UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA

Zoran V. Ševarac

SOFTVERSKO INŽENJERSTVO INTELIGENTNIH SISTEMA

doktorska disertacija

Beograd, 2012.

UNIVERSITY OF BELGRADE
FACULTY OF ORGANIZATIONAL SCIENCES

Zoran V. Ševarac

SOFTWARE ENGINEERING OF INTELLIGENT SYSTEMS

Doctoral Dissertation

Belgrade, 2012.

Mentor:

dr Vladan Devedžić,
redovni profesor, Fakultet organizacionih nauka, Univerzitet u Beogradu

Članovi komisije:

dr Jelena Jovanović,
docent, Fakultet organizacionih nauka, Univerzitet u Beogradu

dr Dragan Gašević,
vanredni profesor, Fakultet za računarstvo i informacione sisteme, Atabaska univerzitet, Edmonton, Kanada

Datum odbrane: _____

Posvećeno sinu Marku

Softversko inženjerstvo inteligentnih sistema

Rezime:

U ovoj disertaciji istražena je primena savremenih metoda softverskog inženjerstva u razvoju softvera otvorenog koda u oblasti inteligentnih sistema. Razvijen je opšti model softverskog frejmvorka i date su preporuke za vođenje projekata otvorenog koda u toj oblasti. Model frejmvorka i preporuke za vođenje projekata razvijeni su, i potvrđeni u praksi kroz razvoj dodatnih funkcionalnosti jednog izabranog softverskog frejmvorka za neuronske mreže. Izabrani softverki frejmvork za neuronske mreže je trenutno jedan od vodećih softvera u toj oblasti u svetu.

Predložene metode razvoja softvera su opšteg karaktera, primenljive su i u drugim oblastima inteligentnih sistema, i odgovaraju na aktuelne probleme i specifičnosti razvoja softvera u tim oblastima. Problemi i specifičnosti se javljaju kao posledica toga što se često radi o softveru istraživačke namene koji treba da bude primenljiv i za praktične probleme, i kao posledica složenosti softvera koji istovremeno treba da bude jednostavan za korišćenje i modifikaciju.

Predložene metode razvoja odnose se na specifikaciju zahteva, planiranje razvoja, projektovanje, implementaciju i održavanje, i date su u vidu preporuka koje su se pokazale kao dobra praksa.

Pored ovog teorijskog i metodološkog doprinosa, kroz disertaciju su razvijena i konkretna unapređenja izabranog frejmvorka iz oblasti inteligentnih sistema, koja značajno unapređuju njegovu praktičnu primenu.

Ključne reči: *razvoj softvera, softverski frejmvork, softver otvorenog koda, inteligentni sistemi.*

Naučna oblast: *računarske nauke*

Uža naučna oblast: *softversko inženjerstvo*

Software Engineering of Intelligent Systems

Abstract:

This dissertation survey the application of modern software engineering methods for open source software development in the field of intelligent systems. Main results are a general model of software framework and recommendations for managing open source projects in that field. Framework model and recommendations for the project management were developed and validated in practice through the development of additional features of a selected software framework for neural networks. Selected neural network framework is one of the leading Java open source neural network frameworks.

Suggested software development methods are also applicable in other fields of intelligent systems, and they are response to problems and specific requirements in this field. Most problems are due the fact that it is often software for research purposes, that should also be applicable for practical problems, and complexity of the software that also needs to be easy to use and modify.

Proposed development methods are related to requirements specification, planning, design, implementation and maintenance. They are given in the form of recommendations that have been proven as a good practice.

In addition to these theoretical and methodological contributions, this dissertation also provides practical contribution in the form of new features for one selected framework in the field of intelligent systems, which greatly enhance its practical application.

Keywords: *software development, software framework, open source software, intelligent systems.*

Scientific field: *computer science*

Narrow scientific field: *software engineering*

SADRŽAJ

1. UVOD

1.1. Predmet istraživanja

Predmet istraživanja ove disertacije je primena principa softverskog inženjerstva u razvoju inteligentnih računarskih programa, posebno softverskih *framework-a* koje se koriste u razvoju inteligentnih sistema.
Osnovna motivacija za istraživanje je nastojanje da se sistematizuju iskustva u razvoju ove vrste softvera, da se sagledaju specifičnosti u planiranju i upravljanju razvojem, identifikuju glavni problemi i na taj način definiše jedan objedinjeni pristup za razvoj inteligentnih računarskih sistema, zasnovan na savremenim principima softverskog inženjerstva. Pri tome, uzeto je u obzir da oblast inteligentnih sistema prate sledeće specifičnosti koje takođe utiču na razvoj softvera u toj oblasti:

- istraživanja su u toku, postoji veliki broj otvorenih pitanja;
- multidisciplinarna priroda problema;
- vrlo dinamična oblast – brze promene, nova rešenja i tehnologije;
- rešavaju se veoma složeni problemi često nedovoljno dobro definisani.

Pored metoda za specifikaciju zahteva, analizu i projektovanje, analizirani su organizacija tima, planiranje funkcionalnosti, definisanje programskog interfejsa, upravljanje verzijama, testiranje, strategija objavljivanja i dokumentovanje softvera.
Sve navedeno prikazano je na primeru razvoja dodatnih funkcionalnosti za jedan izabrani softverski *framework*, iz oblasti inteligentnih sistema.

1.2. Formulacija problema

Postoji veliki broj softverskih projekata otvorenog koda koji istraživačima i kompanijama nude razna rešenja iz oblasti inteligentnih sistema. Projekti ovog tipa nastaju na univerzitetima, kreiraju ih kompanije, a često ih pokreću pojedinci ili neformalne grupe. Mnogi od njih ostvarili su veliki uspeh, postali su široko prihvaćeni, dobili podršku značajnih kompanija i praktično se nametnuli kao standard u svojoj oblasti. Ovi projekti uspešno su ostvarili svoj cilj i obezbedili stabilan višegodišnji razvoj. Međutim, mnogo veći broj projekata nije uspešno dostigao svoj cilj, negde usput se zaglavio i ugasio.
U okviru ovog istraživanja analizirani su uspešni projekti iz oblasti inteligentnih sistema, kako bi se došlo do odgovora na sledeća pitanja:

1. Na koji način kreirati i definisati projekat otvorenog koda iz oblasti inteligentnih sistema, kako bi se obezbedio stabilan razvoj i uspešno dostigli ciljevi projekta?
2. Koje paterne koristiti u dizajnu softvera, kao i da li je moguće definisati skup paterna za kreiranje softverskih *framework-a*?
3. Koje su ključne opšte karakteristike softvera od kojih zavisi uspeh projekta?
4. Kako organizovati i upravlljati razvojem projekta ovog tipa?

1.3. Ciljevi istraživanja

Opšti cilj disertacije je definisanje jasnog okvira, metoda i principa za projektovanje i razvoj softverskih *framework-a* otvorenog koda iz oblasti inteligentnih sistema. Na taj način disertacija daje doprinos metodologiji razvoja softvera iz oblasti inteligentnih sistema.
Specifični ciljevi istraživanja su sledeći:

- Definisanje procesa razvoja softverskih *framework-a* otvorenog koda iz oblasti inteligentnih sistema;
- Sagledavanje životnog ciklusa softverskih *framework-a* otvorenog koda iz oblasti inteligentnih sistema;
- Pregled najvažnijih softverskih paterna koji se koriste u projektovanju softverskih *framework-a*, i njihovo sagledavanje u celini u kontekstu kreiranja softverskog *framework-a* iz oblasti inteligentnih sistema;
- Prikaz razvojnog okruženja u širem smislu, kao i konkretnih alata koji se koriste u softverskim projektima;
- Razvoj dodatnih funkcionalnosti za izabran *framework* otvorenog koda iz oblasti inteligentnih sistema, u kome će se primeniti sve prethodno navedeno.

1.4. Pregled sadržaja po poglavljima

U drugom poglavlju dat je pregled relevantnih oblasti za ovo istraživanje: životni ciklus softvera otvorenog koda, proces razvoja softvera, infrastruktura za razvoj softvera otvorenog koda, i nekoliko primera softverskih *framework-a* iz oblasti inteligentnih sistema.
U trećem poglavlju je opisan proces planiranja razvoja i definisanja zahteva na konkretnom primeru softverskog *framework-a* iz oblasti neuronskih mreža.
U četvrtom poglavlju izvršeni su analiza i projektovanje u vezi sa dodatnim funkcionalnostima datim u trećem poglavlju. Takođe je dat i opšti model *framework-a*, i definisani su principi projektovanja *framework-a*.
U petom poglavlju opisana je implementacija najvažnijih klasa koje realizuju dodatne funkcionalnosti za izabrani *framework* iz oblasti inteligentnih sistema.
U šestom poglavlju dati su primeri korišćenja, a u sedmom je izvršena evaluacija unapređenog softverskog framework-a iz oblasti inteligentnih sistema.
U osmom poglavlju data su zaključna razmatranja koja obuhvataju doprinos istraživanja, mogućnosti primene i dalje pravce razvoja.

2. PREGLED RELEVANTNIH OBLASTI

U ovom poglavlju opisani su ukratko osnovni principi softverskog inženjerstva, procesa razvoja softvera i izabrani sistemi otvorenog koda iz oblasti inteligentnih sistema relevantni za ovo istraživanje.
Opisan je životni ciklus softvera otvorenog koda, proces razvoja softvera uopšte sa osvrtom na bitne karakteristike značajne za razvoj softvera otvorenog koda, dat je prikaz razvojnog okruženja i raspoloživih alata na najvećem svetskom servisu za razvoj softvera otvorenog koda *SourceForge-u*. Nakon toga dat je uporedni pregled nekoliko softverskih *framework-a* otvorenog koda iz oblasti inteligentnih sistema.

2.1. Životni ciklus softvera otvorenog koda

Softver otvorenog koda je softver koji je dostupan u obliku izvornog programskog koda, pod uslovima koji su određeni licencom, koja korisnicima omogućava da proučavaju, menjaju i poboljšavaju taj softver [Feller, 2001]. Softverske projekte otvorenog koda obično razvija programerska zajednica na Internetu koju čini grupa zainteresovanih programera dobrovoljaca.
Tokom postojanja jednog softverskog projekta otvorenog koda mogu se identifikovati sledeće faze [Fogel, 2005]:

1. osnovna ideja;
2. inicijalni razvoj pre zvaničnog objavljivanja;
3. prvo objavljivanje softvera;
4. kreiranje zajednice korisnika i dalji razvoj softvera;
5. prestanak razvoja;

1. Osnovna ideja

Iza svakog softverskog projekta otvorenog koda stoji neka početna zamisao pojedinca ili grupe programera, kojom se rešava neki problem/potreba, ili služi za proučavanje određenog problema ili oblasti. Osnovna ideja treba da definiše softverski projekat u nekoliko dimenzija, i to [Fogel, 2005]:

- šta softver radi i čemu služi;
- kome je namenjen odnosno ko su potencijalni korisnici;
- da li postoji neki slični softver za istu stvar i koji;
- ako postoji šta su prednosti a šta su nedostaci rešenja koje se predlaže.

Kroz navedene elemente sagledava se cilj i smisao projekta koji se planira, odnosno definiše filozofija koja će biti osnov za formiranje programerske zajednice oko projekta.

2. Inicijalni razvoj

Inicijalni razvoj podrazumeva inicijalni rad na realizaciji osnovne ideje projekta, čiji rezultat treba da bude softver koji uspešno odgovara na predviđene zahteve i to prema principima koji su definisani filozofijom projekta.
Na inicijalnom razvoju su obično angažovani pojedinac ili manji tim, koji će biti nosioci projekta kada/ako zaživi.
Inicijalni razvoj treba da pokaže opravdanost projekta kao i da pokretač projekta poseduje sposobnosti potrebne za uspešnu realizaciju projekta.

3. Prvo objavljivanje softvera

Prvo objavljivanje softvera predstavlja pravi početak životog ciklusa softvera otvorenog koda. Objavljivanje na Internetu softver postaje javno dostupan za preuzimanje sa kompletnim izvornim kodom i dokumentacijom. Korisnicima je potrebno predstaviti softver, dati uspešne demo primere korišćenja, obezbediti potrebnu dokmentaciju i uputstva, i obezbediti podršku. Time se kreira osnova za dalji razvoj projekta i stavlja do znanja da se radi o ozbiljnom pokušaju pokretanja projekta.
U ovoj fazi veoma su dragocene povratne informacije od korisnika, jer one mogu da ukažu na najbolje pravce daljeg razvoja, odnosno na najpotrebnije funkcionalnosti i karakteristike softvera i najbrži način za privlačenje korisnika.

4. Kreiranje zajednice i dalji razvoj softvera

Da bi softverski projekat otvorenog koda zaživeo i opstao, neophodno je da se izgradi zajednica zainteresovanih programera i korisnika, koja će raditi na njegovom razvoju. To je moguće samo ako projekat ispuni planirane ciljeve, odgovori na zahteve korisnika i ispromoviše se na odgovarajući način..
Članovi zajednice mogu doprinositi projektu na razne načine i to [Fogel, 2005]:

- neposrednim učešćem u razvoju;
- pružanjem podrške korisnicima na forumu ili mailing listi;
- izradom dokumentacije i uputstava;
- testiranjem softvera;
- kreiranjem i održavanjem sadržaja Web sajta projekta;
- finansiranjem projekta.

Zajednice koje se formiraju oko projekata otvorenog koda obično imaju nekoliko članova koji čine uže jezgro (obično pokretači projekta) i one koji su povremeno aktivni. Zajednice oko projekata otvorenog koda predstavljaju njihov najznačajniji resurs, jer bez njih nema ni projekta. Sam razvoj je iterativan proces koji se sastoji iz [Fogel, 2005]:

- prikupljanja novih zahteva korisnika;
- planiranja i dodavanja novih funkcionalnosti;
- ispravljanja grešaka (*bug*-ova);

- raznih optimizacija i poboljšanja softvera;
- objavljivanja novih verzija softvera.

Kako se metodologija razvoja softvera otvorenog koda pokazala veoma uspešna u praksi, često se javljaju slučajevi kada kompanije koje se bave razvojem komercialnog softvera pokreću projekte otvorenog koda, 'otvaraju' programski kod svojih proizvoda, sponzorišu uspešne projekte i sl. Postoje razni modeli saradnje između organizacija koje se bave razvojem komercijalnog i onih koje se bave razvojem tzv. *slobodnog softvera otvorenog koda.*
Pri tom razne licence (npr. GPL[1], LGPL[2], BSD[3], Apache[4]) korisnicima daju različite nivoe slobode u korišćenju softvera, a autorima odgovarajuće nivoe zaštite njihovog dela [Laurent, 2004].

5. Prestanak razvoja

Postoje razni razlozi za ili prestanak razvoja nekog projekta otvorenog koda. Tipični uzroci su [Fogel, 2005]:

- Neuspeh u ostvarivanju osnovnih ideja projekta, zbog pogrešne postavke ideja ili u neuspeha u realizaciji.
- Softver je dospeo u stabilnu fazu kada u potpunosti ispunjava svoju namenu i nema više potrebe za daljim razvojem. U ovom slučaju projekat ne prestaje da postoji već samo prestaje njegov dalji razvoj.
- Zastarelost. Moguće je da usled pojave novih tehnologija određena rešenja data u projektu zastare u smislu da više ne podržavaju aktuelne tehnologije, da značajno zaostaju u performansama u odnosu na nova rešenja, ili da se pojave novi projekti koja nude bolja i savremenija rešenja.
- Raspad zajednice. Usled neslaganja, različitih vizija daljeg razvoja, gubitka interesovanja osnivača ili korisnka može doći do raspada zajednice oko projekta što praktično odmah dovodi do prestanka razvoja, mada je moguće i tzv. grananje *(fork)* projekta što znači da dolazi do podele u zajednici i da jedan deo zajednice nastavlja razvoj kao poseban projekat.

Nakon prestanka razvoja iz bilo kog od navedenih razloga, u budućnosti je moguće ponovno aktiviranje i nastavak razvoja.
Rizik od prestanka razvoja umanjuje se ako projekat dobije podršku veoma širokog kruga korisnika i uticajnih kompanija, i korisicima da veliku slobodu u korišćenju. Npr. ukoliko neka kompanija zasnuje razvoj svog komecijalnog proizvoda na korišćenju nekog *framework-a* otvorenog koda ona će imati interes da podržava razvoj tog *framework-a*.

