metode za pristup i modifikaciju težine.



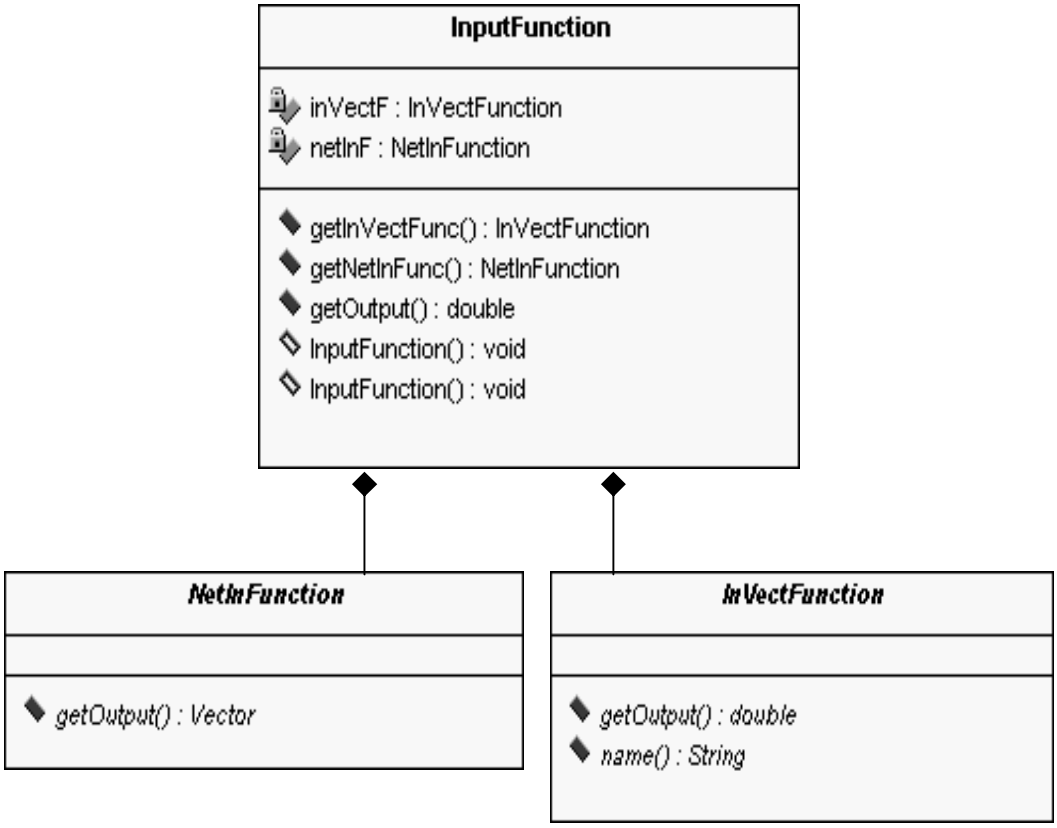
Slika 15. Klase Connection i Weight

Klasa *InputFunction*

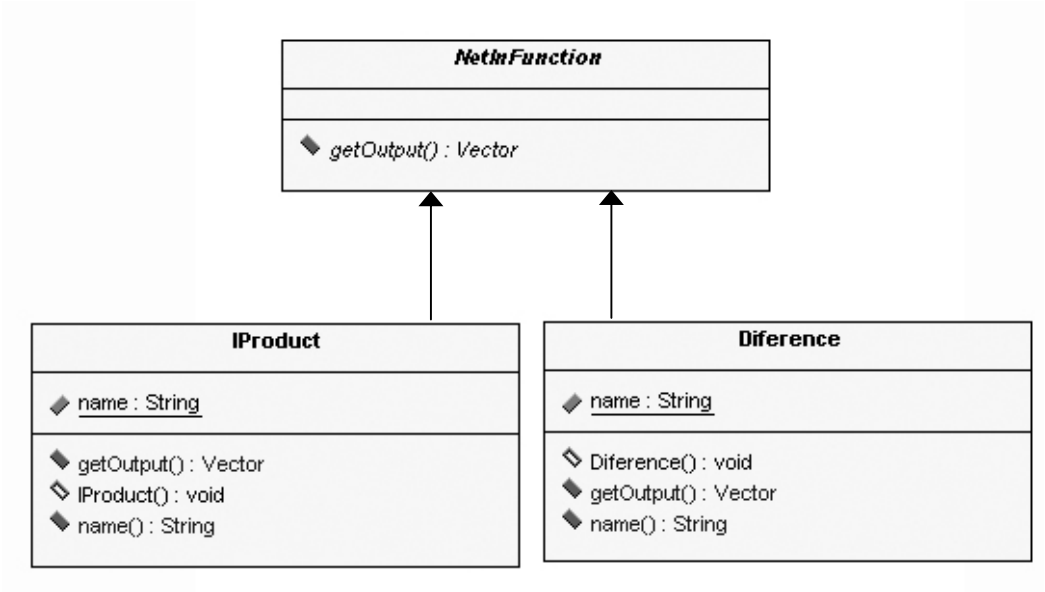
Ulazna funkcija neurona predstavlja funkciju vektora težina ulaznih veza i vrednosti ulaza. Kod MP neurona to je skalarni proizvod vektora, a kod SOM euklidska metrika. Moguće su i razne varijacije npr. logičke funkcije. Kako bismo izbegli ponavljanje koda i omogućili realizaciju svih navedenih slučajeva, ulaznu funkciju ćemo dekomponovati na dve komponente:

1. Funkciju ulaza mreže – *NetInFunction*
2. Ulaznu vektorsku funkciju – *InVectFunction*

Funkcija ulaza mreže je vektorska funkcija čiji su argumenti vektor težina i vektor ulaza. Kao rezultat vraća vektor koji je njihova razlika, linearna transformacija ili nešto drugo u zavisnosti od konkretne implementacije.
Argument *ulazne vektorske funkcije* je vektor dobijen od *funkcije ulaza mreže*.
Ova funkcija vraća sklar koji predstavlja *ukupni ulaz iz mreže – net*. Ova vrednost je argument za funkciju prenosa. Konkretne implementacije mogu biti npr. L_1 ili L_2 norma.

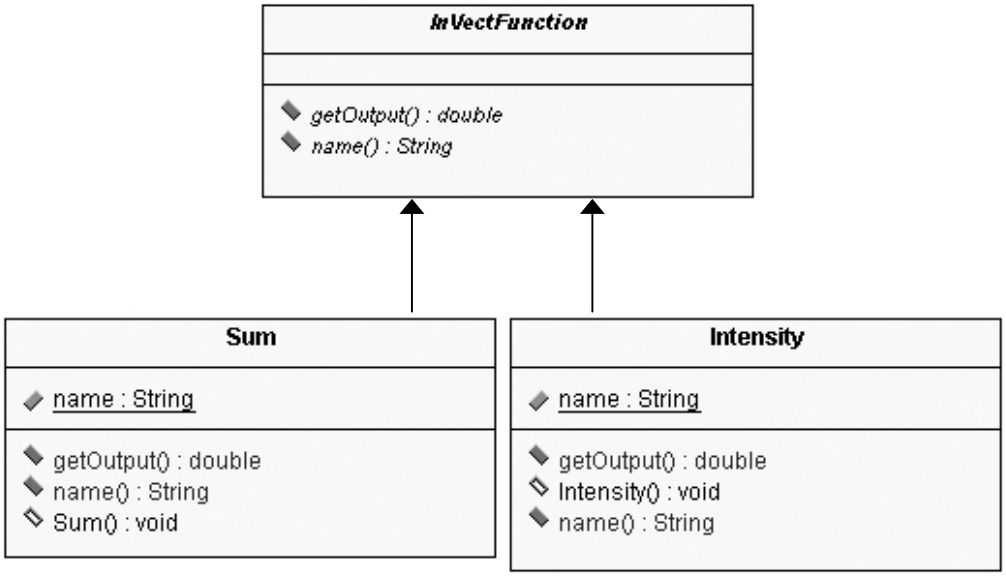


Slika 16. Klasa InputFunction



Slika 17. Naslednici klase NetInFunction

Klase koje nasleđuju *InVectFunction*:
Diference – razlika vektora težina i ulaznog vektora
IProduct – Množenje vektora težina dijagonalizovanim vektorom ulaza