[1] http://www.gnu.org/licenses/gpl.html
[2] http://www.gnu.org/licenses/lgpl.html
[3] http://www.freebsd.org/copyright/freebsd-license.html
[4] http://www.apache.org/licenses/LICENSE-2.0.html

2.2. Proces razvoja softvera

Proces razvoja softvera je skup faza i aktivnosti tokom razvoja softverskog proizvoda.Postoji nekoliko modela procesa razvoja softvera, koji na različiti način struktuiraju aktivnosti koje se odvijaju tokom tog procesa. U ovom poglavlju dat je kratak pregled aktuelnih modela procesa razvoja softvera, i njihovih glavnih karakteristika, koji su od interesa za ovaj rad.

2.2.1. Sekvencijalni proces razvoja (*Waterfall*)

Sekvencijalni ili *Waterfall* model razvoja softvera podrazumeva samo jedan prolaz kroz sledeće faze razvoja softvera [McConnell, 1996]:

1. specifikacija zahteva;
2. analiza;
3. dizajn;
4. implementacija;
5. testiranje;
6. isporuka i održavanje.

1. **Specifikacija zahteva**, podrazumeva jasno definisanje šta određeni softver treba da radi i koje karakterisike treba da ima.

2. **Analiza**, predstavlja sagledavanje elemenata zahteva i kreiranje odgovarajućeg konceptualnog modela koji predstavlja grubu skicu i osnovni logički model softverskog sistema [Jacobson, et. al., 1999].

3. **Dizajn,** predstavlja kreiranje modela softverskog sistema kojim se definiše struktura i interakcija između komponenti softverskog sistema. Model softverskog sistema se kreira na osnovu konceptualnog modela koji je nastao u fazi analize, korišćenjem nekog jezika za modeliranje. Najčešće korišćeni jezik za modeliranje je UML, kojim se softverski sistem opisuje pomoću skupa dijagrama (dijagram klasa, dijagram sekvenci, kolaboracioni dijagram, dijagram stanja, itd.)

4. **Implementacija**, podrazumeva generisanje/pisanje programskog koda na osnovu modela softverskog sistema kreiranog u fazi dizajna. Rezultat ove faze je izvršna vezija softvera koja obezbeđuje sve potrebne funkcionalnosti definisane u zahtevu.

5. **Testiranje** podrazumeva skup procedura i tehnika kojima se utvrđuje da li sve funkcionalnosti koje softver obezbeđuje ispravno rade u raznim mogućim scenarijima i graničnim slučajevima.

6. **Isporuka i održavanje** podrazumevaju predaju softvera korisniku na korišćenje, i intervencije po potrebi kako bi se otklonili eventualni problemi u radu, ili modifikovale postojeće funkcionalnosti kako bi se odgovorilo na nove zahteve.

Osnovne karakteristike sekvencijalnog procesa razvoja su [Thayer, et.al. 2002]:

1. kroz sve navedene faze procesa prolazi se redom, i kada se jedna faza završi više nema povratka na prethodu;
2. na sledeću fazu se prelazi tek pošto tekuća bude završena u potpunosti (do savršenstva);

Sekvencijalni model je prvi model procesa razvoja softvera koji je nastao po uzoru na proizvodne i građevinske projekte [Thayer, et.al. 2002]. Kod ovih projekata, maksimalno se izbegava vraćanje na prethodne faze pošto to predstavlja velike troškove (npr. ponovno projektovanje zgrade nakon što je izgrađena, ili dizajn automobila nakon što je proizvedena cela serija).
Upravo zbog toga sekvencijalni model nije odgovorio na specifične zahteve softverskih projekata, jer se oni bitno razlikuju od navedenih vrsta projekata.
Osnovni problem kod softverskih projekata je što nije moguće u startu predvideti sve zahteve i probleme, jer oni nisu u potpunosti poznati. U praksi se pokazalo da deo zahteva postaje poznat tek kada krajnji korisnik počne da koristi softver.
Drugi problem je velika složenost savremenih softverskih sistema, zbog koje nije moguće odjednom sagledati sve funkcionalnosti i isprojektovati odgovarajuće rešenje.
Odgovarajuće rešenje za ove probleme nađeno je uvođenjem iterativnog i inkrementalnog procesa razvoja softvera, koji je nastao na osnovu iskustva iz prakse, i koji je uvažio navedene specifičnosti

2.2.2. Iterativni i inkrementalni proces razvoja

Osnovni princip iterativnog i inkrementalnog procesa razvoja je da se razvoj planira u više ciklusa (iteracija), pri čemu se u svakoj iteraciji prolazi praktično kroz sve faze razvoja koje postoje i kod *waterfall* modela [Larman, 2003]. U svakoj iteraciji se dodaju nove funkcionalnosti i vrše modifikacije u dizajnu, Nakon implementacije vrši se testiranje onoga što je napravljeno, uzimaju se povratne informacije od krajnjih korisnika i na osnovu toga sagledavaju određena rešenja i planira dalji razvoj. Ovakav pristup omogućava da se u svakoj iteraciji primenjuju iskustva iz prethodne iteracije.
Osnovni problemi prisutni kod sekvencijalnog razvoja su [Larman, 2003]:

- korisnik najčešće ne zna sve zahteve;
- logika procesa nije poznata;
- nije moguće predvideti sve probleme u radu;
- nakon uvođenja odnosno početka korišćenja ubrzo se javljaju novi zahtevi.

Model iterativnog i inkrementalnog razvoja je kreiran tako da odgovara upravo na navedene probleme. U zavisnosti od složenosti problema razvoj se planira u nekoliko iteracija, i za svaku iteraciju se definišu aktivnosti i očekivani rezultati.
Ideja je da svaka iteracija nadograđuje i menja ono što je nastalo kao rezultat prethodne iteracije, i u svakoj iteraciji se dodaje više detalja.

2.2.3. Agilni razvoj softvera

Agilni razvoj softvera je skup metodologija zasnovanih na iterativnom i inkrementalnom modelu razvoja. Osnovni principi koje agilne metodologije potenciraju su [Beck, et. al., 2004]:

- veoma kratki razvojni ciklusi (iteracije)
- fleksibilni samoorganizujući razvojni timovi, sa ravnom organizacionom strukturom
- projekte kreiraju i vode visoko motivisani pojedinci
- brzo prilagođavanje promenama zahteva
- isporuka softvera koji funkcioniše na kraju svake iteracije
- softver koji radi je osnovno merilo napretka

Popularna agilna metodologija je tzv. ekstremno programiranje (*Extreme programmming*) [Beck, et al., 2004].

Sam naziv ektremno programiranje potiče od toga što su pojedine aktivnosti koje se smatraju ključnim za efikasan razvoj softvera potenciraju do ekstremnog nivoa. Praksa ekstremnog programiranja daje sledeće preporuke [Beck, et al., 2004]:

- težiti jednostavnom i jasnom programskom kodu;
- česta i detaljna analiza i preuređivanje programskog koda *(code review i refactoring)*;
- programiranje u paru;
- automatizovano testiranje kompletnog programskog koda;
- planiranje razvoja sa očekivanim promenama korisničkih zahteva tokom projekta;
- intenzivna komunikacija sa klijentom i između programera u okviru razvojnog tima.

Programiranje u paru, znači da za jednim računarom rade dve programera od kojih jedan radi neposredno za tastaturom i bavi se detaljima vezanim za pisanje programskog koda, a drugi posmatra širu sliku i proverava kod koji je napisao prvi programer. Pri tom programeri redovno menjaju uloge.

Refactoring podrazumeva izmene programskog koda pri čemu se ne menja funkcionalnost programa prema korisnicima, već se poboljšava dizajn programa. Poboljšanja mogu biti u vezi sa poboljšanjem čitljivosti koda, smanjenjem složenosti ili efikasnijem izvršavanju,

Pored navedenih postoje i mnoge druge tehnike i preporuke extremnog programiranja, za planiranje, upravljanje, i dizajn softvera, ali ovde su navedene one koje su od interesa prilikom analize razvoja softvera otvorenog koda, koji su razmatrani u ovom radu.

2.3. Infrastruktura za razvoj softvera

Savremeni sistemi za upravljanje razvojem softvera (software development management system) obezbeđuju kompletnu infrastrukturu za razvoj softvera. Ovi sistemi integrišu razne servise koji se koriste za razvoj softvera i upravljanje softverskim projektima, podržavaju proces razvoja, i kolaborativni rad učesnika na projektu putem Interneta [Feller, 2001].
Tipični servisi koje obezbeđuju sistemi za upravljanje razvojem softvera u svetu softvera otvorenog koda su [Fogel, 2005]:

1. upravljanje verzijama izvornog koda (*source code version control*);
2. praćenje bug-ova i zahteva za dodatnim funkcionalnostima (*bugs and features requests*);
3. podrška korisnicima - mailing lista, forumi i sl. (*support*);
4. upravljanje članovima razvojnog tima i njihovim dozvolama (*roles, permissions*);
5. servis za distribuciju/preuzimanje softvera (*download*);
6. Web sajt projekta;
7. statistike posećenosti sajta, aktivnosti na projektu, daunlouda softvera.

1. Upravljanje verzijama izvornog koda (*source code version control*)

Sistem za upravljanje verzijama izvornog koda je neophodan za rad više programera na istom projektu. On omogućava praćenje, objedinjavanje i sinhronizaciju izmena koje istovremeno nezavisno vrše članovi projektnog tima. Takođe omogućava upravljanje verzijama softvera (tzv. grananje, eng. *branch*) prilikom popravljanja *bug*-ova, dodavanja novih funkcionalnosti, objavljivanja softvera i sl. Na serveru za kontrolu verzija nalazi se razvojna verzija izvornog koda softvera na kojoj se prate sve izmene. Te izmene je na zahtev moguće objediniti, a zatim ažurirati razvojne verzije koda kod svih programera u timu.
Osnovne operacije koje obezbe]uju ovi sistemi su:

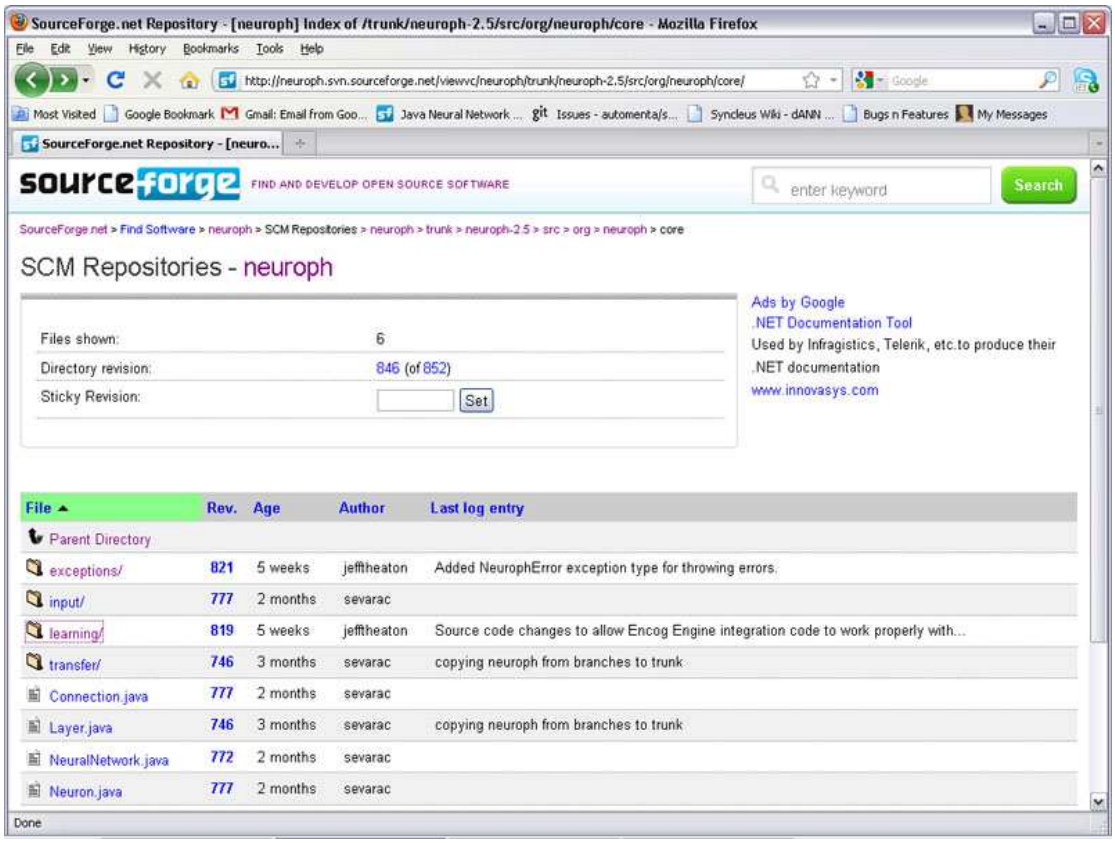
- *commit*, šalje izmene sa lokalnog računara na server za kontrolu verzija;
- *update*, ažurira kod na lokalnom raćunaru sa izmenama koje se nalaze na serveru za kontrolu verzija;
- *diff*, prikazuje razlike u verzijama izvornog koda na lokalnom računaru i serveru za kontrolu verzija;
- *revert*, vraćanje izmena na neku od prethodnih verzija izvornog koda;
- *branch*, kreiranje nove grane u razvoju;
- merge, objedinjavanje izmena iz dve različite verzije izvornog koda.

Savremena integrisana razvojna okruženja (IDE) omogućavaju jednostavno izvršavanje ovih operacija direktno iz razvojnog okruženja.

Na SourceForge-u nude se sledeći sistemi za upravljanje verzijama izvodnog koda:

- Subversion (staviti ovde i linkove na home page projekata)
- GIT
- Bazaar
- CVS
- Mercurial

Pored izvršavanja osnovnih operacija za upravljanje verzijama iz IDE-a, u okviru sistema za upravljanje razvojem softvera moguć je i pregled verzija izvornog koda putem Web interfejsa, kao i povezivanje ovog Web interfejsa sa tzv. Trac sistemom, što omogućava praćenje izmena vazanih za određene zadatake, ispravke bug-ova ili dodatnih funkcionalnosti. Na slici 1 dat je prikaz Web interfejsa za pregled verzija izvornog koda *Neuroph* projekta [Sevarac, 2006] na *Source Forge*-u.



Slika 1. Web interfejs za pregled verzija izvornog koda na *Source Forge*-u

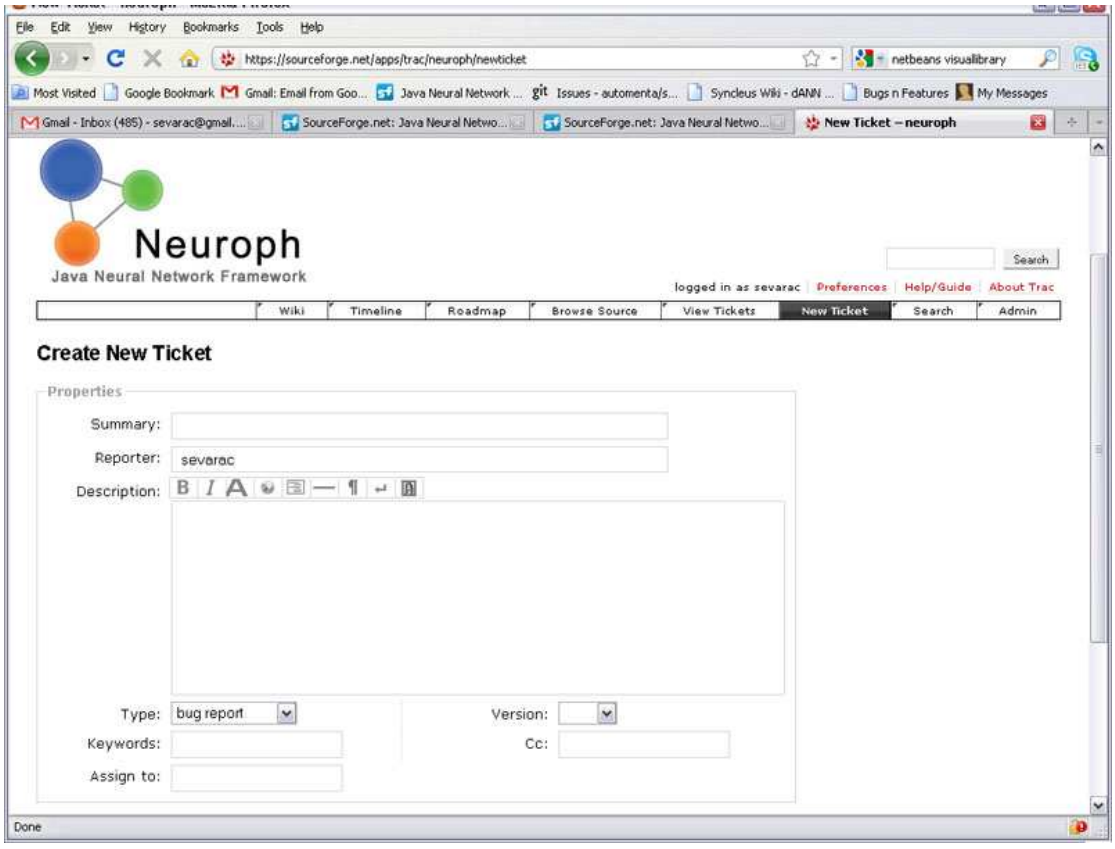
2. Praćenje bug-ova i zahteva za dodatnim funkcionalnostima (*bugs and features requests*)

Praćenje bug.ova i zahteva za dodatnim funkcionalnostima (i njihovo ispravljanje odnosno implementacija) spadaju među glavne aktivnosti tokom životnog ciklusa softverskog proizvoda, koje presudno utiču na njegov uspeh. Ukoliko se korisnicima ne

pomogne da prevaziđu probleme koje imaju, ili omogući da koriste softver na način koji im je potreban, oni ga neće ni koristiti već će potražiti alternativno rešenje.
Sistemi za praćenje bug-ova i zahteva za funkcionalnostima s jedne strane omogućavaju korisnicima softvera da putem predefinisanog formulara šalju informacije o svojim problemima odnosno zahtevima, a sa druge strane razvojnom timu da ih na sistematizovan način prati, razvrstava i rešava. Postoje razni sistemi koji obezbeđuju ovo, a na SourceForge-u se nude sledeći:

- Trackers (*SourceForge* specific)
- Mantis BT (http://www.mantisbt.org)
- Trac (http://trac.edgewall.org)

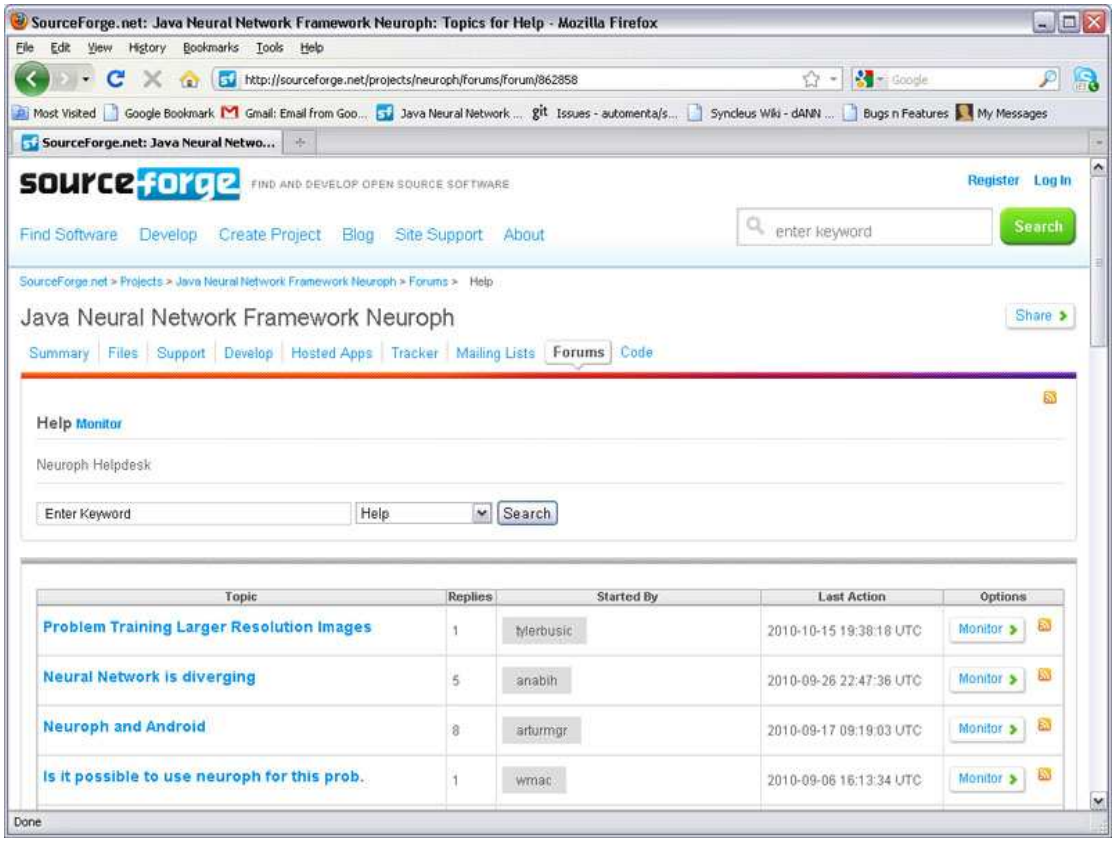
Trac sistem pored praćenja bug-ova i zahteva ta funkcionalnostima obezbeđuje i servis za upravljanje zadacima (tasks). Pomoću njega se vrši planiranje i dodeljivanje zadataka učesnicima na projektu. S druge strane svaki učesnik može da vidi koji su mu zadaci dodeljeni odnosno šta prema planu on treba da uradi.
Na slici 2 prikazan je formular za slanje zahteva za ispravku *bug*-ova ili dodatnu funkcionalnost za Neuroph projekat na *Source Forge-u*. Isti formular se može koristiti i za kreiranje i dodeljivanje zadataka.



Slika 2. Formular za slanje zahteva za ispravku greške
ili dodavanje nove funkcionalnosti

3. Podrška korisnicima (*support*)

Pored ispravljanja bug-ova i dodavanja novih funkcionalnosti koji predstavljaju osnovni vid podrške, korisnicima je potrebno obezbediti način da razmenjuju iskustava, konsultuju se, analiziraju određene primene i direktno komuniciraju sa razvojnim timom. Najpopularniji načini za ovo su mailing liste i forumi.
Informacije do kojih se dolazi na ove vidove komunikacije su veoma dragocene razvojnom timu, a pored toga ovi servisi su osnova za izgradnju zajednice korisnika i programera oko projekta. Veoma često zainteresovani korisnici koji su aktivni u diskusijama počinju da učestvuju i u razvoju.
U zavisnosti od broja korisnika, broja poruka i potreba može biti pogodno da se formira nekolko mailing lista ili foruma i da se specijalizuju za određene teme.
Podrška je takođe jedan od važnih kriterijuma na osnovu koga potencijalni korisnici vrše evaluaciju nekoliko alternativnih opcija prilikom izbora softvera. Dobra podrška je siguran pokazatelj kvaliteta projekta i softvera.
Na slici 3 data je slika ekrana foruma za pomoć korisnicima za projekat Neuroph na *SourceForge*-u.



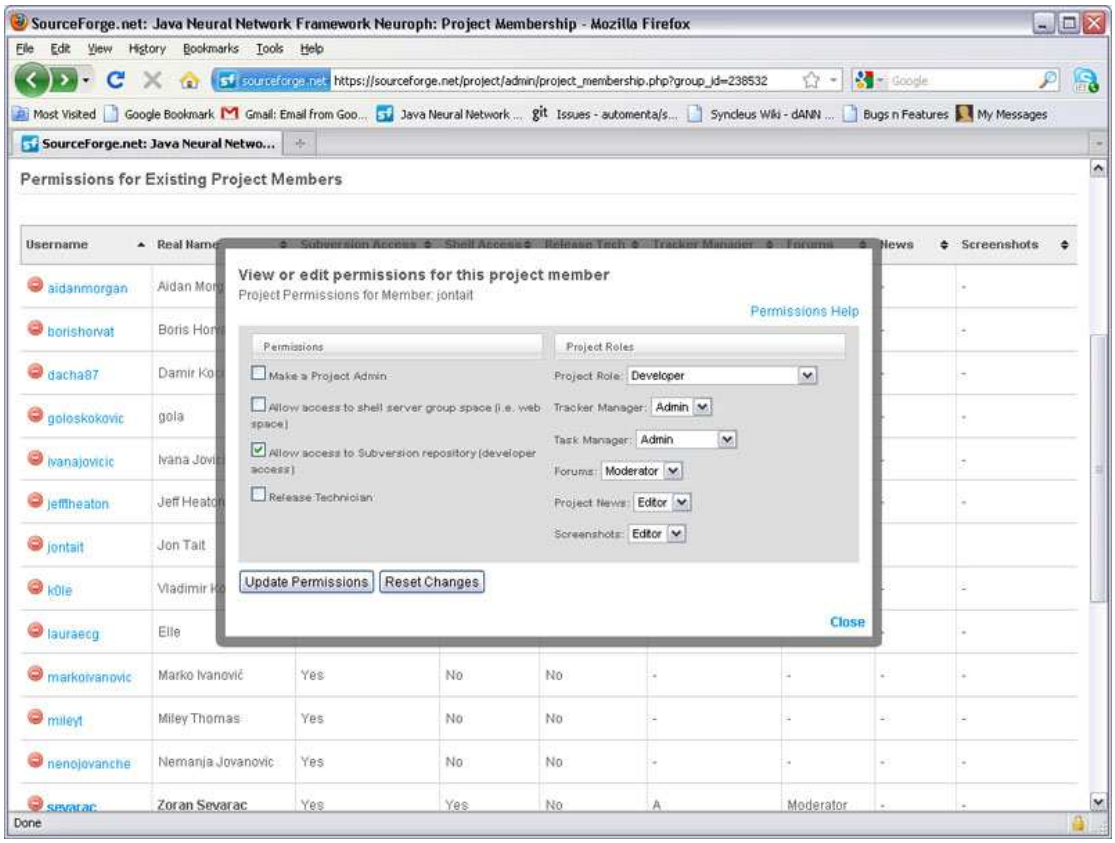
Slika 3. Forum za podršku Neuroph projekta na SourceForge-u

4. Upravljanje članovima razvojnog tima i njihovim dozvolama (*roles, permissions*)

Proces razvoj i održavanja softvera obuhvata razne aktivnosti koje se dodeljuju članovima razvojnog tima. U zavisnosti od zadataka koji im se dodeljuju, izdvajaju se različite uloge članova tima i definišu se različiti nivoi pristupa servisima i materijalima koji se generišu u okviru projekta. Neke od tipičnih predefinisanih uloga su:

- menadžer projekta
- programer
- tester
- kreator programske dokumentacije
- dizajner korisničkog interfejsa
- menadžer za tehničku podršku
- tehničar za podršku
- savetnik/konsultant
- istraživač

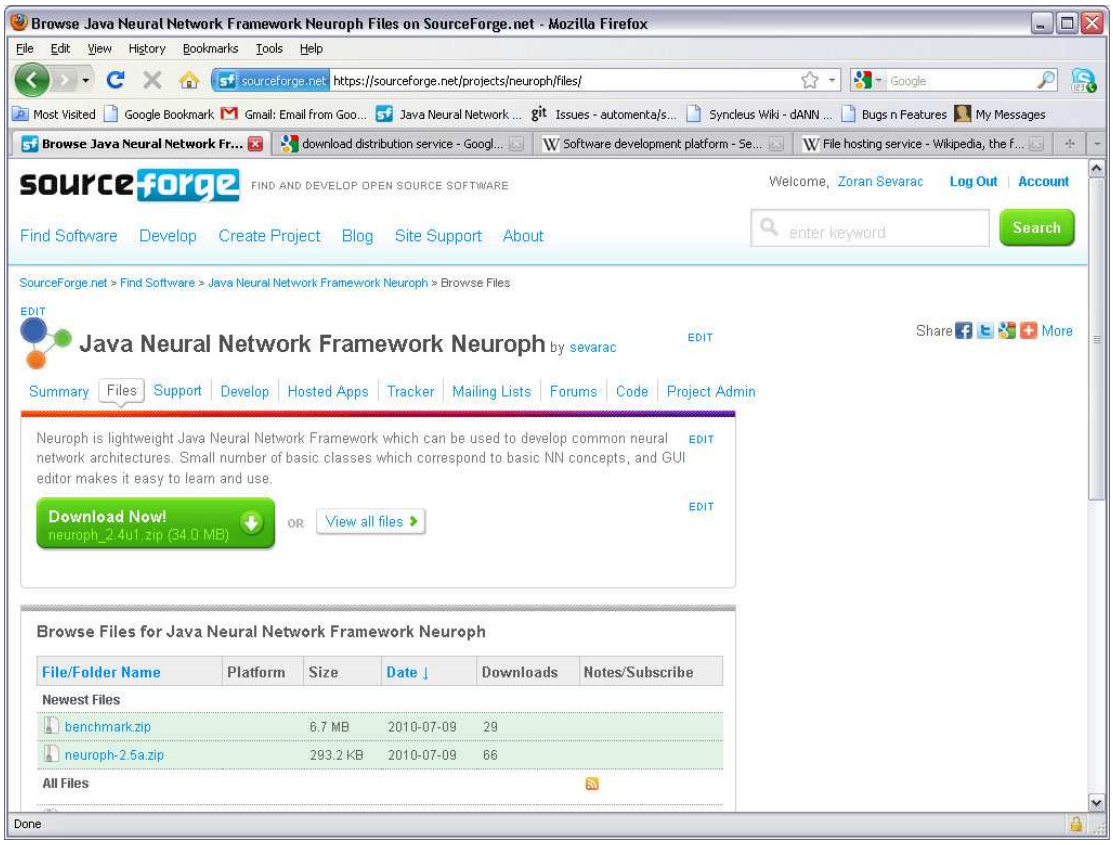
Na slici 4, dat je prikaz ekrana za upravljanje članovima razvojnog tima, njihovim ulogama i dozvolama u okviru Neuroph projekta na SourceForge-u.



Slika 4. Upravljanje članovima razvojnog tima, njihovim ulogama i dozvolama na Source Forge-u

5. Servis za preuzimanje softvera (*download*)

Servis za preuzimanje softvera podrazumeva stranu sa koje je moguće preuzimanje (download) softvera, zajedno sa distribuiranim sistemom tzv. mirror-a preko kojih se vrši preuzimanje. Sistem mirror-a omogućava preraspoređivanje opterećenja servisa za preuzimanje, u zavisnosti od lokacija sa kojih dolaze zahtevi, i na taj način spreči zagušenje servisa usled velikog broja zahteva. U okviru sistema mirror-a isti fajl se nalazi na više lokacija na Internetu, a sistem automatski dodeljuje optimalnu lokaciju sa koje će određeni zahtev za preuzimanje biti uslužen.
Sastavni deo ovog sistema je i servis za objavljivanje softvera pomoću koga se fajllovi postavljaju na servis za preuzimanje



Slika 5. Strana za preuzimanje softvera (download)

6. Web sajt projekta

Web sajt projekta je osnovna lokacija koja korisnicima treba da pruži sve relevantne informacije u vezi sa projektom. Tipičan sajt jednog projekta otvorenog koda treba da sadrži:

- novosti u vezi projekta;
- stranu za preuzimanje;
- sekciju sa dokumentacijom (tehnička dokumentacija i uputstva);

- pregled karakteristika softvera;
- sekciju sa slikama ekrana;
- kratku istoriju projekta i istoriju verzija;
- stranu sa jasno definisanom vizijom i ciljevima projekta;
- plan daljeg razvoja;
- online demonstraciju i primere primene;

7. Statistike

Statistički podaci o projektu omogućavaju brzi uvid u stanje projekta, trenutne aktivnosti i postignute rezultate. Statistike od interesa su:

- posećenost Web sajta projekta;
- broj i dinamika preuzimanja softvera;
- aktivnosti na projektu, broj i dinamika izmena na serveru za kontrolu verzija;
- statistike zahteva za podršku;
- statistike foruma (broj i dinamika poruka).

Sve navedene podatke treba posmatrati i analizirati u različitim vremenskim intervalima, npr. nedelju dana, mesec dana dva meseca, šest meseci i godinu dana, i na osnovu toga se može steći dosta dobra slika o projektu.