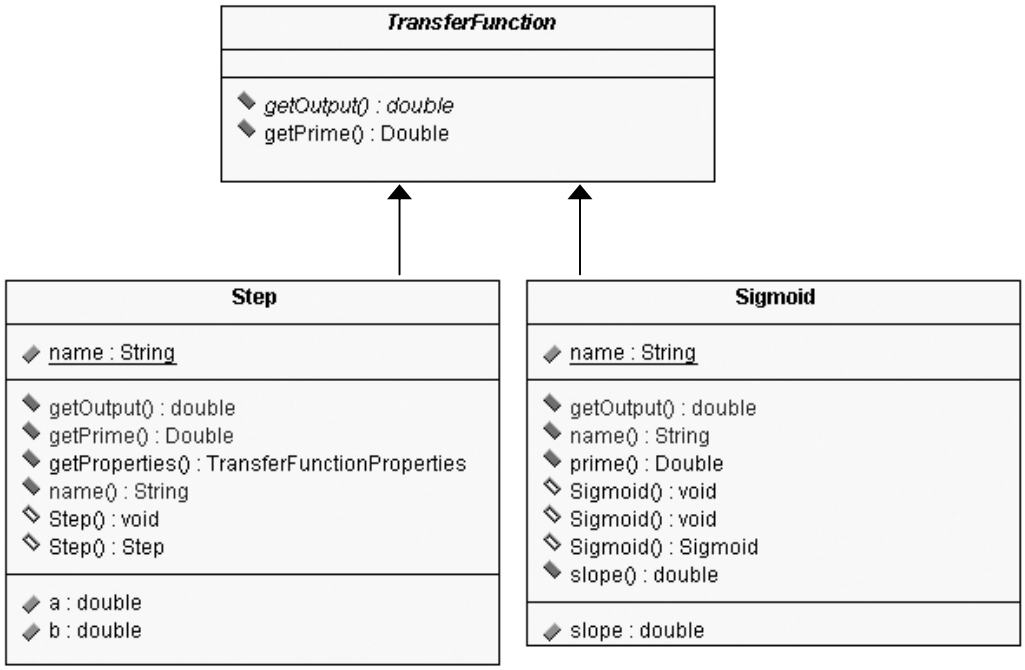


Slika 18 Naslednici klase InVectFunction

Klase koje nasleđuju *InVectFunction*:
Intensity – intenzitet vektora
Sum – suma
SumSqr – suma kvadrata
Or – logičko *ili*
And – logičko *i*

Klasa *TransferFunction*

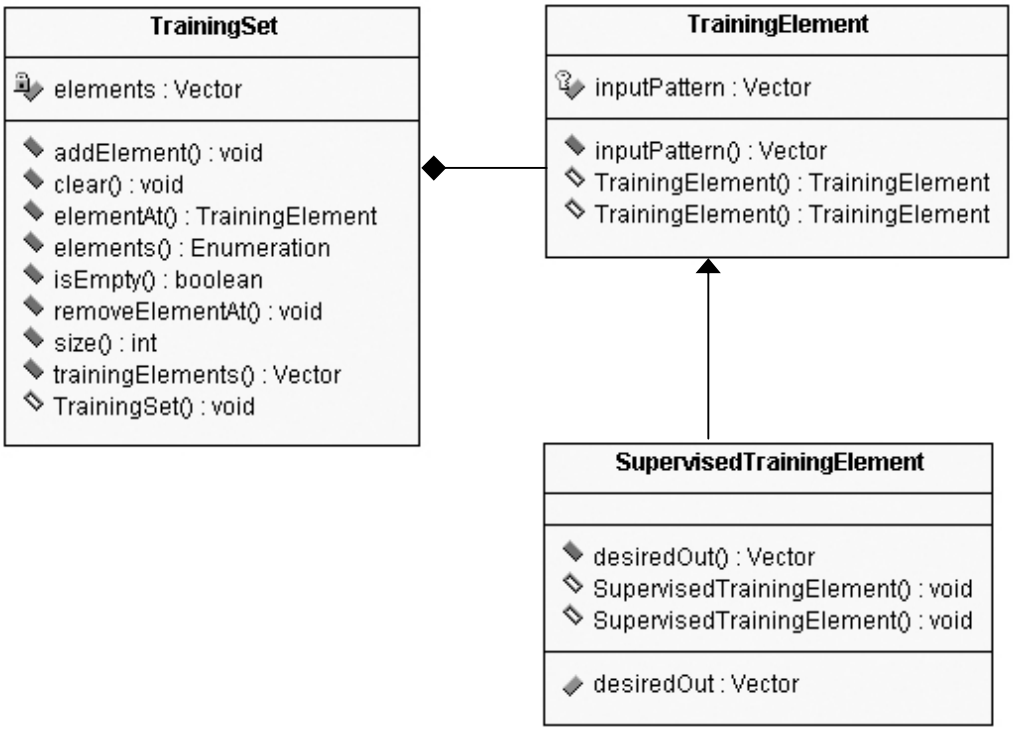
TransferFunction realizuje funkciju prenosa. Ova klasa je apstrakcija funkcije prenosa i ima metode koje vraćaju vrednost funkcije i izvod. Takođe ima i metod koji vraća istinitosnu vredost u zavisnosti od toga da li funkcija ima prag. Prag funkcije prenosa biće realizovan pomoću *decorater* paterna – klasa *ThresholdFunction*.



Slika 19. Naslednici klase TransferFunction

Klase *TrainingSet* i *TrainingElement*

Skup elemenata za trening je skup vektora ulaza, odnosno skup parova vektora (u, y) koji predstavljaju ulaz i željeni izlaz.
Svaka komponenta vektora je vrednost ulaza (izlaza) ćelije iz ulaznog (izlaznog) sloja.



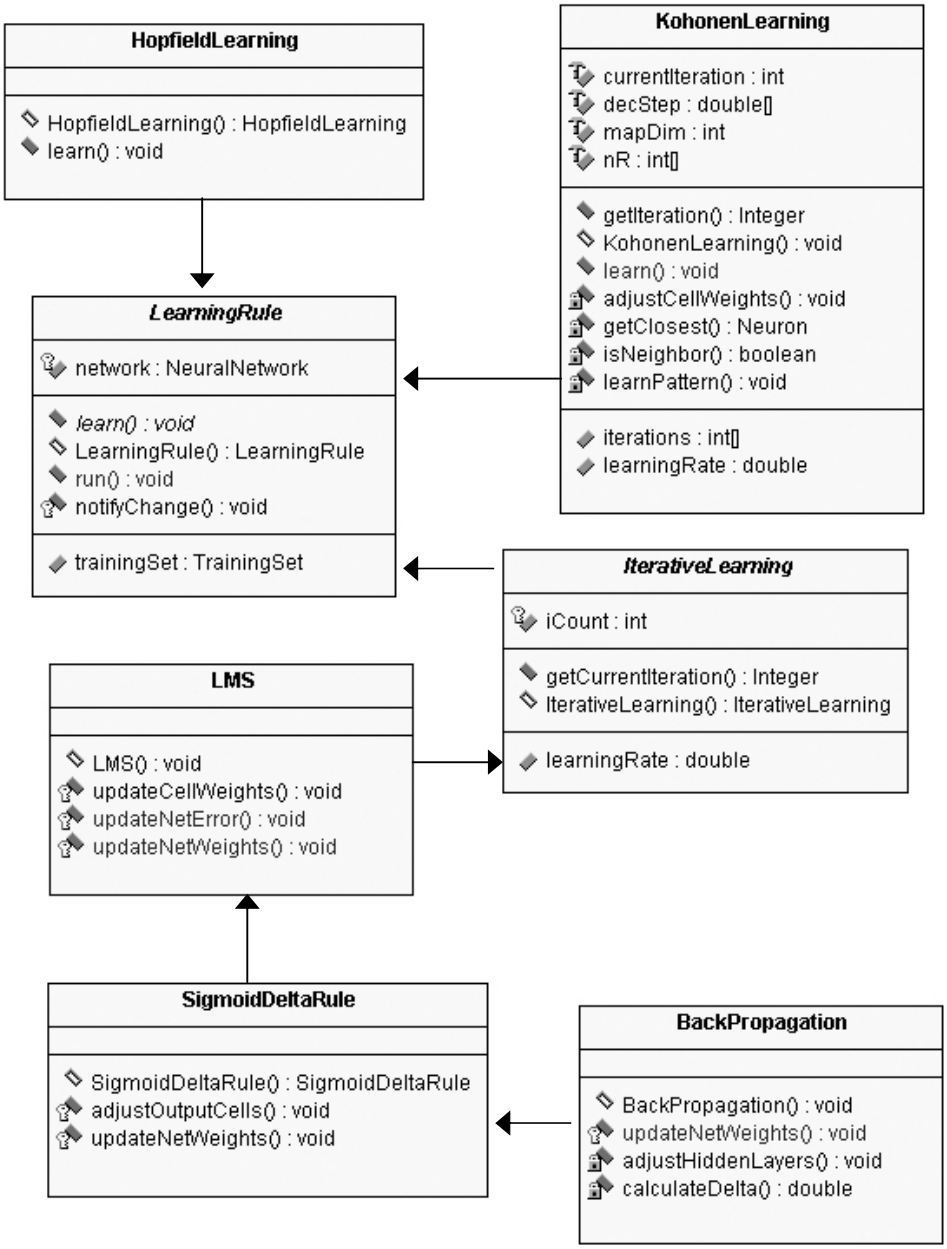
Slika 20. Klase TrainingSet i TrainingElement

Klasa *LearningRule*

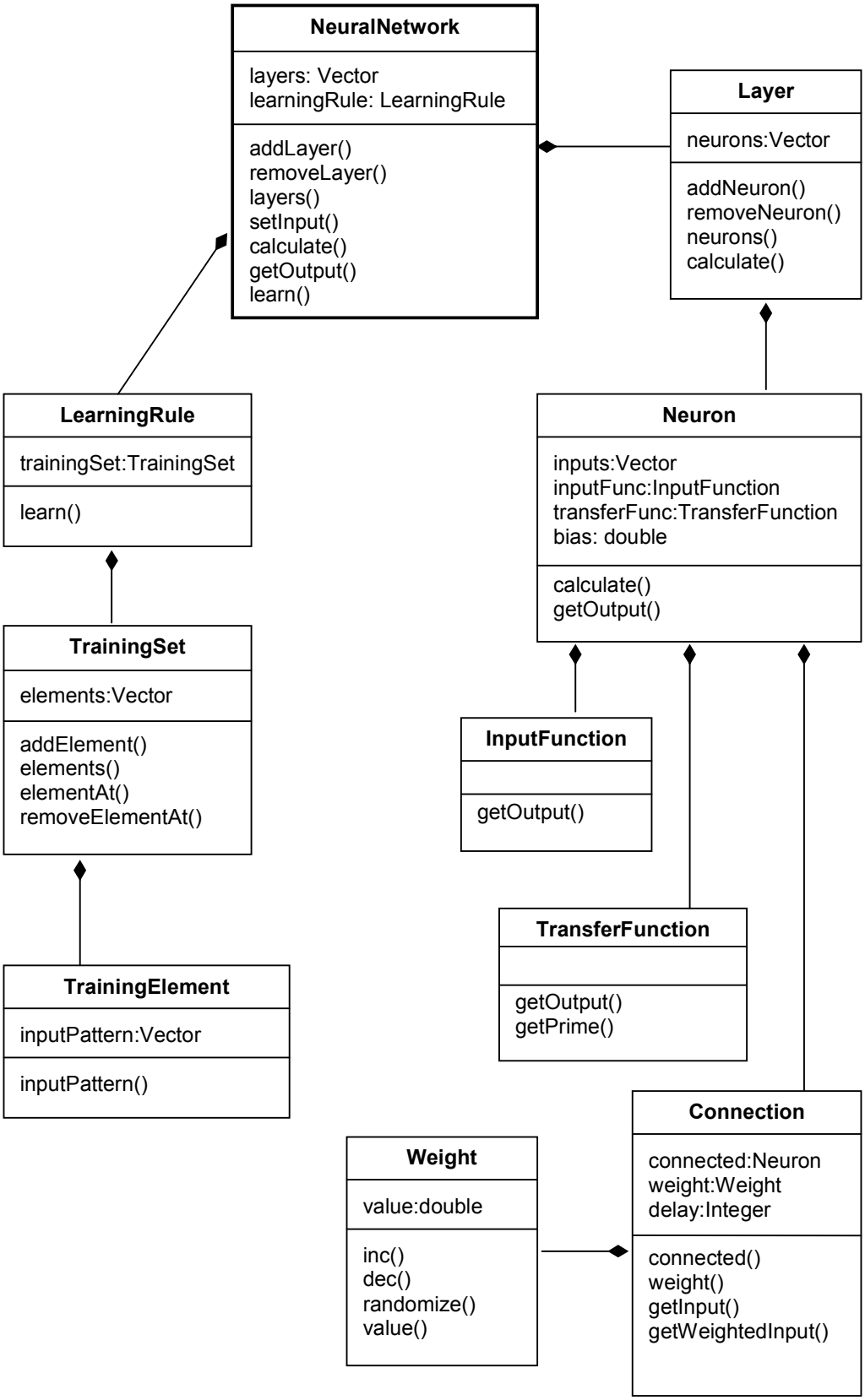
Klasa *LearningRule* je generalizacija algoritma za učenje. Sadrži:

- referencu na mrežu na koju se primenjuje
- skup elemenata za trening i metodu za postavljanje ovog skupa
- metod *learn* kojim inicira učenje

Naslednici ove klase implementiraju konkretne algoritme. Implementirani su svi razmatrani algoritmi.



Slika 21. Algoritmi za učenje

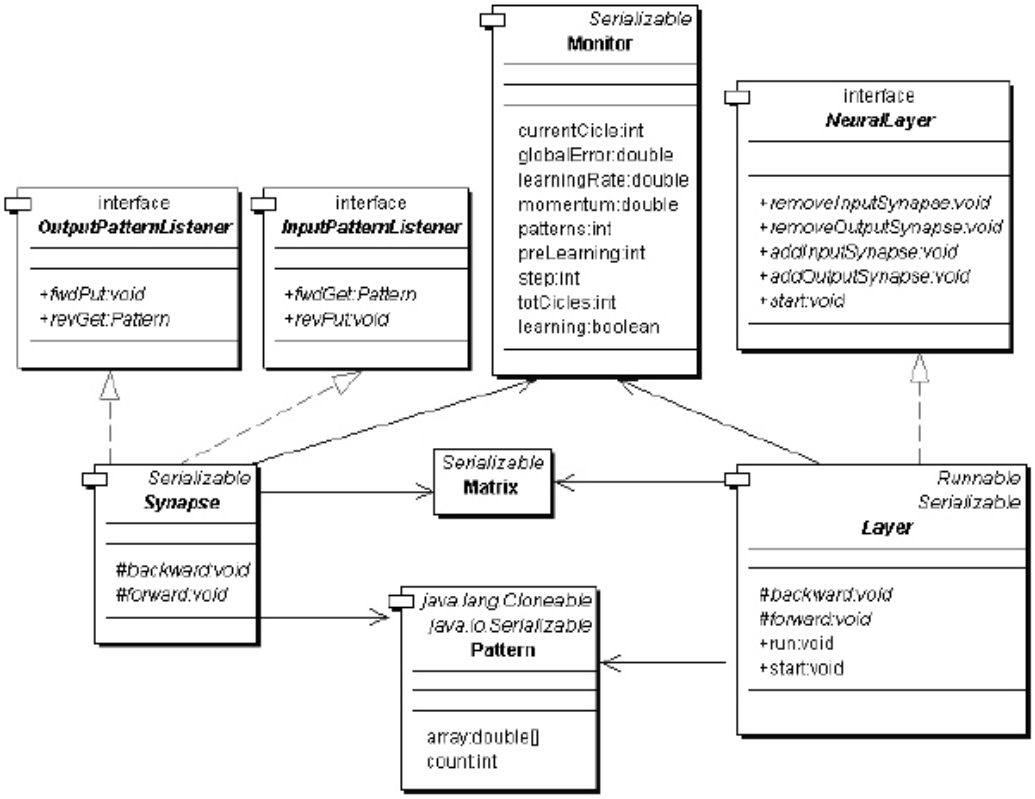


Slika 22. Globalni pogled - osnovne klase framework-a

4. POSTOJEĆI APLIKACIONI OKVIRI ZA RAZVOJ NM

4. 1. JOONE

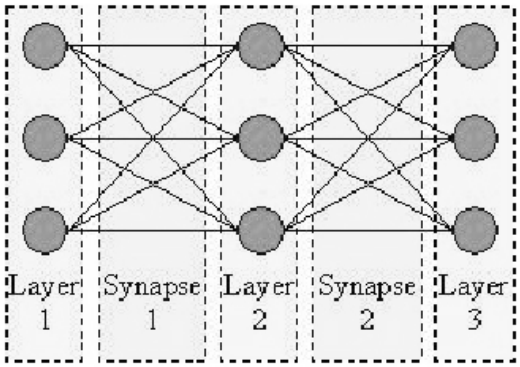
JOONE - Java Object Oriented Neural Engine
Aplikacioni okvir za razvoj neuronski mreža kod koga je osnovna ideja da se omogući paralelno testiranje i trening više mreža kako bi se odabrala najbolja za određeni problem.
Akcenat je dat na distribuiranoj simulaciji zbog potrebe paralelnog izvršavanja. Osnovne klase su *Layer* i *Synapse* koje modeliraju sloj neurona i težinske veze između slojeva respektivno.
Klasa *Layer* podržava izračunavanje u posebnom procesu i time je podržana distribuiranost.
Klasa *Monitor* je kontroler koji prati i upravlja radom slojeva.
Na slici je dat dijagram osnovnih klasa JOONE-a.



Slika 23. Dijagram klasa framework-a JOONE

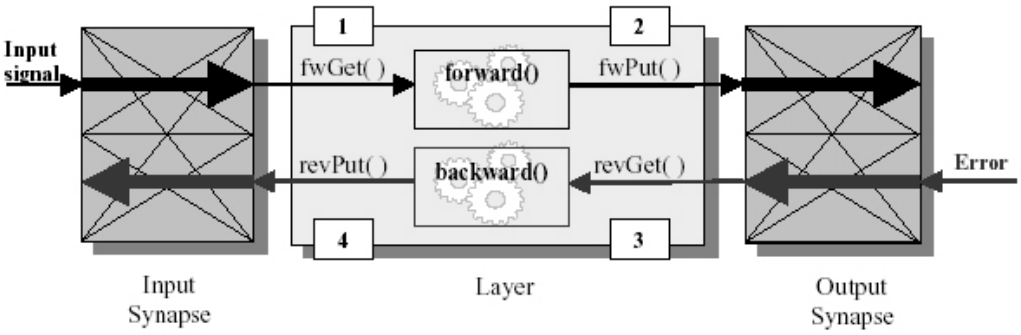
U strukturnom pogledu ima sličan pristup kao programski paket NeuroSolutions jer je sloj *(Layer)* osnovni element NM. Time je pojednostavljena konstrukcija mreža, međutim sve ćelije u sloju imaju iste karakteristike i ne može se kontrolisati struktura i funkcionalnost na ćelijskom nivou. Različiti slojevi nasleđuju osnovnu klasu i implementiraju specifične funkcije, koje bi po mom mišljenju trebalo da budu pridružene neuronima iz prethodno navedenih razloga .