2.4. Softverski *framework*

Obzirom da se ovaj rad bavi analizom razvoja posebne vrste softvera, tzv. *softverskih framework-a*, u ovom odeljku detaljnije je objašnjeno šta taj pojam podrazumeva. Uopšteno posmatrano, softverski *framework* sadrži skup softverskih komponenti koje se mogu koristiti iz nekog spoljnog programa [Xiaoping, J., 1999]. Ono što je pri tom značajno istaći, jeste da framework ne obezbeđuje samo neki skup funkcionalnosti koje se koriste iz spoljnog programa (kao softverska biblioteka), već dizajn frameworka određuje arhitekturu programa koji ga koristi, odnosno koji je na njemu zasnovan.
Osnovna svrha softverskih framework-a je da ubrzaju i olakšaju razvoj određene vrste softvera, tako što obezbeđuju standardni osnovni dizajn i konkretna rešenja, za probleme iz određene oblasti (npr. sistema agenata), ili za neke specifične probleme u vezi sa razvojem određene vrste aplikacija (npr. Web aplikacija). Framework obezbeđuje osnovni dizajn tako što sadrži skup povezanih klasa, koje zajedno u celini čine odgovarajuće rešenje za oblast ili problem od interesa za koje je framework namenjen. Pri tom, glavna karakteristika framework-a je da sadrži skup apstrakcija (interfejsa, apstraktnih i osnovnih klasa) i definiše tačke proširenja (*extension points*) koje se koriste za kreiranje proširenja i prilagođavanje framework-a potrebama konkretne primene odnosno aplikacije.
To praktično znači da framework određuje osnovni dizajn sistema, pri čemu se specifični elementi mogu redefinisati ili proširiti od strane korisničkog programa. Framework obezbeđuje osnovne klase i interfejse koje korisnički softver nasleđuje, i implementira odgovarajuće komponente koje se uklapaju u postojeći dizajn framework-a. Uopšteno, može se reći da framework sadrži apstrakcije za višekratnu upotrebu (*reusable abstractions*), sa dobro definisanim programskim interfejson (API).

Glavne karakteristike framework-a su [Riehle, 2000]:

1. **Inverzija kontrole toka izvršavanja** (*inversion of control*). Kod biblioteka i korisničkih aplikacija kontrolu toka izvršavanja definiše pozivaoc funkcije (ili korisnička aplikacija), dok kod framework-a osnovni tok izvršavanja definiše framework. Neki autori smatraju [Riehle, 2000] de je ovo ključna karakteristika frameworka po kojoj se razlikuju od softverskih biblioteka. Kod softverskih biblioteka, korisnički kod poziva biblioteku, dok kod framework-a, framework poziva korisnički kod
2. **Definisano standardno ponašanje** (*default behaviour*) – framework treba da definiše standardno/podrazumevano (*default*) ponašanje u situaciji kada ga korisnik nije definisao. Ovo standardno ponašanje treba da bude nešto korisno što ima smisla u kontekstu određene primene/problema, a ne samo niz praznih operacija.
3. **Proširivost** (extensibility)- framework se može proširiti od strane korisnika redefinisanjem postojećih funkcionalnosti (primenom tehnike nasleđivanja i preklapanja/override metoda) ili dodavanjem specijalizovanih implementacija određenih komponenti frameworka (npr. interfejsa, apstraktnih klasa)

4. **Nepromenljivi delovi framework-a**: osnovna struktura i funkcionalnosti framework-a su nepromenljivi. Korisnici mogu da proširuju framework, ali ne mogu da menjaju njegov kod i osnovnu logiku.

Arhitektura framwork-a

Softverski *framework* sastoj se od nepromenljivih (frozen spots) i promenljivih delova (hot spots, extension points). Nepromenljivi delovi definišu celokupnu arhitekturu sistema, odnosno osnovne komponente sistema i veze između njih. Promenljivi delovi (ili tačke proširenja) framework-a podrazumevaju mesta na kojima programeri koji koriste framework, mogu da dodaju sopstveni programski kod koji obezbeđuje funkcionalnosti potrebne za aplikaciju koja se razvija. Na taj način framework definiše mesta u arhitekturi na kojima je moguće izvršiti prilagođavanje odnosno proširenje. Ova mesta se formalno kreiraju eksplicitnim definisanjem potrebnog interfejsa komponente, ili kreiranjem klase koja faktički definiše interfejs, a čije se metode mogu redefinisati u izvedenim klasama.
Kada se konkretan softverski sistem razvija pomoću framework-a, programeri koriste tačke proširenja u skladu sa zahtevima sistema koji razvijaju. Osnovnu arhitekturu aplikacije i tok izvršavanja definiše framework, i framework poziva/koristi komponente koje programeri kreiraju obično nasleđivanjem apstraktnih klasa i interfejsa iz framework-a.

2.5. Primeri softverskih framework-a iz oblasti inteligentnih sistema

U ovom odeljku dati su primeri nekoliko softverskih *framework-a* otvorenog koda iz raznih oblastii inteligentnih sistema. Pored imena i Internet adrese, za svaki projekat je naveden osnivač i razvojno okruženje, kako bi se napravilo poređenjei analiza u skladu sa elementima koji su navedeni u prethodnim sekcijama.
Softverski *framework* za sistem agenata obezbeđuje osnovu za kreiranje sistema za kolaborativno distribuirano procesiranje. U tabeli 1 navedeno je nekoliko primera. Njihova česta praktična primena je simulacija složenih distribuiranih sistema. Od navedenih *framework-a,* JADE je najopštiji i verovatno najrasprostranjeniji *framework* ovog tipa, i iza njega stoji kompanija Telecom Italia, koja je razvoj ovog *framework-a* započela za sopstvene potrebe (simulaciju telekomunikacionih sistema), a zatim ga objavila pod licencom otvoreog koda. BeeGent sistem, koji je razvila kompanija Toshiba, omogućava 'agentifikaciju' postojećih aplikacija, što znači da se oko postojećih aplikacija kreiraju agenti koji obezbeđuju mehanizme komunikacije i kolaborativni rad. Sistem agenata Couguaar prvobitno je razvijen od strane američke vojske za potrebe simulacije lanaca snabdevanja, a zatim je objavljen pod licencom otvorenog koda.
Sistem agenata Repast je specijalizovan za kreiranje simulacija i modeliranje sistema korišćenjem agenata, i pored *framework-a* obezbeđuje i odgovarajuće alate sa grafičkim interfejsom.
Od navedenih sistema samo Repast je razvijen na SourceForge-u kao projekat otvorenog koda od samoga početka, dok su ostali projekti prvobitno razvijeni od strane kompanija i istraživačke agencije , a zatim postali projekti otvorenog koda jer je postojao interes za dalji razvoj i šire prihvatanje tehnologije na taj način.

Tabela 1. Pregled Java framework-a otvorenog koda za sisteme agenata

Sistemi agenata				
	Naziv *framework-a*	**Osnivač**	**Razvojno okruženje**	**URL**
1	JADE	Telecom Italia	sopstveno okruženje	http://jade.tilab.com
2	BeeGent	Toshiba Corp.	sopstveno okruženje	http://www.toshiba.co.jp/rdc/beegent
3	Couguaar	DARPA	sopstveno okruženje	http://www.cougaar.org
4	Repast	Michael North	Source Forge	http://repast.sourceforge.net

Softverski *framework* za Semantički Web omogućava manipulaciju modelima za reprezentaciju znanja u RDF i OWL jezicima. U tabeli 2 navedeno je nekoliko primera. Najrasprostranjeniji je Jena, koji je kreirala kompanija Hewlett Packard u okviru istraživačkog projekta. Odmah iza njega je Sesame koji je kreirala kompanija Aduna, koja teži da uspostavi vezu između akademskih istraživanja u oblasti semantičkog Web-

a i industrijske primene ove tehnologije. CubicWeb je *framework* koji obezbeđuje gotove komponente za Web aplikacije zasnovanih na logici koja je opisana tehnologijama semantičkog Web-a. JRDF ima za cilj kreiranje standardnog Java programskog interfejsa (API) za tehnologije semantičkog Web-a, po uzoru na već ustaljene Java API kao što su na primer Collections, XML, JDBC itd. Od navedenih primera u ovoj oblasti, dva su razvijena na SourceForge-u, dok su druga dva inicijalno razvijena od strane kompanija.

Tabela 2. Pregled Java framework-a otvorenog koda za semantički Web

Semantčki Web				
	Naziv framework-a	**Osnivač**	**Razvojno okruženje**	**URL**
1	Jena	Hewlett Packard Labs	Source Forge	http://jena.sourceforge.net
2	Sesame	Aduna	sopstveno okruženje	http://www.openrdf.org
3	CubicWeb	Logilab	sopstveno okruženje	http://www.cubicWeb.org
4	JRDF	Andrew Newman	Source Forge	http://jrdf.sourceforge.net

Softverski *framework* za mašinsko učenje obezbeđuje razne tehnike mašinskog učenja koje su na raspolaganju korisničkim programima preko odgovarajućeg programskog interfejsa. U tabeli 3 navedeno je nekoliko primera. Weka je trenutno najveći i najpoznatiji, i nastao je kao akademski projekat otvorenog koda. Apache Mahout je projekat organizacije za razvoj softvera otvorenog koda Apache, kreiran je sa ciljem da obezbedi podršku za mašinsko učenje drugim veoma poznatim projektima organizacije Apache (npr. pretraživač Lucene). Java Machine Learning Library je projekat u razvoju, ali predstavlja skup velikog broja algoritama sa programskim interfejsom u duhu Java jezika i dobrom dokumentacijom. Od ova tri framework-a samo je on razvijen na Source Forge-u.

Tabela 3. Pregled Java framework-a otvorenog koda za mašinsko učenje

Mašinsko učenje				
	Naziv framework-a	**Osnivač**	**Razvojno okruženje**	**URL**
1	Weka	Machine Learning Group at University of Waikato	sopstveno okruženje	http://www.cs.waikato.ac.nz/ml/weka/
2	Apache Mahout	Apache Software Foundation	Apache	http://mahout.apache.org
3	Java Machine Learning Library	Thomas Abeel	Source Forge	http://java-ml.sourceforge.net

Softverski *framework* za genetske algoritme omogućava primenu tehnika za pretraživanje prostora stanja i optimizaciju zasnovanih na principima evolutivnih algoritama. U tabeli 4 navedeno je nekoliko primera. JGAP je veoma poznat i često korišćen, ima jasan i fleksibilan dizajn. Programski kod i dokumentacija su veoma kvalitetni, što je jedan od razloga za uspeh.
JAGA je nastao kao akademski projekat sa ciljem da kreira istraživačku alatku u ovoj oblasti. Framework ima solidnu strukturu i obezbeđuje osnovne vrste algoritama iz oblasti genetskih algoritama, uz dobru dokumentaciju.
Jenetics ja mali *framework*, koji je razvijen sa ciljem da jasno izdvoji osnovne koncepte u genetskim algoritmima. Iako nije ostvario značajan razvoj i primenu, ovaj *framework* je odličan primer dobro koncipiranog i implementiranog framework-a. Sva tri framework-a razvijena su na Source Forge-u.

Tabela 4. Pregled Java framework-a otvorenog koda za genetske algoritme

Genetski algoritmi				
	Naziv framework-a	**Osnivač**	**Razvojno okruženje**	**URL**
1	JGAP	Neil Rotstan	Source Forge	http://jgap.sourceforge.net
2	JAGA	Greg Paperin, University College London	Source Forge	http://www.jaga.org
3	Jenetics	Franz Wilhelmstötter	Source Forge	http://jenetics.sourceforge.net

Softverski *framework* za neuronske mreže omogućava kreiranje raznih vrsta neuronskih mreža i njihovu primenu. U tabeli 5 navedeni su najznačajniji projekti otvorenog koda iz ove oblasti. JOONE je prvi veliki framework iz ove oblasti, i na neki način je kreirao osnovu i postavio standarde za one koji su razvijeni kasnije. I pored velikog broja problema, imao je veliki broj primena i funkcionalnosti.
Encog je *framework* visokih performansi sa izuzetnom podrškom za paralelno procesiranje. Nastao je kao prateći softver za popularnu knjigu o programiranju neuronskih mreža u Java-i , i praktično se razvio iz JOONE-a. Neuroph je nastao sa ciljem da pruži jednostavan programski interfejs (API) za neuronske mreže u duhu programskog jezika Java, i da pojednostavi upotrebu neuronskih mreža.
Nastao je kao akademski projekat u okviru Laboratorije za veštačku inteligenciju na Fakultetu organizacionih nauka. Neuroph i Joone su razvijeni na Source Forge-u, dok je Encog razvijen na Google Code-u.

Tabela 5. Pregled Java framework-a otvorenog koda za neuronske mreže

Neuronske mreže				
	Naziv framework-a	**Osnivač**	**Razvojno okruženje**	**URL**
1	JOONE	Paolo Marrone	Source Forge	http://sourceforge.net/projects/joone
2	Encog	Jeff Heaton	Google Code	http://www.heatonresearch.com/encog
3	Neuroph	Zoran Sevarac, GOAI Group, FON	Source Forge	http://neuroph.sourceforge.net

U daljem tekstu dat je detaljniji prikaz navedenih softverskih framework-a iz oblasti neuronskih mreža. Detaljno su analizirani razni aspekti sva tri framework-a, sagledane su njihove glavne karakteristike i životni ciklus.

2.5.1. JOONE

2.5.1.1. O projektu

JOONE (Java Object Oriented Neural Engine) je Java softverska framework otvorenog koda za razvoj neuronskih mreža. Razvoj je počeo 2001. godine i trajao sve do 2006. kada je objavljena poslednja verzija. JOONE je objavljen pod *Lesser GNU Public License (LGPL)* licencom za softver otvorenog koda.
Predstavlja prvi široko prihvaćenu Java framework ove vrste, i pored samih komponenti za kreiranje neuronskih mreža, obezbeđuje editor za neuronske mreže sa grafičkim korisničkim interfejsom i veliki broj dodatnih komponenti koje plakšavaju primenu neuronskih mreža.
Osnovna ideja koja stoji iza ovog projekta je razvoj distribuiranog okruženja za trening neuronskih mreža koje bi omogućilo simultani distribuirani trening više neuronskih mreža sa različitim parametrima, i definisanje mehanizama za izbor najboljih rešenja. Time bi se značajno unapredilo i olakšalo rešavanje problema pomoću neuronskih mreža Iako ova ideja nije realizovana do kraja, JOONE je imao veliki broj korisnika, aktivnu zajednicu koja je radila na njegovom razvoju i na neki način je kreirao osnovu za dalji razvoj ove oblasti.
Zamišljen je kao framework koji može da podrži razne vrste neuronskih mreža, ali u praksi je najčešće korišćen za realizaciju tzv. višeslojnih perceptrona.
Glavni problemi u razvoju ovog framework-a su bili: veliki broj internih *bug*-ova, pomalo haotična arhitektura u celini koja je posledica rasta framework-a, neintuitivan i komplikovan javni programski interfejs (API), slaba dokumentacija za pojedine delove.

2.5.1.2. Osnovne karakteristike

JOONE framework poseduje veliki broj funkcionalnosti za razvoj i primenu neuronskih mreža. U sekciji uporedni pregled dat je detaljan pregled glavnih funkcionalnosti
Osnovni skup klasa framework-a za neuronske mreže prati kvalitetno napravljena aplikacija sa vizuelnim editorom neuronskih mreža, koja olakšava razvoj i korišćenje neuronskih mreža. Ove dve stvari su implementirane potpuno odvojeno - JOONE framework predstavlja jezgro odnosno *engine* koji sadrži osnovne komponente i okruženje za izvršavanje, dok aplikacija obezbeđuje grafički interfejs za manipulaciju i rad sa neuronskim mrežama. Zajedno čine kompletno razvojno okruženje za neuronske mreže. Osnovne karakteristike JOONE-a su:

- kompletna biblioteka Java klasa za kreiranje i primenu neuronskih mreža;
- aplikacija sa grafičkim interfejsom za manipulaciju neuronskim mrežama;
- veliki broj dodatnih ulazno/izlaznih adaptera za rad sa podacima u različitim formatima (tekst, slike, baze podataka, excel);
- podrška za *BeanShell* skript jezik koji omogučava pisanje skriptova kojima je upravlja Java objektima;
- arhitektura i domenski model zasnovani na slojevima neurona (*Layers*), sinapsama/vezama između neurona (Synapses), i matričnim operacijama
- veliki broj klasa, neintuitivan i 'iscepkan' konceptualni model težak za praćenje.