NM se kreira tako što se slojevi neurona i sinaptičkih težina 'slažu' jedan na drugi.

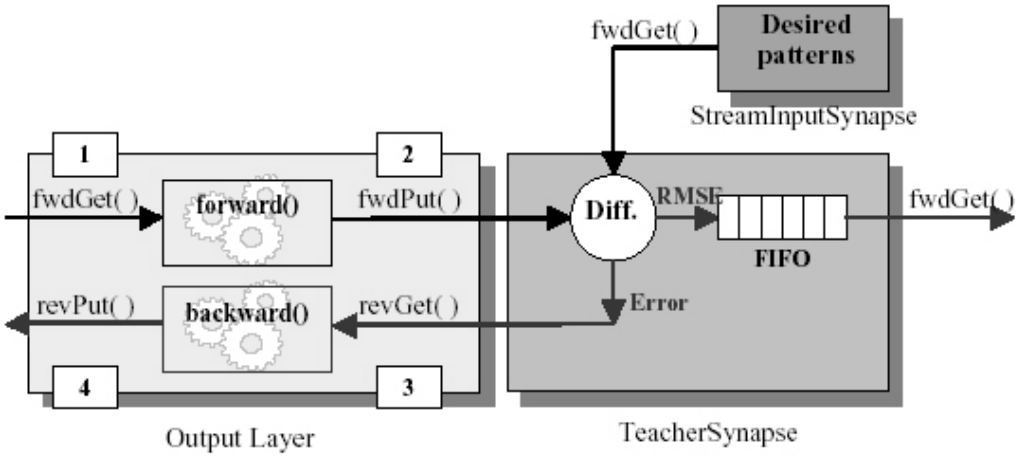


Slika 24. JOONE model neuronske mreže

Dat je detaljniji princip funkcionisanja klase Layer.

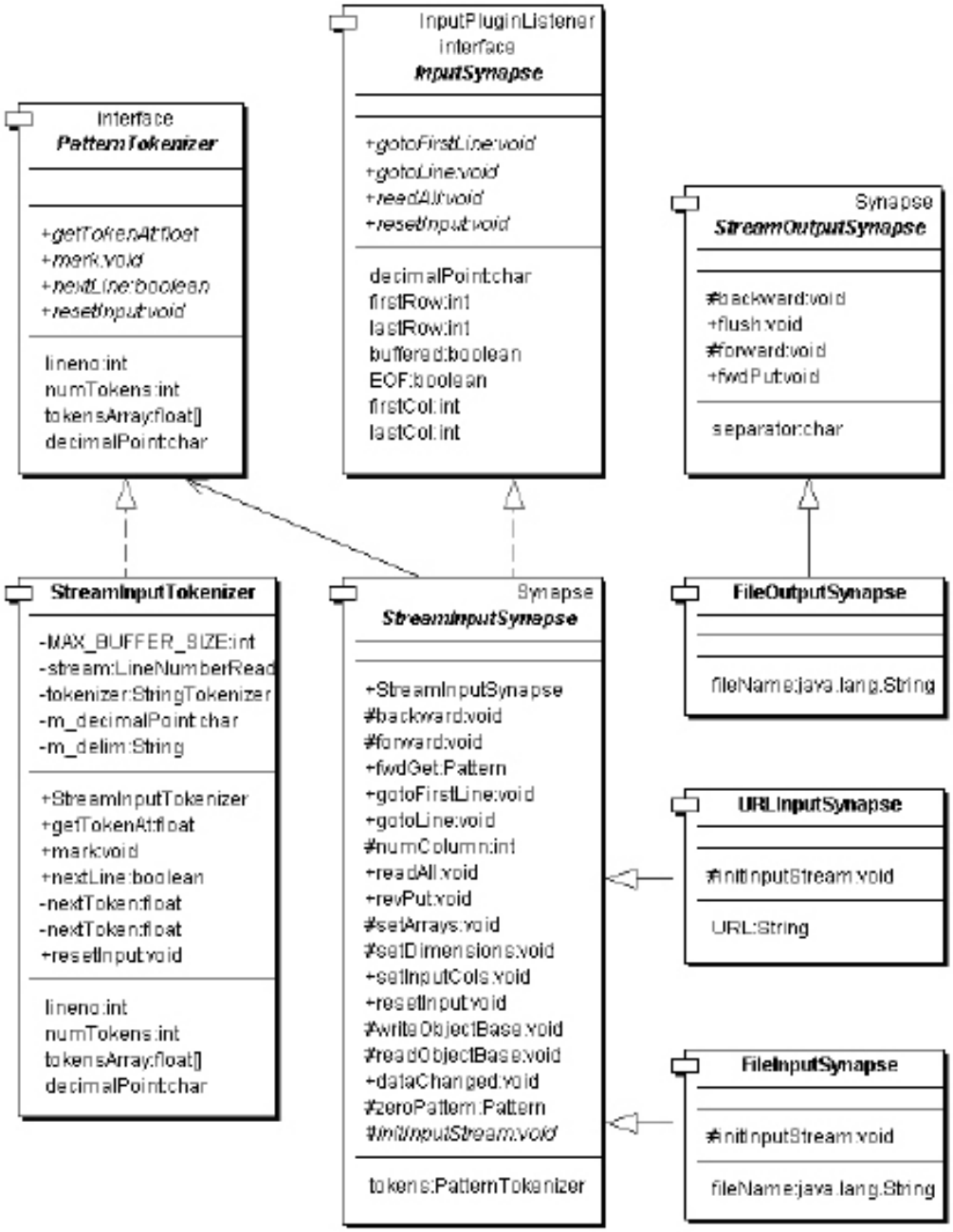


Slika 25. Prostiranje signala kroz sloj



Slika 26. Trening

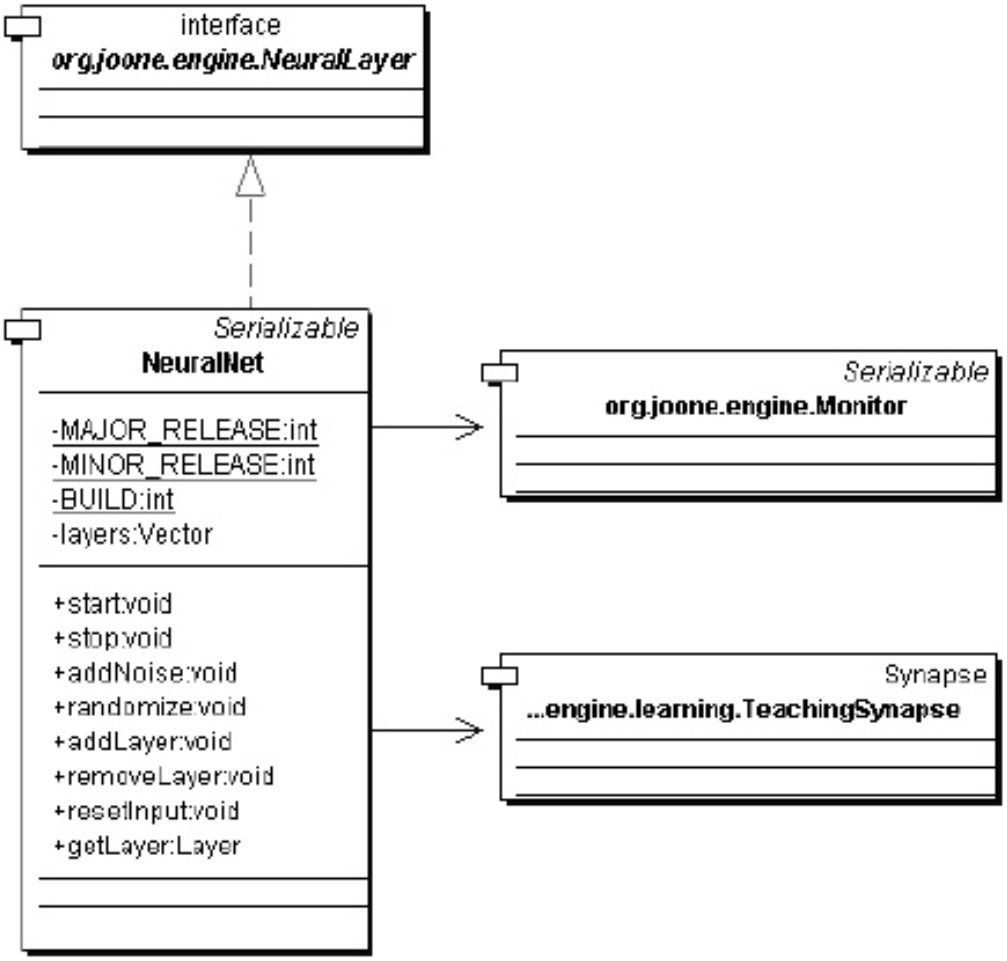
JOONE ima jako dobro rešenje za ulazni/izlazni interfejs tako što iz osnovnih ulazno/izlaznih klasa izvodi specifične klase koje odgovaraju ulazu iz fajla ili URL-a, odnosno izlazu u Excel fajl ili double matricu.
Implementiran je interfejs za finansijske izveštaje sa Yahoo-a.