2.5.2. Encog

2.5.2.1. O projektu

Encog je softverski framework koji obezbeđuje podršku za razne tehnologije veštačke inteligencije, a posebno je poznat u oblasti neuronskih mreža. Pored neuronskih mreža Encog ima podršku za agente, genetske algoritme i *data mining* algoritme. Razvoj je u nekom obliku počeo 2005. godine kroz primere u okviru knjige Introduction to neural newtrorks with Java. Primeri u navedenoj knjizi bili su u velikoj meri bazirani na JOONE framework-u, međutim zbog pomenutih problema sa tim framework-om, autor knjige je počeo razvoj sopstvenog framework-a i prva verzija objavljena je 2008 godine. Poslednja verzija 2.5 objavljena je u oktobru 2010. godine pod *Apache 2* licencom za softver otvorenog koda. Framework ima aktivan razvoj, i veliku zajednicu korisnika i programera koji doprinose razvoju na razne načine.
Kao i JOONE, Encog takođe obezbeđuje Java framework i posebnu aplikaciju sa grafičkim interfejsom koja služi kao editor za razvoj i trening neuronskih mreža. Pored Java verzije, potpuno isti framework objavljen je i za .NET platformu u programskom jeziku C# .
Arhitektura je u osnovi donekle slicna JOONE-u, ali ipak ima originalna rešenja i bolje je struktuirana. I pored objektnog modela u osnovi, implementacija je u velikoj meri zasnovana na nizovima zbog performansi. Spoljni programski interfejs je dobar ali u izvesnoj meri opterećuje korisnika sa detaljima interne implementacije. Glavni kvalitet Encog-a su visoke performanse i izuzetno dobra podrška za paralelno procesiranje na

procesorima sa više jezgara i podrškom za korišćenje grafičkih procesora (GPU) prilikom izračunavanja.
Na neki način, Encog predstavlja sledeći korak u razvoju framework-a otvorenog koda za neuronske mreže posle JOONE-a.

2.5.2.2. Osnovne karakteristike

Encog obezbeđuje kompletnu softversku podršku za razvoj i primenu neuronskih mreža pomoću Java i C# biblioteka, i aplikacije za kreiranje i trening neuronskih mreža sa grafičkim interfejsom. Kao i kod JOONE-a, osnova framework-a i aplikacija su dve odvojene komponente, a objektni model je sličan JOONE-u obzirom da je u početnim fazama razvoja na njemu bio zasnovan. Glavne karakteristike *Encog-a* su:

- Java i C# biblioteke klasa za razvoj i primenu neuronskih mreža;
- aplikacija sa grafičkim interfejsom za manipulaciju neuronskim mrežama;
- rad sa raznim ulaznim formatima podataka (slike, baze podataka, tekst, xml);
- napredne varijacije algoritama za učenje;
- podrška za razne vrste neuronskih mreža;
- korišćenje genetskih algoritama u učenju neuronskih mreža;
- pre-procesiranje podataka;
- podrška za skript jezik *Encog script*;
- podrška za procesiranje sa više procesora/procesorskih jezgara i GPU;
- zahvaljujući podrški za paralelno procesiranje ima veoma visoke performanse;
- osnovne konceptualni model čini skup komponenti *Layer, Synapse, Logic, Training;*
- implementacija koja je zasnovana na operacijama sa nizovima teška je za praćenje logike i pored relativno jasnog objektog modela.

2.5.3. Neuroph

2.5.3.1. O projektu

Neuroph je softverski framework otvorenog koda koji obezbeđuje biblioteku Java klasa i specijalizovano razvojno okruženje za neuronske mreže. Projekat je započeo razvoj kao studentski projekat iz predmeta Inteligentni sistemi na Fakultetu organizacionih nauka 2004. godine, a prva verzija je objavljena 2008. godine na *SourceForge-u*. Šest meseci nakon javnog objavljivanja, zahvaljujući podrški za prepoznavanje slika pomoću neuronskih mreža, Neuroph započinje veoma dinamičan razvoj i zbog svojih karakteristika i jednostavnosti korišćenja stiče veliku popularnost. Od 2010. godine predstavlja sam vrh u svojoj oblasti i uspostavlja saradnju sa drugim poznatim projektima otvorenog koda, NetBeans i Encog. U saradnji sa NetBeans projektom razvijeno je integrisano razvojno okruženje za neuronske mreže, a u saradnji sa Encog projektom obezbeđena je integracija sa Encog framework-om.
Neuroph framework ima podršku za najčešće korišćene vrste neuronskih mreža i algoritama za učenje, vrlo je fleksibilan i jednostavan za proširenje. Upravo ta fleksibilnost i jednostavnost korišćenja su glavne prednosti ovog frameworka. Projekat

ima aktivan razvoj, dobru podršku korisnicima putem foruma, detaljnu programsku dokumentaciju i uputstva za primenu.
Integrisano razvojno okruženje (IDE) za neuronske mreže zasnovano na NetBeans Platformi donelo je značajan napredak, jer obezbeđuje korisnički interfejs i funkcionalnosti koje se sreću kod softvera profesionalnog nivoa kvaliteta, a sva rešenja su dostupna pod licencom otvorenog koda. Integracija sa Encog frameworkom obezbedila je podršku za korišćenje tehnologija za paralelno procesiranje i efekat sinergije za oba projekta: jednostavnost i fleksibilnost Neuroph-a i visoke performanse Encog-a. Poslednja verzija Neuroph-a 2.5 u saradnji sa timom Encog-a portovana je na Dot Net platformu u jeziku C#.

2.5.3.2. Osnovne karakteristike

Kao i prethodno opisani projekti, Neuroph takođe obezbeđuje biblioteku Java klasa, i aplikaciju sa grafičkim interfejsom za kreiranje neuronskih mreža. Glavna karakteristika Neuroph framework-a po kojoj se izdvaja od ostalih je vrlo intuitivan i jednostavan programski interfejs (API), i jasna logika zahvaljujući domenski orijentisanom dizajnu.
Aplikacija za razvoj neuronskih mreža zasnovana na NetBeans Platformi pruža visoko profesionalni korisnički interfejs, napredne funkcionalnosti i integraciju sa Java razvojnim okruženjem u okviru iste aplikacije. Sve to ga čini jedinstveni alatom ove vrste, jer se u okviru iste aplikacije može razviti neuronska mreža, istestirati i ugraditi u korisnički program. Neuroph podržava osnovne vrste neuronskih mreža i algoritama za učenje, a pored toga ima i specijalizovane alate za primenu neuronskih mreža u određenim oblastima. U osnovnim crtama glavne karakteristike Neuroph-a su:

- Java i C# biblioteke klasa za neuronske mreže;
- integrisano razvojno okruženje za neuronske mreže i Java-u zasnovano na NetBeans Platformi;
- podrška za prepoznavanje slika, štampanih slova i rukom pisanih slova;
- podrška za predviđanje na berzi;
- podrška za kreiranje raznih vrsta neuronskih mreža i algoritama za učenje;
- podrška za rad sa podacima u tekstualnom formatu i slikama;
- veoma jasan i fleksibilan objektni model jednostavan za korišćenje i nadogradnju;
- veoma jednostavan i jasan javni programski interfejs (public API);
- kvalitetna programska dokumentacija (javadoc) i samodokumentujući kod;
- bogat objektni model ima niže performanse u odnosu na matrično orijentisane operacije i optimizovani dizajn primenjen u Encog-u;
- podrška za integraciju sa Encog framework-om i korišćenje komponenti za paralelno procesiranje visokoih performansi.

3. PLANIRANJE RAZVOJA SOFTVERA

U ovom poglavlju opisano je planiranje razvoja dodatnih funkcionalnosti za Neuroph framework. Prvo je izvšena analiza trenutnog stanja, u kojoj su u najbitnijim elementima sagledane karakteristike tri aktuelna framework-a za razvoj neuronskih mreža. Zatim su definisani cilj i strategija razvoja, na osnovu kojih su iz uporednog pregleda funkcionalnosti (datih u prilogu 1) izabrane funkcionalnosti koje će se razvijati. Na kraju je dat pregled planiranih funkcionalnosti za razvoj sa kratkim opisom i napomenama u vezi njihovog značaja.

3.1. Analiza trenutnog stanja

U prethodnom poglavlju dat je kratak pregled tri najznačajnija Java framework-a otvorenog koda iz oblasti neuronskih mreža. U prilogu 1 dat je detaljan uporedni pregled karakteristika, a u prilogu 2 primeri korišćenja ovih framework-a u Java kodu. Kako bi se sagledalo postojeće stanje i širi kontekst uporednog pregleda, dat je kratak zaključak za svaki od obrađenih framework-a.

JOONE – Prvi značajni Java framework otvorenog koda u oblasti neuronskih mreža. Iako podržava mali broj vrsta neuronskih mreža i algoritama za učenje, ima veliki broj pomoćnih funkcionalnosti i alata. U tom pogledu definisao je standard šta sve jedan framework za neuronske mreže treba da sadrži. Ima komplikovanu arhitekturu, veliki broj bug-ova, slabu dokumentaciju i više se ne razvija. Ipak ima veliki značaj jer je kao prvi široko prihvaćeni framework, postavio osnovu za dalji razvoj u ovoj oblasti.
Komplikovan dizajn, je sasvim sigurno jedan od razloga za veliki broj bug-ova i teško održavanje, što je sve zajedno dovelo do gašenja projekta. U jednom trenutku je postalo jasno da je sa postojećom arhitekturom teško izaći na kraj sa svim bug-ovima i tada je jedino rešenje bilo napraviti značajne promene u arhitekturi softvera i praktično pisati sve ispočetka. Međutim to je veliki posao, a u zajednici nije bilo interesovanja za tako nešto.
Još jedan problem u vezi ovog projekta je što pokretač projekta nije aktivno vodio projekat u određenom smeru, već su članovi zajednice dograđivali framework prema svojim potrebama bez koordinacije. To je na kraju rezultiralo haotičnim dizajnom i bug-ovima.
Imajući sve ovo u vidu, može se reći da je JOONE framework dobar kao neka vrsta specifikacije koje sve funkcionalnosti framework ove vrste treba da sadrži, i dobar primer koje greške treba izbeći.

Encog - Ovaj framework je praktično direktni naslednik JOONE-a. Ima podršku za veliki broj vrsta neuronskih mreža i algoritama za učenje, i veliki broj pomoćnih komponenti, koje su praktično reimplementirane funkcionalnosti JOONE-a.
Veoma jaka strana Encog-a je što ima podršku za paralelno procesiranje na procesorima sa više jezgra i grafičkim procesorima. Ima aktivan razvoj, kvalitetnu dokumentaciju i dobru podršku. U pogledu broja funkcionalnosti i performansi predstavlja pojedinačno najrazvijeniji framework u ovom trenutku.

U pogledu arhitekture, iako je na konceptualnom nivou vrlo slična JOONE-u, na implementacionom nivou je znatno bolje rešena. Kod je dosta čitljiviji, ima manji broj klasa i logika nije toliko iscepkana kao kod JOONE-a. Međutim i pored toga celokupna logika je pomalo nejasna, jer je skrivena iza velikog broj matričnih operacija sa nizovima. To je cena visokih performansi i podrške za paralelno procesiranje.

Neuroph – Iako je najmlađi framework u ovoj oblasti (u odnosu na prethodne dva), veoma brzo je steko veliku popularnost i primenjen je u raznim oblastima. To je pre svega zahvaljujući jednostavnom korišćenjenju, razumljivom dizajnu i veoma dobroj dokumentaciji. Još jedan razlog za široku rasprostranjenost i brzo prihvatanje je što Neuroph obezbeđuje gotove alate/komponente za tipične zadatke za koje se koriste neuronske mreže kao što su prepoznavanje slika, prepoznavanje slova i predviđanje na berzi.
Poseban značaj ima i integrisano razvojno okruženje za neuronske mreže i Javu (IDE) koje obezbeđuje profesionaln i grafički interfejs za rad sa neuronskim mrežama.
U pogledu podržanih funkcionanosti u poređenju sa druga dva framework-a, Neuroph-u nedostaje podrška za nekoliko vrsta neuronskih mreža i algoritama za učenje (u odnosu na Encog), kao i razni dodatni ulazno/izlazni adapteri koje olakšavaju primenu (u odnosu na JOONE).
Bogat objektno orijentisani dizajn pored prednosti u pogledu čistog dizajna i jasne logike, ima značajne nedostatke u pogledu performansi. Pošto neuronske mreže imaju veliki broj izračunavanja, brze operacije sa nizovima koje koristi Encog su u tom slučaju značajno brže, jer se izbegava veliki broj poziva funkcija. Taj problem koji praktično nije bilo moguće rešiti sa postojećom arhitekturom Neuroph-a, rešen je u saradnji sa Encog projektom tako što je napravljena integracija koja omogućava da Neuroph koristi brzi Encog *engine* za izračunavanja. Na taj način krajnji korisnik tokom razvoja može da koristi jasan i jednostavan programski interfejs (API) Neuroph-a, a u produkciji da se sve izvršava na brzom Encog engine-u.

3.2. Definisanje cilja i strategije razvoja

Cilj razvoja je kreiranje dodatnih funkcionalnosti za Neuroph framework kako bi imao sve mogućnosti koje imaju i druga dva značajna framework-a iz ove oblasti, JOONE i Encog. Na taj način Neuroph održava konkurentnost, i otvara nove mogućnosti za dalji razvoj i primenu.
Osnovno pitanje koje se postavlja, jeste da li ima smisla praviti nešto što već postoji, i da li je moguće iskoristiti postojeće komponente?
Programski kod postojećih rešenja iz Encog-a i JOONE-a ne može direktno iskoristiti, ali se mogu se sagledati postojeća rešenja i na neki način se unaprediti u kontekstu Neuroph framework-a.
U konkretnom slučaju ima smisla praviti dodatne funkcionalnosti za Neuroph, jer će one biti implementirane u skladu sa glavnim principima Neuroph framework-a: jednostavan logičan dizajn koji prati domenski model, samodokumentujući kod, i jednostavan javni programski interfejs API. Ovo je značajno unapređenje u odnosu na postojeća rešenja i zasniva se na uspešnim principima potvrđenim u praksi. Takođe,

treba istaći da je značajan deo Encog bi framework-a koji obezbeđuje podršku za paralelno procesiranje već integrisan sa Neuroph-om.
Strategija razvoja treba da obezbedi uspešnu realizaciju ciljeva razvoja, i definiše principe na osnovu kojih se upravlja razvojem. Obzirom na postavljeni cilj razvoja, strategija razvoja treba da definiše principe na osnovu kojih će se izabrati funkcionalnosti koje će se razvijati, i kojim redosledom.
Strategija razvoja se sastoji iz sledećih principa:

1. Za što kraće vreme napraviti dodatne funkcionalnosti koji će Neuroph učiniti konkurentnim u odnosu na druge framework-e;
2. Nove funkcionalosti izabrati tako da se što lakše mogu nadograditi na postojeće;
3. Prilikom dodavanja novih funkcionalnosti, prioritet imaju one za kojima se pokazala stvarna potreba u praksi;
4. Povećati mogućnosti za praktičnu primenu;
5. Pojednostaviti korišćenje.

Strategija razvoja je definisana tako da obezbedi što bolje efekte u plasmanu gotovog sofverskog proizvoda, sa najmanjim utroškom razvojnih resursa. U suštini se zasniva na ideji da treba maksimalno povećati mogućnosti praktične primene postojećih funkcionalnosti, a onda dodavati nove. Pri tom nove funkcionalosti izabrati tako da se što lakše mogu nadograditi na postojeće. Razvoj novih funkcionalnosti takođe treba da bude stavljen u funkciju pojednostvljivanja korišćenja softvera, što sasvim sigurno uvek ima pozitivne efekte kod korisnika.

3.3. Identifikovanje funkcionalnosti - zahteva

Iz tabela uporednog prikaza funkcionalnosti datih u prilogu 1, mogu se identifikovati funkcionalnosti koje Neuroph framework nema, a JOONE i Encog imaju. Sve ove funkcionalnosti su potencijalni kandidati za razvoj kojim se unapređuje Neuroph framework. Funkcionalnosti planirane za razvoj su izabrane u skladu sa principima definisanim u strategiji razvoja.

Tabela 6. Dodatne funkcionalnosti za Neuroph - vrste neuronskih mreža

Vrsta neuronskih mreža	**U planu za razvoj**
Boltzmann machine	Da
Counterproragation neural network (CPN)	Da
Recurrent-Elman	Ne
Recurrent-Jordan	Ne
Recurrent-SOM	Ne
Adaptive resonance theory (ART1)	Ne
Time Delay neural network	Ne
Support Vector Machine (SVM)	Ne
PCA neural network	Da

Tabela 7. Dodatne funkcionalnosti za Neuroph - algoritmi za učenje

Algoritmi za učenje	**U planu za razvoj**
Simulated Annealing	Ne
Resilient backpropagation	Da
Manhattan Update Rule Propagation	Ne
Levenberg Marquardt (LMA)	Ne
Scaled Conjugate Gradient	Ne
Instar/Outstar	Da
Genetic algorithm training	Ne
K Means	Ne

Tabela 8. Dodatne funkcionalnosti za Neuroph - funkcije transfera

Funkcije transfera	**U planu za razvoj**
SoftMax	Ne
Sin wave	Da
Logarithmic	Da
Delay	Ne
Context function	Ne

Tabela 9. Dodatne funkcionalnosti za Neuroph - generisanje slučajnih brojeva

Tehnike za generisanje slučajnih brojeva	**U planu za razvoj**
Gaussian	Da
Fan-In	Ne
Nguyen-Widrow	Da
Distort	Da
Consistent	Da

Tabela 10. Dodatne funkcionalnosti za Neuroph - adapteri za ulazne podatke

Adapteri za ulazne podatke	**U planu za razvoj**
JDBC	Da
XML	Ne
URL	Da
Yahoo finance input	Ne
Stream	Da

Tabela 11. Dodatne funkcionalnosti za Neuroph - adapteri za izlazne podatke

Adapteri za izlazne podatke	**U planu za razvoj**
Fajl	Da
Slika	Ne
Stream	Da
JDBC	Da

Tabela 12. Dodatne funkcionalnosti za Neuroph - razne napredne funkcionalnosti

Razne napredne funkcionalnosti	**U planu za razvoj**
Normalizacija podataka	Da
Benchmark	Da

3.4. Pregled zahteva

U ovom odeljku dat je kratak opis funkcionalnosti koje su izabrane za razvoj uz kratak opis, kako bi se sagledao njihov značaj u kontekstu strategije razvoja.

Vrste neuronskih mreža

Boltzmann machine [Fahlman, et. al., 1983] – vrlo slična već postojećim Hopfield [Hopfield, 1982] i BAM [Kosko, 1988] mrežama, s tim što uvodi još jedan sloj neurona što joj omogućava rešavanje složenijih problema i bolje rezultate. Za učenje se može koristiti tzv. Hebovo [Hebb, 2002] učenje koje takođe već postoji u Neuroph-u.

Counterproragation neural network (CPN) [Hecht-Nielsen, 1987] – takođe se može napraviti od postojećih komponenti, predstavlja troslojnu mreža zasnovanu na principima kompetitivnog učenja. Značajna je zato što garantuje uspešnost učenja za razliku od nekih drugih algoritama kao npr. Backpropagation [Rumelhart, 1986] koji se mogu zaglaviti u lokalnom minimumu.

PCA neural network [Oja, 1989] – implementira PCA (*Principal Component Analysis*) algoritam koji služi za redukciju dimenzija ulaznih podataka. Može se koristiti kao ulazni filter za preproceiranje u kombinciji sa drugim neuronskim mrežama

Algoritmi za učenje

Resilient Backpropagation [Riedmiller, 1994] – veoma napredna varijacija Backpropagation algoritma, omogućava učenje u manjem broju iteracija i bolju stabilnost algoritma. Može se izvesti iz postojeće implementacije Backpropagation algoritma u Neuroph-u.

Instar/Outstar [Carpenter, 2003] – algoritam za učenje koji predstavlja kombinaciju dva već postojeća algoritma u Neuroph-u, Instar i Outstar. Koristi se za trening CPN mreže.

Funkcije transfera za neurone

Sin – sinusna funkcija
Logarithmic – logaritanska funkcija

Tehnike za generisanje slučajnih brojeva

Tehnike za generisanje slučajnih brojeva koriste se prilikom kreiranja neuronskih mreža za početnu inicijalizaciju težinskih koeficijenata. Incijalizacija težinskih koeficijenata ima uticaj na brzinu i uspešnost učenja mreže. Za razvoj su izabrane dve tehnike:

- Gaussian
- Nguyen-Widrow [Nguyen, et. al., 1990]

Ulazni adapteri

Ulazni adapteri omogućavaju čitanje podataka iz raznih izvora:

Stream – ulaz za neuronske mreže se uzima iz ulaznog toka (*InputStream*);
Fajl – ulaz za neuronske mreže se uzima iz zadatog fajla;
URL - ulaz za neuronske mreže se uzima iz zadatog URL-a;
JDBC – ulaz za neuronske mreže se uzima pomoću SQL upita iz baze.

Izlazni adapteri

Izlazni adapteri omogućavaju skladištenje vrednosti sa izlaza neuronske mreže

Stream - skladištenje izlaza neuronske mreže preko izlaznog toka (*OutputStream*)
Fajl – skladištenje izlaza neuronske mreže u fajl;
URL – slanje izlaza neuronske mreže na zadati URL;
JDBC – skladištenje izlaza neuronske mreže pomoću SQL upita u bazu podataka;

Razne napredne funkcionalnosti

Normalizacija podataka – skup tehnika za pripremu podataka za obradu pomoću neuronskih mreža koja podrazumeva svođenje vrednosti na interval [0, 1] ili [-1, 1]. Ovo je funkcionalnost od suštinske važnsti, jer će obezbediti integrisano rešenje za pripremu podataka i trening u okviru Neuroph-a.

Benchmark – sistem za testiranje performansi neuronskih mreža. Treba da omogući testiranje brzine učenja neuronskih mreža implementiranih pomoću Neuroph-a ali i drugih framework-a radi poređenja. Ova funkcionalnost će omogućiti testiranje i pronalaženje najboljih implementacija i tehničkih rešenja, i poređenje performansi sa drugim rešenjima.

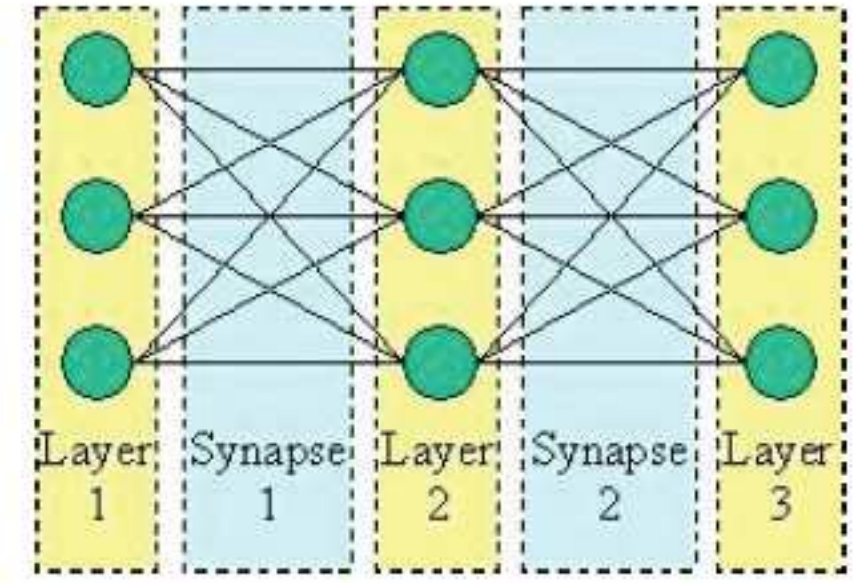
4. ANALIZA I PROJEKTOVANJE

U ovom poglavlju izvršena je analiza arhitekture tri izabrana softverska framework-a iz oblasti neuronskih mreža, i detaljnije su analizirani struktura i načini proširenja Neuroph framework-a, koji je izabran kao osnova na kojoj su demonstrirani principi softverskog inženjerstva prilikom razvoja softverskog framework-a iz oblasti inteligentnih sistema. Na osnovu analize Neuroph framework-a, dat je opšti model framework-a i definisani su principi projektovanja framework-a. U skladu sa definisanim modelom i principima projektovanja, izvršeni su analiza zahteva i projektovanje novih funkcionalnosti za Neuroph framework.

4.1. Analiza arhitektura izabranih softverskih framework-a iz oblasti neuronskih mreža

4.1.1. Arhitektura JOONE framework-a

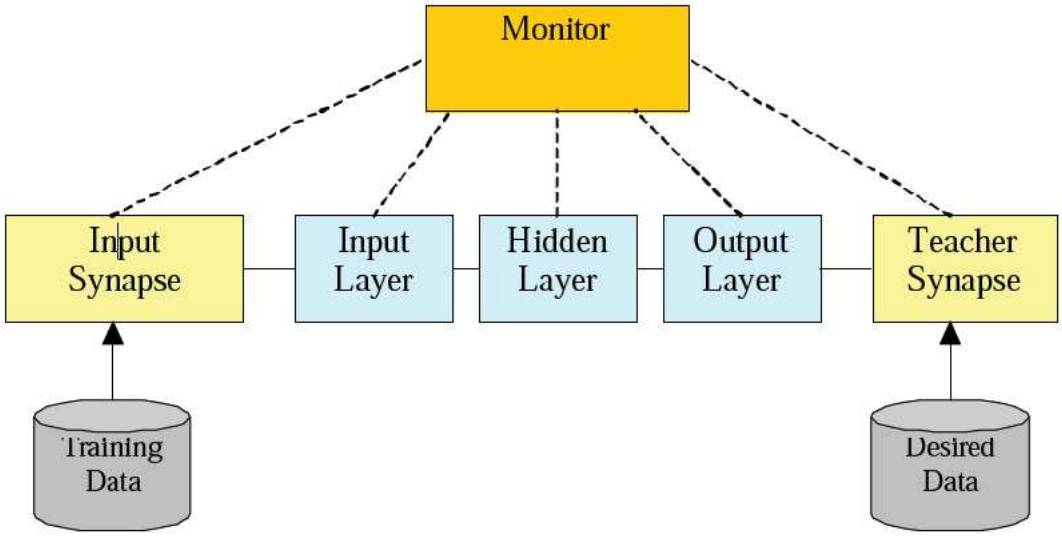
Osnovne komponente JOONE framework-a su *Layer* (sloj neurona) i *Synapse* (veze između neurona iz dva sloja). Na slici 6 prikazano je kako se neuronska mreža gradi pomoću ove dve komponente.



Slika 6. JOONE neuronska mreža napravljena pomoću *Layer* i *Synapse* komponenti

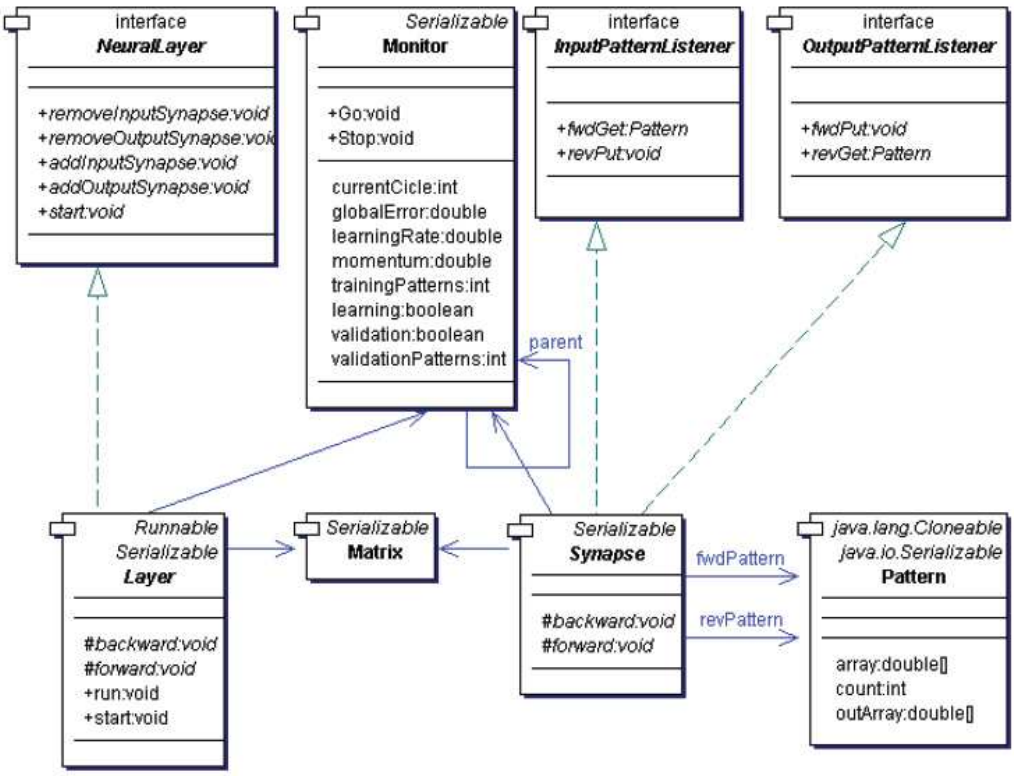
Layer objekat je osnovni element neuronske mreže koji se sastoji od skupa neurona istih karakteristika. Ova komponenta pomoću svoje funkcije prenosa (*transfer function*), na osnovu niza ulaza (ulaznog vektora) izračunava niz izlaza (izlazni vektor), i preko sinapsi prenosi te vrednosti na sledeći sloj neurona – takođe *Layer objekat*. *Layer* je aktivni element neuronske mreže u JOONE, i svaki *Layer* se izvršava u posebnoj programskoj niti (*thread*).

Synapse objekat predstavlja veze između dva sloja neurona (*Layer*) preko kojih se vrši prenos signala sa jednog sloja na drugi. Parametri sinapsi, tzv. težinski koeficijenti veza između neurona, se podešavaju tokom učenja u skladu sa primenjenim algoritmom za učenje neuronske mreže. Obe JOONE komponente (i Layer I Synapse) imaju ugrađeni mehanizam za podešavanje težinskih koeficijenata veza u skladu sa algoritmom za učenje.
Monitor objekat ima dve osnovne uloge: sadrži podešavanja (*learning rate, mometum*) i parametre (broj iteracija učenja, broj podataka za trening) algoritama za učenje i služi kao kontroler, odnosno upravlja radom neuronske mreže (start/stop). Pored toga Monitor upravlja događajima (*Events*) tokom rada neuronske mreže, koji se mogu koristiti i za asinhronu komunikaciju sa nekom spoljnom aplikacijom. Na slici 7. prikazan je kompletan sistem JOONE neuronske mreže koji se sastoji iz nekoliko slojeva neurona, montitora, učitelja i podataka za trening.



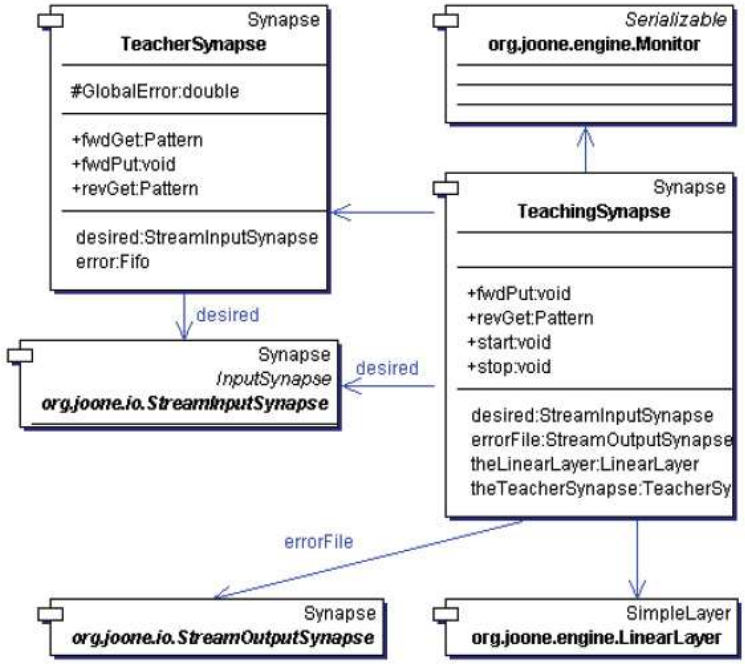
Slika 7. Kompletan JOONE sistem za učenje neuronske mreže

Na slici 8. dat je dijagram osnovnih klasa JOONE framework-a koje se koriste za kreiranje neuronskih mreža.