Slika 27. Ulazno/izlazne klase

.

Dat je dijagram glavne klase *NeuralNetwork* koja predstavlja kontejner za slojeve i agregira kontroler i učitelja.



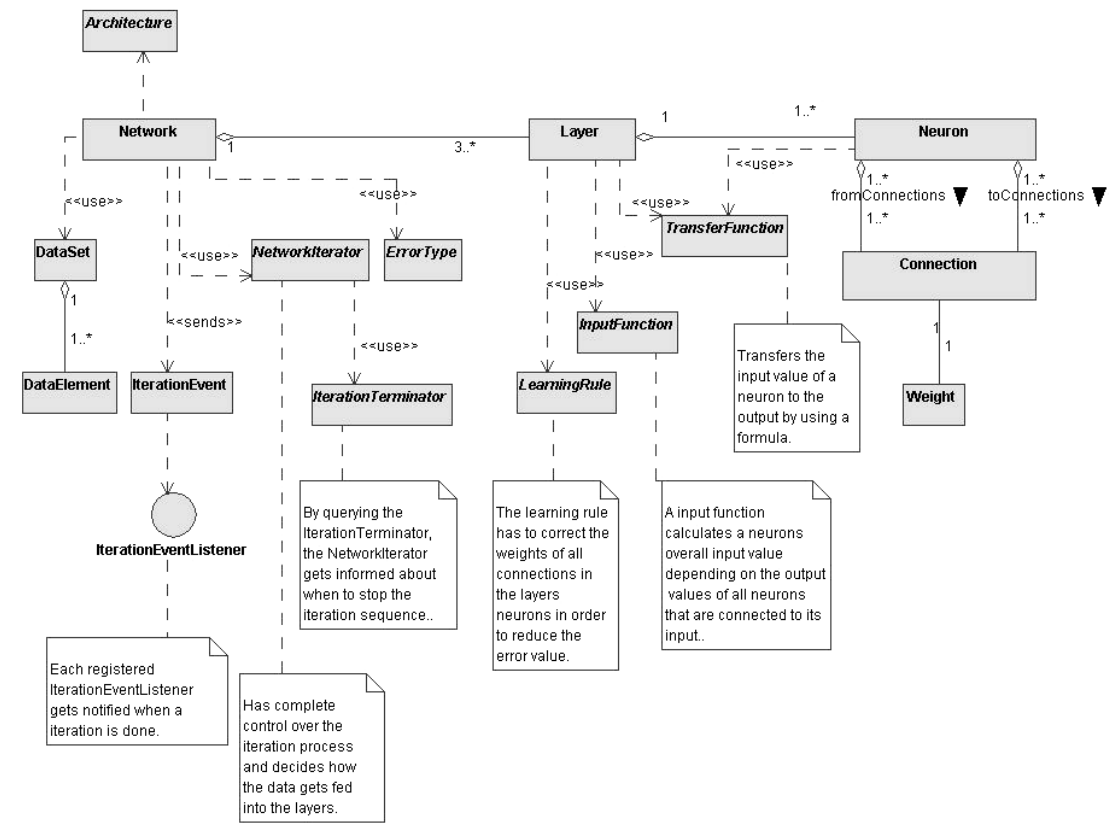
Slika 28. Klasa NeuralNetwork

U dokumentaciji se kaže da okvir podržava sve arhitekture, međutim detaljnije su obrađene samo mreže sa prostiranjem signala unapred sa Backpropagation algoritmom.

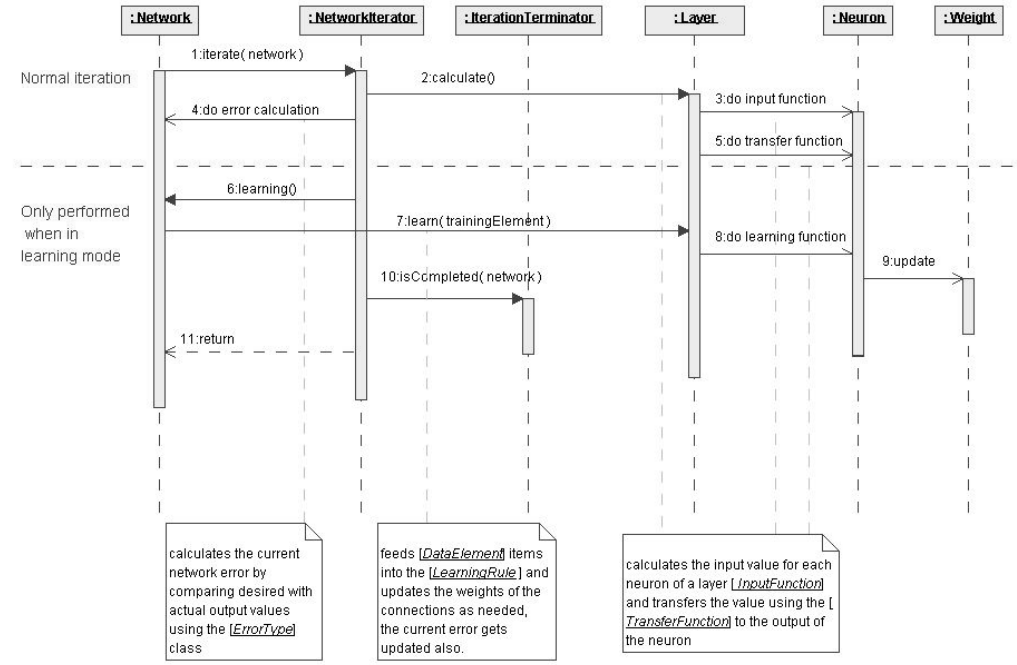
4. 2.OPENAI NNET

Java NeuralNet je projekat OpenAi grupe i predstavljao je uzor u početnim fazama razvoja. Po mom mišljenju je veoma dobro koncipiran aplikacioni okvir samo što je ostao na nivou apstrakcije - nisu razvijene biblioteke konkretnih klasa.

Dati su glavni dijagrami klasa i sekvenci.



Slika 29. Dijagram osnovnih klasa

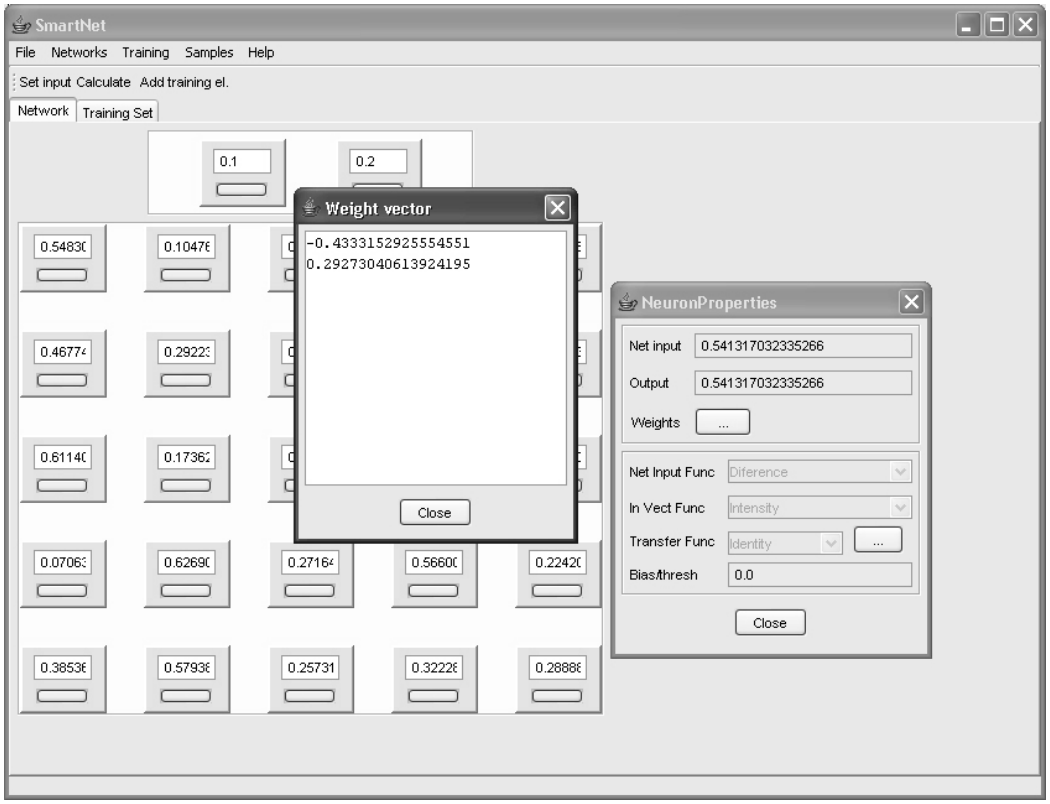


Slika 30. Dijagram sekvenci jedne iteracije

Neuroph je u strukturnom pogledu veoma sličan, s tim što su osnovne funkcionalnosti svih klasa proširene u skladu sa zahtevima konkretnih modela, i sadrži gotove modele.