Slika 8. Dijagram osnovnih klasa JOONE framework-a

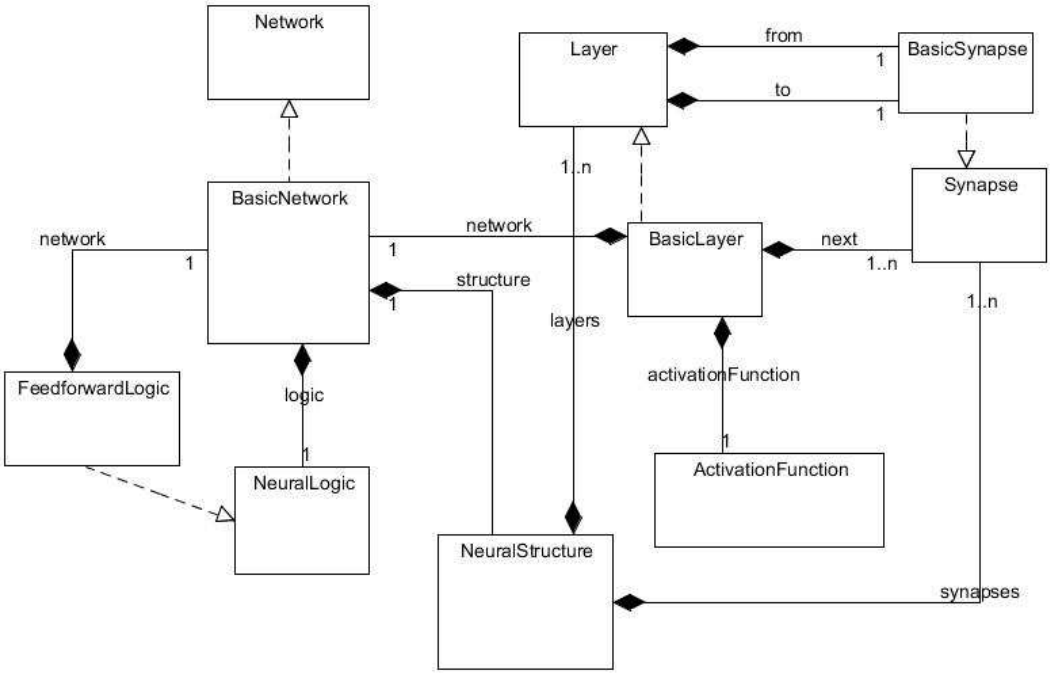
Na slici 9. dat je dijagram klasa pomoću kojih se realizuju algoritmi za učenje neuronskih mreža u JOONE framework-u.



Slika 9. Dijagram klasa koje realizuju učenje

4.1.2. Arhitektura Encog framework-a

Dizajn Encog-a se zasniva na korišćenju nekoliko interfejsa i odgovarajućih osnovnih klasa koje implementiraju ove intefejse. Ovo je vrlo dobar patern koji s jedne strane omogućava programiranje prema interfejsu i daje veliku fleksibilnost u implementaciji, a s druge omogućava visok stepen iskorišćavanja postojećeg koda za implementaciju tipskih komponenti koje zahtevaju manje izmene. Na slici 10. dat je dijagram klasa koji sadrži osnovne klase i interfejse Encog framework-a.



Slika 10. Dijagram osnovnih klasa Encog framework-a

Osnovni interfejsi su u Encog-u su:

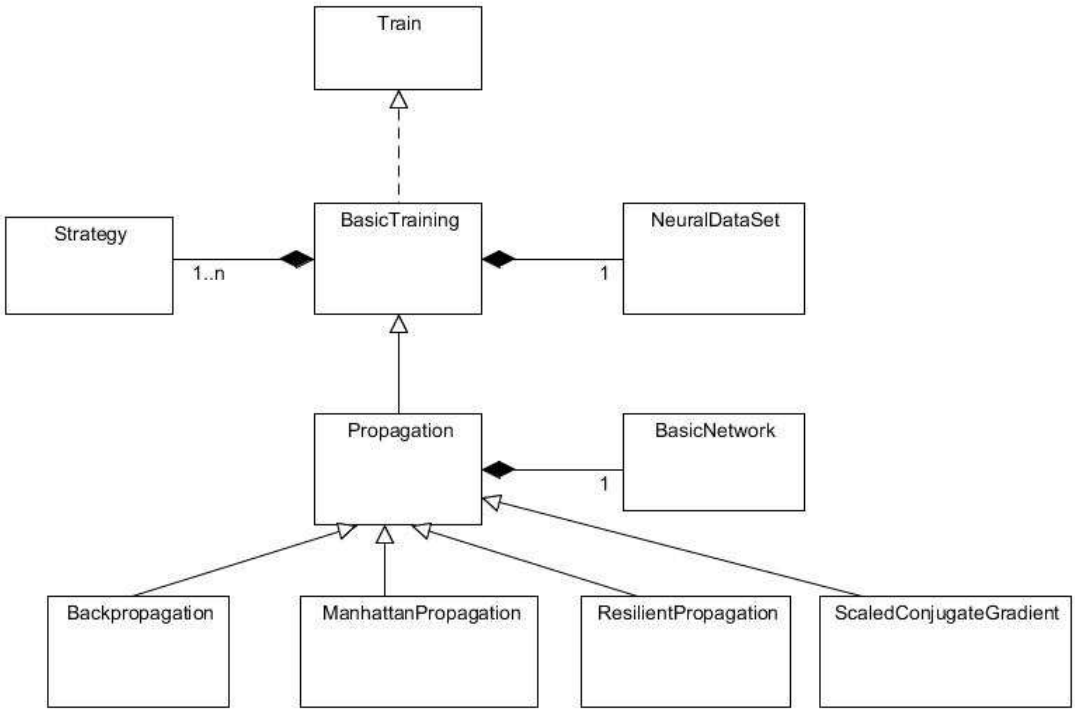
- **Layer** - definiše metode za rad sa slojevima neurona;
- **Synapse** – definiše metode za rad sa vezama između slojeva neurons;
- **ActivationFunction** - definiše metode potrebne za rad sa aktivacionom funkcijom neurona;
- **Network** – definiše metode za rad sa neuronskom mrežom;
- **NeuralLogic** – definiše metode kojima se realizuje interna logika izračunavanja neuronske mreže.

Osnovne klase u Encog-u su:

- **BasicLayer** - implementira Layer interfejs i predstavlja osnovnu klasu kojom se realizuje sloj neurona. Specifična ponašanja za slojeve neurona se realizuju nasleđivanjem ove klase. Ima atribut activationFunction (interfejs ActivationFunction) koji predstavlja aktivacionu funkciju za sve neurone u sloju.

- **BasicSynapse -** implementira Synapse interfejs i predstavlja osnovnu klasu kojom se realizuju veze između dva sloja neurona. Različiti načini povezivanja slojeva neurona se kreiraju nasleđivanjem ove klase.
- **BasicNetwork -** implementira Network interfejs i predstavlja osnovnu klasu kojom se realizuje neuronska mreža. Ona objedinjuje klase koje predstavljaju arhitekturu mreže (NeuralStructure) i logiku izračunavanja (NeuralLogic) i obezbeđuje javni Network interfejs;
- **NeuralStructure** – sadrži informacije o arhitekturi neuronske mreže (slojevim aneurona i vezama).

Na slici 11. dat je dijagram klasa pomoću kojih se realizuju algoritmi za učenje neuronskih mreža u Encog framework-u.

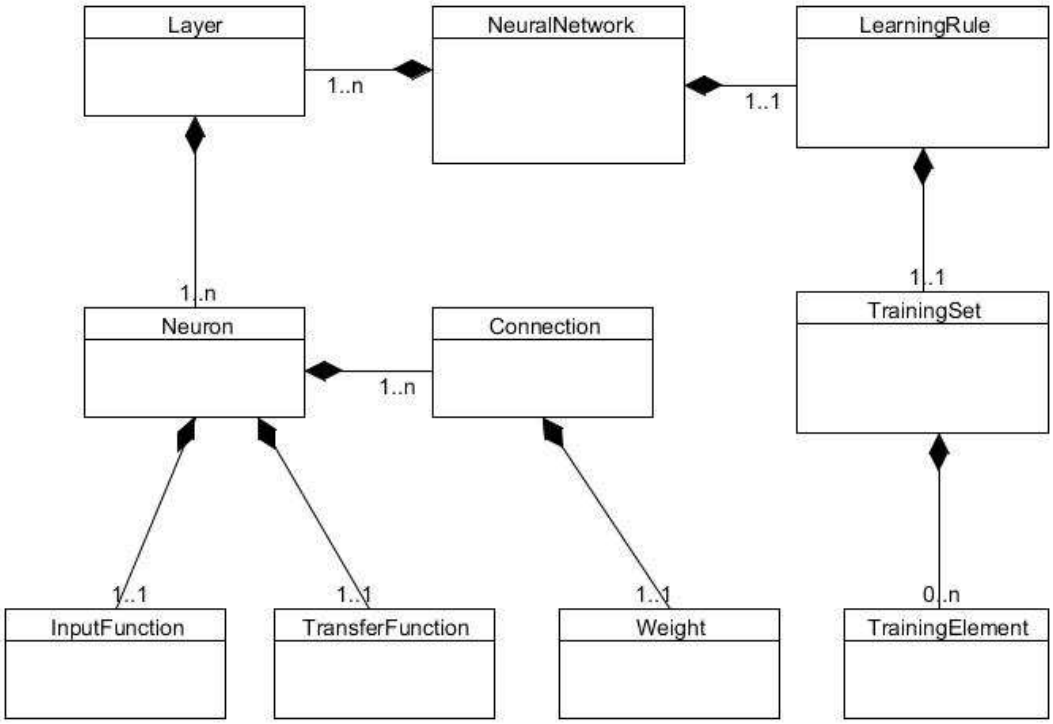


Slika 11. Dijagram klasa – algoritmi za učenje neuronskih mreža

- **Train** – interfejs, definiše opšte metode za upravljanje treningom/učenjem mreže;
- **BasicTraining** – osnovna klasa koja implementira Train interfejs i iz koje se nasleđivanjem izvoce razni algoritmmi za učenje;
- **NeuralDataSet** – skup podataka za trening neuronske mreže;
- **Strategy** – interfejs koji se koristi za dodavanje dodatnih strategija osnovnom algoritmu za trening;
- **Propagation** – osnovna klasa za familiju algoritama za učenje koji se zasnivaju na propagaciji greške;
- **Backpropagation, ManhattanPropagation, resilientPropagation, ScaledConjugateGradient** – algoritmi za učenje neuronskih mreža tipa višeslojni perceptron.

4.1.3. Arhitektura Neuroph framework-a

Dizajn Neuroph-a u potpunosti prati osnovne koncepte domenskog modela neuronskih mreža. Osnovne klase obezbe]uju opštu strukturu, logiku i komponente koje su zajedničke za sve vrste neuronskih mreža. Konkretne vrste neuronskih mreža se kreiraju nasleđivanjem osnovnih klasa i redefinisanjem ili dodavanjem odgovarajućih funkcionalnosti. Na taj način
može se kreirati veliki broj neuronskih mreža i iskoristiti veliki deo već postojećih funkcionalnosti (*reusability*). Na slici 12 prikazan je dijagram osnovnih klasa Neuroph framework-a.

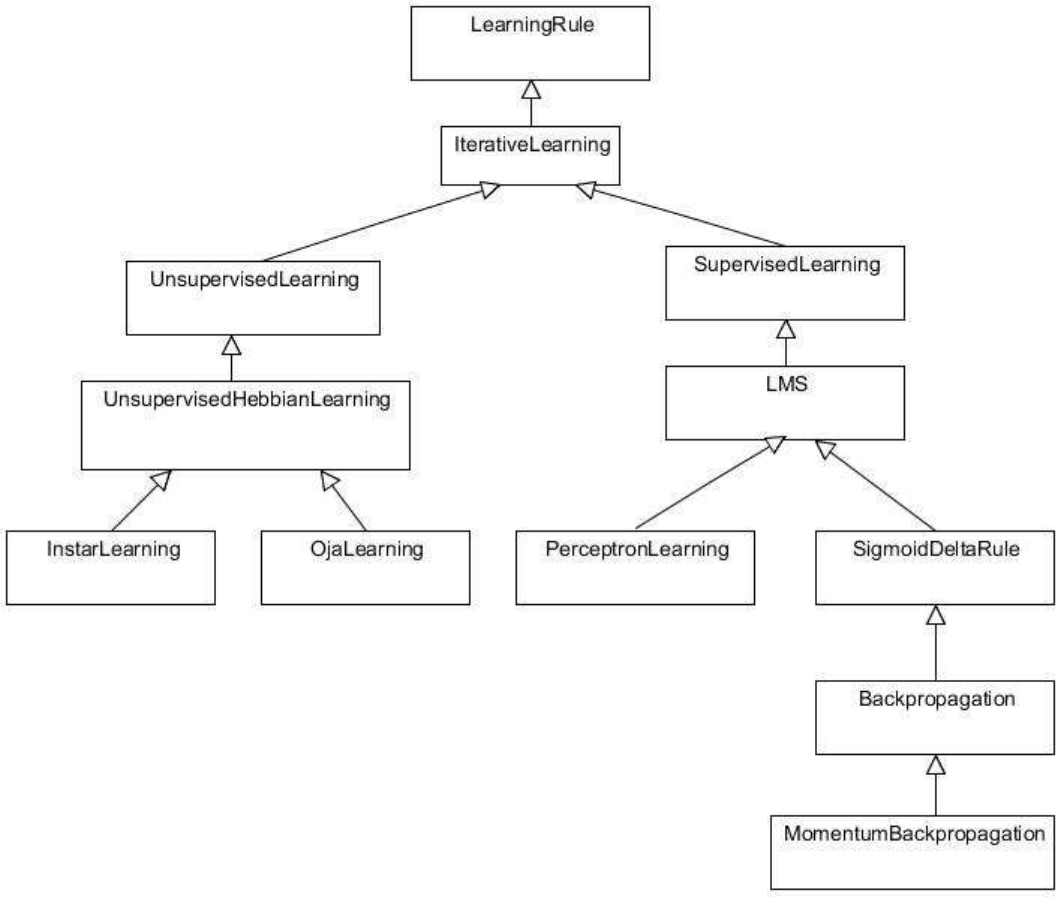


Slika 12. Dijagram osnovnih klasa Neuroph framework-a

- **Neuron**, osnovni element procesiranja u neuronskoj mreži, sadrži veze ka drugim neuronima (*Connection)*, i logiku za izračunavanje (*InputFunction* i *TransferFunction*).
- **Connection**, veza između dva neurona, sadrži reference ka neuronima koje povezuje i odgovarajući težinski koeficijent te veze (*Weight* objekat)
- **Weight**, predstavlja težinski koeficijent koji se dodeljuje vezi između dva neurona. Ovo je na neki način ključna klasa, jer se učenje kod neuronskih mreža zasniva na podešavanju težinskih koeficijenata veza.
- **InputFunction**, predstavlja ulaznu funkciju koja na osnovu ulaza dobijenih iz drugih neurona i težinskih koeficijenata ulaznih veza računa ukupni ulaz za određeni neuron.

- **TransferFunction**, aktivaciona funkcija ili funkcija transfera uzima ukupni ulaz određenog neurona i računa izlaz neurona.
- **Layer**, gradivni element neuronske mreže koji predstavlja sloj/ kolekciju/skup neurona.
- **NeuralNetwork**, opšta osnovna klasa za neuronske mreže. Specifične vrste neuronskih mreža se kreiraju nasleđivanjem ove klase i dodavanjem metoda za kreiranje odgovarajućih komponenti i podešavanja.
- **LearningRule**, osnovna klasa za sve algoritme za učenje. Konkretni algoritmi za učenje se kreiraju nasleđivanjem ove klase i dodavanjem odgovarajuće logike algoritma u izvedeenim klasama.
- **TrainingSet**, skup elemenata za trening sadrži skup podataka koje neuronska mreža treba da nauči.
- **TrainingElement**, predstavlja jedan element iz skupa podataka koje neuronska mreža treba da nauči.

Na slici 13. dat je dijagram klasa pomoću kojih se realizuju algoritmi za učenje neuronskih mreža u Neuroph framework-u.