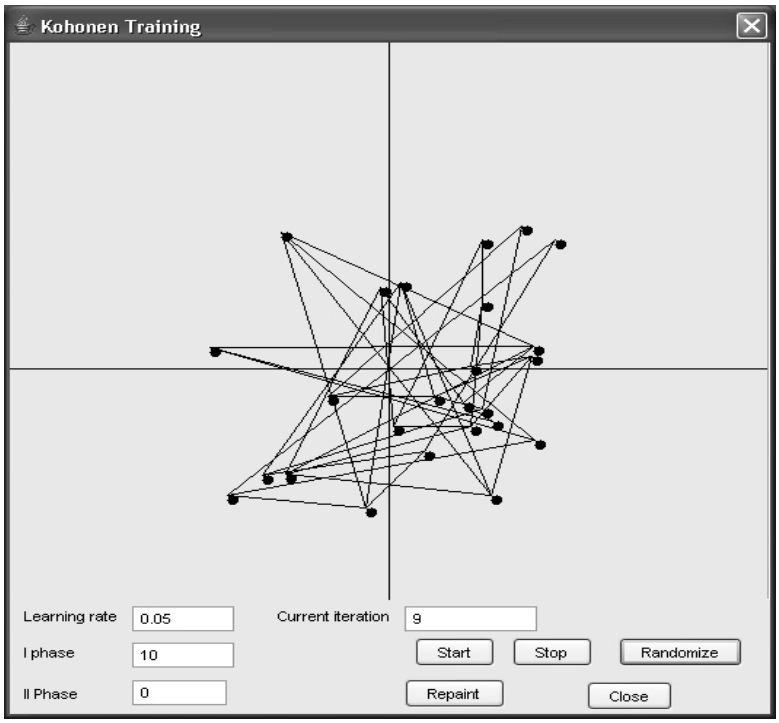
5. PRIMENA

Na realizovanom aplikacionom okviru izgrađena je aplikacija SmartNet, koja demonstrira osnovne mogućnosti okvira u radu sa različitim neuronskim mrežama. Aplikacija omogućava kreiranje razmatranih NM putem čarobnjaka , a zatim trening mreže. Tokom kreiranja i treninga moguće je praćenje i kontrolisanje svih relevantnih parametara mreže i procesa preninga.

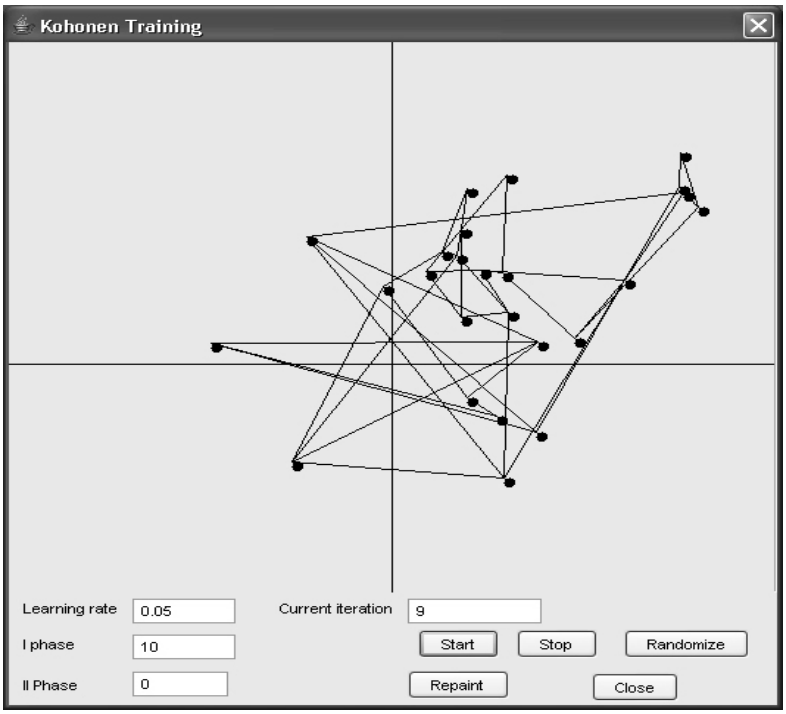


Slika 31. Aplikacija SmartNet

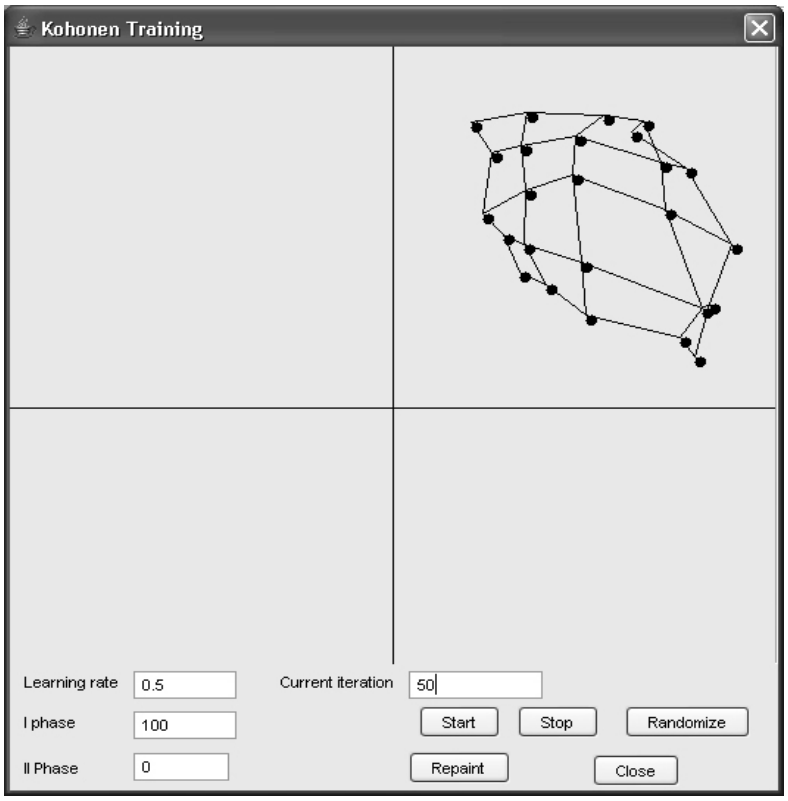
Od podržanih NM najzanimljivija je Kohonenova samoorganizujuća mreža, koja je prikazana na slici 31.
Osnovna funkcionalnost mreže je da uređuje skup visokodimenzionalnih ulaznih vektora. Suština navedene funkcionalnosti se može najbolje uočiti iz primera koji je implementiran u sklopu aplikacije [Samples> Kohonen].
Primer demonstrira kako od početnog neuređenog skupa slučajnih vektora mreža kreira uređenu strukturu.
Evolucija mreže je prikazana u nekoliko faza tokom treninga: na početku, posle 10 iteracija i posle 50 iteracija.



Slika 32. Vektori težina ćelija pre početka treninga



Slika 33. Vektori težina ćelija nakon 10 iteracija treninga



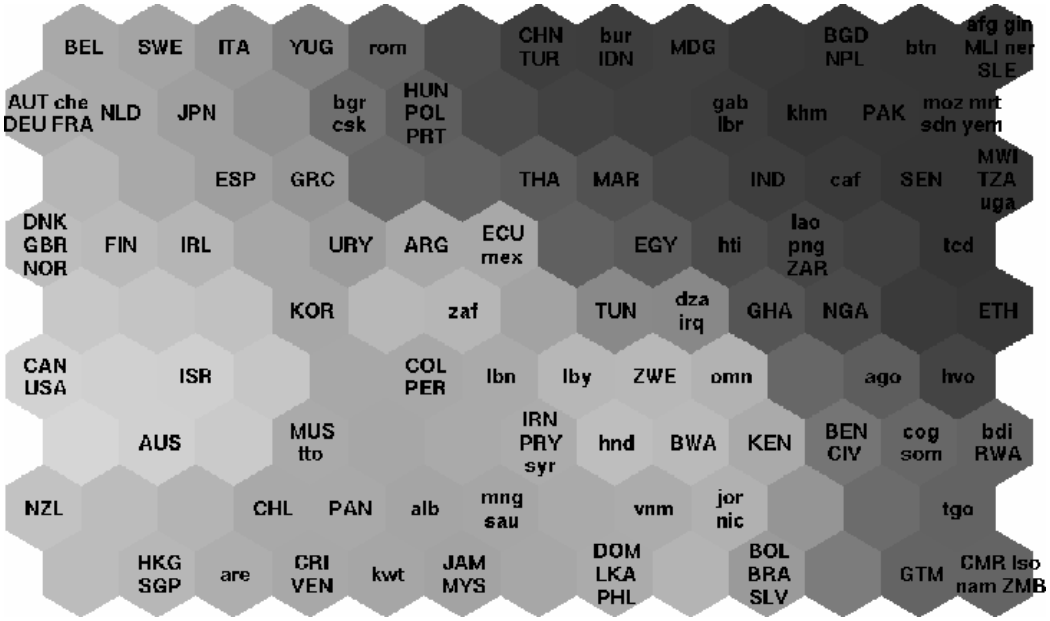
Slika 34. Vektori težina ćelija nakon 50 iteracija treninga

Ova mreža ima veliku praktičnu primenu u mnogim oblastima, za rešavanje problema vezanih za klasifikaciju, prepoznavanje, optimizaciju, vizuelizaciju i interpretiranje velikih skupova visokodimenzionalnih podataka. Poznate primene ove mreže su

1) Svetska mapa životnog standarda
2) WebSom

Svetska mapa životnog standarda

Ovde je SOM (samo-organizujuća mapa) upotrebljena za prikazivanje složenih relacija u statističkim podacima. Skup za trening su bili podaci o standardu izabranih zemalja. Podaci su se odnosili na 39 parametara kao što su zdravlje, obrazovanje, ishrana itd. To znači da je dimenzija ulaznog vektora 39. Upotrebljen je dvodimenzionalni sloj mape i nakon treninga u sloju mape grupisale su se zemlje sa sličnim parametrima. Svakoj ćeliji je zatim dodeljena određena boja, tako da svaka susedna ćelija predstavlja blagi prelaz odnosno nijansu svojih suseda.
Dobijene boje su zatim prenešene na geografsku kartu i tako je dobijena svetska mapa životnog standarda.

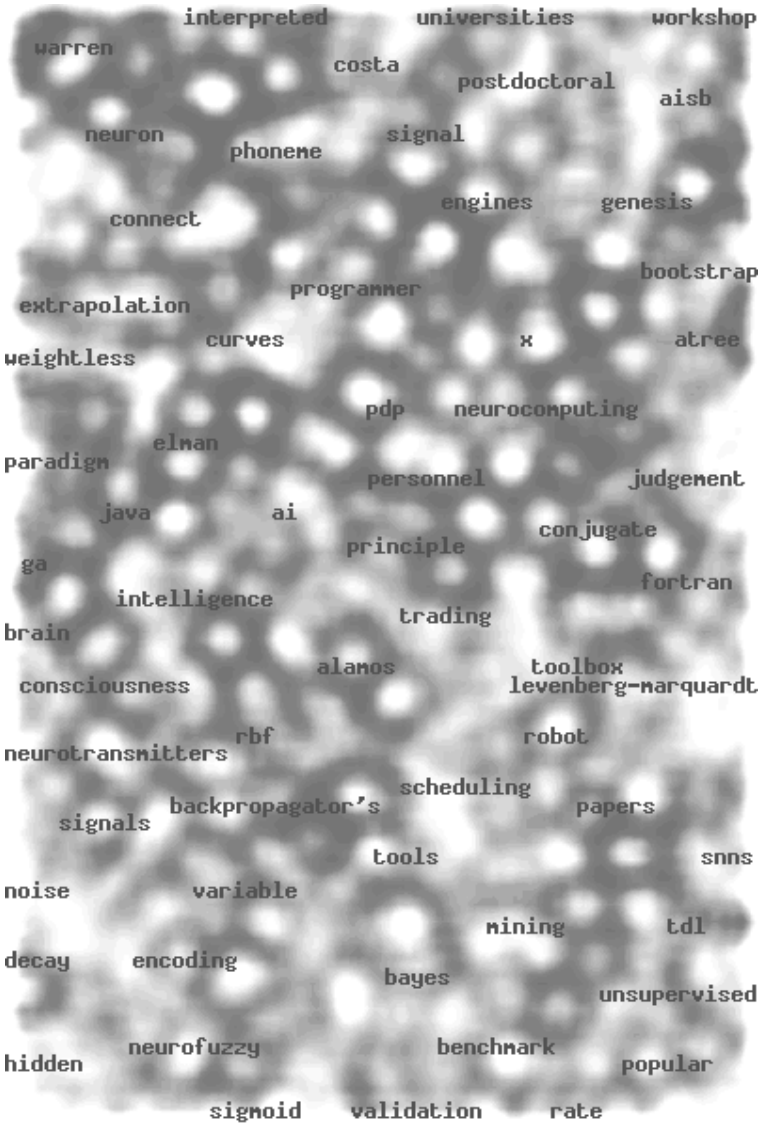


Slika 35. Zemlje uređene pomoću SOM

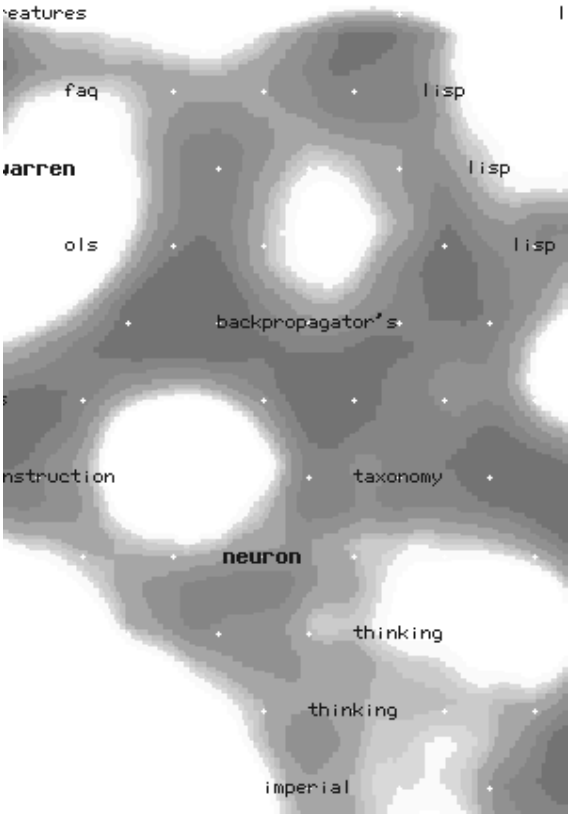
Slika 36. Svetska mapa životnog standarda

WebSom

WebSom je projekat u kome je samoorganizujuća mreža upotrebljena za pretraživanje dokumenata na Internetu. Osnovni princip je da se povezani (slični) dokumenti nalaze 'jedni blizu drugih'. Koriste se dve SOM. Jedna za predstavljanje reči i jedna za predstavljanje dokumenata. Dokumenti se razvrstavaju u kategorije na osnovu reči koje sadrže u naslovu. Projekat je započet 1996 i još uvek je u razvoju.



Slika 37. WebSom – I nivo



Slika 38. WebSom – II nivo

6. ZAKLJUČNA RAZMATRANJA

Razvijeni aplikacioni okvir u potpunosti je odgovorio na postavljene zahteve. Aplikacija SmartNet demonstrira osnovne mogućnosti okvira i NM izvedenih iz njega. Uspešno su realizovane dve najčešće korišćene mreže: višeslojni perceptron i samo-organizujuća mapa. Pored njih, okvir se može koristiti bez ikakvih izmena za izvođenje mreža bilo koje druge arhitekture što je pokazano na primeru Hopfieldove mreže.
Osnovne klase iste su za sve NM, a razlike su na nivou implementacije konkretnog modela.Pored osnovnih klasa definisane su i pomoćne (utility) klase koje uglavnom pojednostavljuju korišćenje osnovnih i obezbeđuju dodatne funkcionalnosti vezane za realizaciju konkretnih aplikacija.

Aplikacioni okvir je primenljiv u svim situacijama kada je za rešavanje određenog problema potrebna neuronska mreža. To su problemi koje karakterišu:

1) nelinearnost
2) visokodimenzionalnost
3) šum i neprecizni podaci
4) nedostatak odgovarajućeg matematičkog rešenja ili algoritma

Ponašanje karakteristično za neuronske mreže takođe upućuje na mogućnosti primene. Kao najbitnije stavke se izdvajaju:

- Učenje
- Asocijacija
- Klasifikacija
- Generalizacija
- Optimizacija

Dalji razvoj okvira će biti usmeren na praktičnu primenu. Cilj je razvoj gotovih komponenti istreniranih za određene probleme.
Što se tiče samog okvira, potrebno mu je dodati razne ulazno/izlazne interfejsa, npr. podrška za ulaz preko mreže, iz baze podataka, analizu dokumenata i podataka u različitim formatima, grafičko predstavljanje rezultata itd.
XML se može upotrebiti za specifikaciju mreže i rezultata.Potrebno je omogućiti serializaciju svih objekata, čime se omgućava slanje gotovih komponenti preko mreže.
Uz navedena poboljšanja razvijeni aplikacioni okvir bi našao veliku primenu u tehnologijama kao što su inteligentni agenti, obrada jezika, reprezentacija i otkrivanje znanja.

PRILOG

DOKUMENTACIJA ZA NAJVAŽNIJE KLASE APLIKACIONOG OKVIRA

neuroph.nnet

Class NeuralNetwork

public class **NeuralNetwork**
extends Observable

Neuronska mreza

Constructor Detail

NeuralNetwork

`public` **`NeuralNetwork`**`()`

Inicijalizuje praznu neuronsku mrezu

Method Detail

addLayer

`public void` **`addLayer`**`(Layer layer)`

Ubacuje u mrezu zadati sloj neurona

Parameters:

`layer` - Layer Sloj neurona koji treba dodati

---

addLayer

```
public void addLayer(int idx,
                     Layer layer)
```

Ubacuje zadati sloj neurona na odredjenu poziciju

Parameters:

`layer` - Layer Sloj neurona koji treba dodati
`idx` - int Pozicija u mrezi na koju treba ubaciti sloj

---

removeLayer

`public void` **`removeLayer`**`(Layer layer)`

Izbacuje sloj iz mreze

Parameters:

`layer` - Layer Referenca na sloj koji treba izbaciti

---

removeLayerAt

`public void **removeLayerAt**(int idx)`

Izbacuje iz mreze sloj na zadatoj poziciji

Parameters:

`idx` - int Pozicija na kojoj se nalazi sloj koji se izbacuje

---

layers

`public Enumeration **layers**()`

Vraca interfejs za sekvencijalni pristup slojevima u mrezi

Returns:

Enumeration Interfejs za pristup celijama u sloju

---

getLayers

`public Vector **getLayers**()`

Vraca referencu na kolekciju slojeva

Returns:

Vector Kolekcija slojeva

---

layerAt

`public Layer **layerAt**(int idx)`

Vraca referencu na sloj koji se nalazi na zadatoj poziciji

Parameters:

`idx` - int Pozicija na kojoj se nalazi trazeni sloj

Returns:

Referenca na trazeni sloj

---

indexOf

`public int **indexOf**(Layer layer)`

Vraca poziciju (indeks) sloja

Parameters:

`layer` - Layer Sloj ciji se indeks trazi

Returns:

int indeks sloja

---

layerNum

`public int **layerNum**()`

Vraca broj slojeva u mrezi

Returns:

Broj slojeva u mrezi

---

setInput

`public void **setInput**(Vector inVect)`

Zadaje ulaz mrezi

Parameters:

`inVect` - Vector Ulazni vektor

---

getOutput

`public Vector **getOutput**()`

Vraca izlaz mreze

Returns:

Vector Izlazni vektor

---

calculate

`public void **calculate**()`

Racuna izlaz mreze

---

learn

`public void **learn**(TrainingSet trainingSet)`

Inicira ucenje sa zadatim skupom elemanata

Parameters:

`trainingSet` - TrainingSet - Skup elemenata za trening

---

stopTraining

`public void **stopTraining**()`

Zaustavlja trening

---

getTypeId

`public Integer **getTypeId**()`

Vraca kod tipa mreze

Returns:

Integer kod tipa mreze

---

setTypeId

`public void **setTypeId**(int typeId)`

Postavlja kod tipa mreze

Parameters:

`typeId` - int - kod tipa mreze

---

getInputCells

`public Vector` **`getInputCells`**`()`

Vraca referencu na kolekciju ulaznih celija

Returns:

Vector

---

setInputCells

`public void` **`setInputCells`**`(Vector inputCells)`

Postavlja kolekciju ulaznih celija

Parameters:

`inputCells` - Vector - Kolekcija ulaznih celija

---

getOutputCells

`public Vector` **`getOutputCells`**`()`

Vraca referencu na kolekciju izlaznih celija

Returns:

Vector Kolekcija izlaznih celija

---

setOutputCells

`public void` **`setOutputCells`**`(Vector outputCells)`

Postavlja kolekciju izlaznih celija

Parameters:

`outputCells` - Vector - Kolekcija izlaznih celija

---

getLearningRule

`public LearningRule` **`getLearningRule`**`()`

Vraca algoritam za ucenje mreze

Returns:

LearningRule - algoritam za ucenje

---

setLearningRule

`public void` **`setLearningRule`**`(LearningRule learningRule)`

Postavlja algoritam za ucenje mreze

Parameters:

`learningRule` - LearningRule - algoritam za ucenje

---

notifyChange

`public void` **`notifyChange`**`()`

Obavestava observere o promeni stanja

---

neuroph.nnet

Class Layer

public class **Layer**

Sloj neurona

Constructor Detail

Layer

`public` **`Layer`**`()`

Inicijalizuje prazan sloj neurona

Method Detail

setParentNetwork

`public void` **`setParentNetwork`**`(NeuralNetwork parent)`

Postavlja referencu na nmrezu u kojoj se sloj nalazi

Parameters:

`parent` - nmreza u kojoj se sloj nalazi

---

parentNetwork

`public NeuralNetwork` **`parentNetwork`**`()`

Vraca referencu na nmrezu u kojoj se sloj nalazi

Returns:

referencu na nmrezu u kojoj se sloj nalazi

---

neurons

`public Enumeration` **`neurons`**`()`

Vraca interfejs za sekvencijalni pristup celijama u sloju

Returns:

interfejs za pristup celijama u sloju

---

addNeuron

`public void` **`addNeuron`**`(Neuron neuron)`

Dodaje neuron u sloj, kao poslednju celiju u nizu

Parameters:

`neuron` - celija koja se dodaje

---

addNeuron

```
public void addNeuron(int idx,
                      Neuron neuron)
```

Dodaje neuron u sloj, i ubacuje ga na zadati indeks

Parameters:

`idx` - indeks na koji se ubacuje

`neuron` - celja koja se dodaje

---

setNeuron

```
public void setNeuron(int idx,
                      Neuron neuron)
```

Zamenjuje celiju u sloju novom zadatom celijom

Parameters:

`idx` - indeks celija koja se zamenjuje

`neuron` - nova celija

---

removeNeuron

```
public void removeNeuron(Neuron neuron)
```

Izbacuje celiju iz sloja

Parameters:

`neuron` - celija koja se izbacuje

---

removeNeuronAt

```
public void removeNeuronAt(int idx)
```

Izbacuje iz sloja celiju sa zadatim indeksom

Parameters:

`idx` - indeks celije koja se izbacuje

---

neuronAt

```
public Neuron neuronAt(int idx)
```

Vraca neuron pod zadatim indeksom u sloju

Parameters:

`idx` - indeks neurona

Returns:

neuron sa zadatim indeksom

---

indexOf

```
public int indexOf(Neuron neuron)
```

Vraca indeks celije

Parameters:

`neuron` - celija ciji se indeks trazi

Returns:

indeks celije

---

neuronNum

```
public int neuronNum()
```

Vraca broj neurona u sloju

Returns:

broj neurona u sloju

---

neuroph.nnet.neuron

Class Neuron

public class **Neuron**
extends Object

Constructor Detail

Neuron

```
public Neuron(InputFunction inFunc,
              TransferFunction transFunc)
```

Parameters:

`transFunc` - prenosna funkcija

Method Detail

calculate

```
public void calculate()
```

Racuna izlaz celije

---

setInput

```
public void setInput(double input)
```

Postavlja ukupni ulaz za celiju

Parameters:

`input` - vrednost ulaza

---

getNetInput

```
public double getNetInput()
```

Vraca ukupni ulaz za celiju

Returns:

ukupni ulaz za celiju

---

getOutput

```
public double getOutput()
```

Vraca izlaz celije

Returns:

izlaz celije

---

hasInputs

```
public boolean hasInputs()
```

Proverava da li celija ima ulaznih veza

Returns:

true ako ima ulaznih veza, false u suprotnom

---

inputs

`public Enumeration` **`inputs`**`()`

Vraca Enumeration interfejs za pristup ulaznim vezama

Returns:

interfejs za pristup ulaznim vezama

---

addInput

`public void` **`addInput`**`(Connection conn)`

Dodaje ulaznu vezu

Parameters:

`conn` - ulazna veza

---

removeInput

`public void` **`removeInput`**`(Neuron from)`

Izbacuje ulaznu vezu

Parameters:

`from` - ulazna celija sa kojom se prekida veza

---

setInputFunc

`public void` **`setInputFunc`**`(InputFunction in)`

Postavlja ulaznu funkciju

Parameters:

`in` - ulazna funkcija

---

setTransferFunc

`public void` **`setTransferFunc`**`(TransferFunction trans)`

Postavlja prenosnu funkciju

Parameters:

`trans` - prenosna funkcija

---

inputFunc

`public InputFunction` **`inputFunc`**`()`

Vraca ulaznu funkciju

Returns:

ulaznu funkciju

---

transferFunc

`public TransferFunction` **`transferFunc`**`()`

Vraca prenosnu funkciju

Returns:

prenosnu funkciju

---

setParentLayer

```
public void setParentLayer(Layer parent)
```

Postavlja referencu na sloj u kome se celija nalazi

Parameters:

`parent` - referenca na sloj u kome se celija nalazi

---

getParentLayer

```
public Layer getParentLayer()
```

Vraca referencu na sloj u kome se celija nalazi

Returns:

Layer sloj u kome se celija nalazi

LITERATURA

[1] *Introducing Neural Networks*, Alison Carling, Sigma Press, Wilmslow, United Kingdom (1992)
[2] *Neural Networks for Intelligent Signal Processing*, Anthony Zaknich, World Scientific Publishing Co. Pte Ltd. Singapore (2003)
[3] *Neural and Adaptive Systems: Fundamentals Through Simulations*, Principe, Euliano, and Lefebvre, John Wiley and Sons (1999)