Slika 13. Dijagram klasa – algoritmi za učenje neuronskih mreža

Iz datog dijagrama se vidi hijerarhija nasleđivanja

- **LearningRule**, osnovna klasa za sve algoritme za učenje;
- **IterativeLearning**, osnovna klasa za sve iterativne algoritme za učenje;
- **SupervisedLearning**, osnovna klasa za sve supervizorne algoritme za učenje;
- **UnsupervisedLearning**, osnovna klasa za sve ne-supervizorne algoritme za učenje;
- **LMS**, implementacija algoritma za učenje zasnovanog na metodi najmanjeg kvadrata. Ova klasa je takođe osnovna za sve algoritme za učenje koji se zasnivaju na korišćenju MNK metode.
- **Backpropagation**, implementacija najčešće korišćenog algoritma za učenje neuronskih mreža tipa višeslojni perceptron, koja indirektno nasleđuje klasu LMS. Ova klasa je takođe osnovna klasa za sve varijacije backpropagation algoritma kao npr. MomentumBackpropagation.

4.1.4. Uporedna analiza opštih karakteristika predstavljenih arhitektura

U tabeli 13 data je uporedna analiza opštih karakteristika predstavljenih softverskih framework-a, sa aspekta korisnika. Objašnjenje razmatranih karakteristika dato je u daljem tekstu, a svaka karakteristika je ocenjena na skali od 1 do 3. Ocene su date kao razultat analize izvršene tokom ovog istraživanja, kao i analize drugih autora [ref, comparing 3 reference]

Tabela 13. Uporedni pregled opštih karakteristika predstavljenih framework-a

Karakteristike	**JOONE**	**Encog**	**Neuroph**
Struktura	1	2	3
Fleksibilnost/proširivost	1	2	3
API	1	2	3
Jednostavnost korišćenja	1	2	3
Performanse	2	3	1
Prosečna ocena:	1.2	2.2	2.6

Struktura

Da li je struktura framework-a, posmatrano u celini, jednostavna i razumljiva?
Da li su jasno definisane uloge klasa i logika odnosa izmedju klasa?

Fleksibilnost/proširivost

Koliko je jednostavno dodavanje novih vrsta neuronskih mreža i algoritama za učenje?
U kojoj meri izmene na jednom mestu, zahtevaju izmene i u drugim mestima (povezanim klasama)?
Koliki je stepen iskorišćenja postojećeg koda (resusability) prilikom proširenja?

Programski interfejs (API)

Da li se iz programskog interfejsa (API) nazire logika operacija, i da li je intuitivno jasna?
Da li je logična organizacija paketa?

Jednostavnost korišćenja i primene

Jednostavnost kreiranja instanci neuronskih mreža i algoritama za učenje.
Jednostavnost primene za konkretne probleme.

Performanse

Brzina učenja i izračunavanja neuronskih mreža u slučaju većih skupova podataka i većih neuronskih mreža.

Iz datog uporednog pregleda, vidi se da od prikazanih framework-a, Neuroph ima najveći stepen fleksibilnosti u pogledu proširenja, intuitivnu strutkuru i najjednostavniji programski interfejs. Navedne karakteristike su od velikog značaja za softerske framework-e otvorenog koda, i iz tog razloga je Neuroph framework izabrana kao osnova za razvoj u ovom istraživanju, kroz koji su demonstrirani ključni principi softverskog inženjerstva u oblasti inteligentnih sistema.
Iz pregleda se takođe vidi da su slabije performanse glavni nedostatak Neuroph-a u odnosu na druge framework-e, i da tu postoji otvorena mogućnost za unapređenje. Jedno već postojeće rešenje za problem performansi, je već pomenuta integracija sa Encog framework-om.

4.2. Analiza strukture i načina proširenja Neuroph framework-a

U odeljku 4.1.3. data je analiza osnovnih domenskih klasa Neuroph framework-a. U ovom odeljku dat je pogled na globalnu organizaciju framework-a po paketima, i načina za proširenje framework-a, odnosno dodavanje novih funkcionalnosti.
U tabeli 14 data je logička i fizička organizacija framework-a po paketima.

Tabela 14. Organizacija Neuroph framework-a po paketima

Naziv paketa	**Opis**
org.neuroph.core	Sadrži osnovne klase i komponente neuronskih mreža. Ove osnovne klase se nasleđuju prilikom kreiranja specifičnih vrsta neuronskih mreža.
org.neuroph.core.input	Sadrži osnovne klase i standardne ulazne funkcije za neurone
org.neuroph.core.transfer	Sadrži osnovne klase i standardne funkcije transfera za neurone
org.neuroph.core.learning	Sadrži osnovne klase za algoritme za učenje neuronskih mreža
org.neuroph.nnet	Sadrži gotove specifične vrste neuronskih mreža.
org.neuroph.nnet.comp	Sadrži komponente za specifične vrste neuronskih mreža.
org.neuroph.nnet.learning	Sadrži implementacije algoritama za učenje specifičnih vrsta neuronskih mreža.
org.neuroph.util	Sadrži razne pomoćne klase za rad sa neuronskim mrežama (za kreiranje i primenu)
org.neuroph.util.plugins	Sadrži razne dodatne funkcionalnosti za neuronske mreže u vidu plugin-ova

4.2.1. Načini proširenja Neuroph framework-a

Neuroph obezbeđuje gotovu strukturu i komponente za kreiranje novih vrsta neuronskih mreža. Kreiranje novih vrsta neuronskih mreža se vrši nasleđivanjem osnovne (*base*) klase za neuronske mreže *NeuralNetwork* i implementiranjem metode za konstruisanje mreže *createNetwork*. Standardne komponente neuronskih mreža su takođe već implementirane u okviru framework-a, ali takođe je moguće i dodavanje novih to nasleđivanjem klasa *Neuron, TransferFunction, LearningRule* itd.

Postoji nekoliko pomoćnih (utility) klasa koje obezbeđuju metode za parametrizovano kreiranje komponenti neuronskih mreža, koje se koriste u okviru pomenute metode *createNetwork*. To su: *NeuronFactory, LayerFactory* i *ConnectionFactory*, koje se koriste za kreiranje neurona, slojeva neurona i veza između neurona.

Nove vrste neurona

Nove vrste neurona se kreiraju nasleđivanjem klase *org.core.Neuron*. Najčešće korišćene vrste neurona su već implementirane u klasama *InputNeuron, InputOutputNeuron, BiasNeuron, CompetitiveNeuron, DelayedNeuron* and *ThresholdNeuron* (sve se nalaze u paketu *org.neuroph.nnet.comp*). Zahvaljujući tome, za kreiranje novih načina izračunavanja neurona obično je dovoljno da se kreiraju nove funkcije za ulaz i transfer nasleđivanjem klasa *InputFunction i TransferFunction,* i da se instance tih funkcija proslede konstruktoru odgovarajuće vrste Neurona: *Neuron(InputFunction inputFunction, TransferFunction transferFunction).*

Nove vrste slojeva neurona

Standardna klasa *Layer* koja predstavlja sloj neurona, bez izmena odgovara potrebama velikog broja neuronskih mreža. Uloga ove klase je da služi kao osnovna kolekcija neurona, da obezbeđuje manipulaciju neuronima (dodavanje i izbacivanje) i izračunavanje neurona. Standardni način izračunavanja neurona u okviru jednog sloja je sekvencijalno izračunavanje (jedan po jedan neuron redom), ali moguće je naslediti ovu klasu i redefinisati metodu za izračunavanje kako bi se omogućio i neki drugi način izračunavanja. Ovo je urađeno u klasi *CompetitiveLayer* gde se neuroni izračunavaju iterativno u petlji sve dok ne ostane samo jedan aktivan neuron.

Nove vrste neuronskih mreža

Nove vrste neuronskih mreža se kreiraju nasleđivanje osnovne klase *org.core.NeuralNetwork.* Nova vrsta neuronske mreže treba da obezbedi metodu *createNetwork* koja treba da kreira instance slojeva neurona, neurone i algoritam za učenje. Konstruktor nove vrste neuronske mreže treba da prihvata ulazne parametre koji su specifični za određenu vrstu neuronskih mreža i prosledi ih metodi *createNetwork().*Komponente neuronske mreže se mogu kreirati pomoću odgovarajućih *factory* klasa (što je preporučeni način) ili neposrednim instanciranjem.

Nove vrste algoritama za učenje

Algoritam za učenje je najvažniji deo neuronske mreže jer obezbeđuje sposobnost učenja mreži. Novi algoritmi za učenje se kreiraju nasleđivanjem klase *org.neuroph.core.learning.LearningRule* ili neke od njenih podklasa, u zavisnosti od vrste algoritma za učenje. Na primer, za kreiranje supervizornog algoritma [ref] treba naslediti klasu *SupervisedLearning*, a za kreiranje algoritma zasnovanog na LMS (Least Mean Squares[ref]) učenju, još praktičnije je naslediti klasu LMS. Opšti princip je da za svaku familiju algoritama za učenje postoji osnovna klasa koja obezbeđuje osnovnu funkcionalnost, a konkretni algoritmi treba da dodaju svoje specifičnosti. Prilikom nadogradnje osnovnu klasu uvek treba izabrati tako da se maksimizira korišćenje postojećeg koda. Time se pored uštede vremena, izbegava dupliranje koda.

Sistem *Plugin-ova*

Sistem plugin-ova se koristi za dodavanje dodatnih funkcionalnosti neuronskim mrežama, pri čemu se one ne smatraju funkcionalnostima iz samog domena neuronskih mreža, već su vezane za određenu specifičnu primenu. Npr. funkcionalnosti vezane za prepoznavanje slika i karaktera su dodate kao plugin-ovu (*ImageRecognitionPlugin* i *OcrPlugin*). Ovakav pristup omogućava da se glavna hijerarhija nasleđivanja oslobodi detalja koji su vezani za specifičnu primenu, i čini osnovne klase jednostavnijim. Kreiranje novog plugin-a se vrši nasleđivanjem klase *PluginBase*.

4.3. Opšti model softverskog *framework-a*

U ovom odeljku, na osnovu prethodne analize Neuroph programskog okvira, dat je pregled opštih elemenata koje jedan programski okvir iz oblasti inteligentnih sistema treba da obezbedi. Navedeni skup elemenata se može posmatrati i kao opšti model programskog okvira.

Elementi programskog okvira:

- Domenski model, obuhvata osnovni konceptualni model, pri čemu svaki koncept ima odgovarajuću klasu. Domenski model definiše osnovnu strukturu/dizajn framework-a i predstavlja nepromenljiv deo (*frozen spot*);
- Apstraktni sloj, obuhvata osnovne klase i interfejse, predstavla jezgro programskog okvira. Apstraktni sloj je promenljivi deo framework-a, koji korisnik treba da definiše odnosno redefiniše i predstavlja tačke proširenja (*hot spots*, *extension points*);
- Konkretni sloj, u slu;aju Neuroph-a predstavlja implementacija konkretnih neuronskih mreza i algoritama nad apstraktnim slojem. Konkretni sloj sadrži implementaciju standardnog/podrazumevanog ponašanja framework-a.
- *Factory* klase – obezbeđuju jednostavno parametrizovano kreiranje domenskih objekata (korišćenjem factory i builder paterna);
- Globalna konfiguracija/podešavanja programskog okvira (korišćenjem singleton paterna)
- Javni programski interfesj ka korisniku (API), orijentisan na konkretne zadatke i u skladu sa domenskim modelom;
- Sistem pluginova za proširenja;
- Razne pomoćne klase koje olakšavaju primenu i rad sa programskim okvirom.

Pored osnovnih principa OO dizajna, osnovni principi projektovanja programskog okvira prema ovom modelu su:

1. **Klase programskog okvira treba da prate domenski model.** Zahvaljujući tome programski okvir će biti intuitivno razumljiv onima koji poseduju domensko znanje.

2. **Programski interfejs orijentisan na primenu.** Korisnici žele da primene softver za rešavanje određenog problema, a ne da uče tehnološke detalje programskog okvira.

3. **Sakriti složene detalje implementacije.** Sve što nije od interesa za krajnjeg korisnika ne treba da bude u javnom programskom interfejsu (API).

4. **Fleksibilni dizajn.** Omogućiti korisniku da prilagodi programski okvir svojim potrebama (pri tom predvideti tipične scenarije korišćenja).

5. **Maksimizirati korišćenje postojećeg koda, ali ne na štetu drugih principa.** Ovaj princip ukazuje da treba kontrolisati kreiranje apstrakcija i dekompoziciju složenih struktura i ponašanja jer to za posledicu može da ima komplikovanje strukture programskog okvira i usložnjavanje javnog programskog interfejsa ka korisniku.

6. **Kreiranje pomoćnih klasa koje pojednostavljuju tipične slučajeve korišćenja programskog okvira.**

7. **Fizički dizajn i logički dizajn.** Fizički dizajn odnosno organizacija klasa po paketima treba da bude usklađena sa logičkim domenskim dizajnom.

8. **Omogućiti jednostavnu promenu, održavanje i proširivanje.** Klase treba da budu slabo povezane tako da se eventualne izmene odražavaju na što manje celine. Definisati tačke proširivanja, odnosno način na koji će se dodavati nove funkcionalnosti za kojim ase ukaže potreba, kako bi se usmeravao dalji razvoj programskog okvira.

4.3.1. Dizajn javnog programskog interfejsa (API)

Javni programski interfejs (Application Programming Interface - API) je iz perspektive korisnika/programera najvažniji elemenat svakog framework-a. Programski interfejs predstavlja skup klasa i metoda koje korisnik framework-a može da koristi, a dizajn programskog interfejsa zapravo određuje na koji će način korisnik da koristi framework u svom programskom kodu. Od dizajna API-a zavisi i mogućnost framework-a da se vremenom dalje razvija, a samim tim u krajnjoj liniji i sam uspeh framework-a. U ovom odeljku ukratko su navedeni glavni principi za dizajn dobrog programskog interfejsa.

Dobro projektovan programski interfejs (API) ima sledeće karakterisitke:

1. jednostavan je za učenje i pamćenje;
2. obezbeđuje kreiranje čitljivog, razumljivog programskog koda;

3. onemogućava nepravilno korišćenje framework-a;
4. jednostavan je za proširivanje.

Jednostavan za učenje i pamćenje. Programski interfejs treba da bude intuitivan i dosledan u pogledu imenovanja klasa, paketa i metoda. Jednostavno, ako je korisniku iz naziva klase ili metode jasno koja je njihova uloga, i čemu služe, to će u velikoj meri olakšati učenje i pamćenje programskog interfejsa.
Treba težiti što manjem broju klasa i što jednostavnijoj strukturi (tzv. minimalistički pristup), jer što je veći broj klasa to ih je teže naučiti, a što je komplikovanija struktura to postoji više mogućnosti da se vremenom pojave problemi.
Programski interfejs koji je jednostavan za učenje, omogućava korisniku da kreira elementarni 'hello world' primer u svega nekoliko linija koda, a zatim da ga postepeno proširuje kako bi dobio složenije programe.

Obezbeđuje kreiranje čitljivog, razumljivog programskog koda
Čitljiv kod ima odgovarajući nivo apstrakcije, što znači da ne skriva bitne detalje, ali i ne zahteva od programera da navodi nebitne detalje, u kontekstu problema koji rešava. Čitljivost koda kreiranog pomoću određenog API-ja uvek treba posmatrati iz ugla korisnika (programera) i proceniti na konkretnim, realnim primerima korišćenja. Preporuka je da sa sam API projektuje na osnovu realnih primera i željenog načina korišćenja.
Čitljiv programski kod se jednostavnije dokumentuje i održava tokom životnog veka aplikacije, manja je verovatnoća da če da sadrži bug-ove, i bug-ovi se lakše se uočavaju.

Onemogućava nepravilno korišćenje framework-a. Dobro projektovan API sprečava programera da ga nepravilno koristi, i dovede objekte ili čitavu aplikaciju u nekonzistentno stanje.
Sa dobro dizajniranim API-jem lakše se piše ispravan, nego neispravan programski kod, i podržava uobičajna dobra praksa u programiranju. Stvari na koje treba obratiti pažnju su kreiranje objekata, posebno složenih struktura objekata i obradu izuzetaka. Loša praksa je i npr. zahtevati da se metode pozivaju u određenom redosledu, ili da određene operacije imaju sporedne efekte koji nisu vidljivi.

Jednostavan za proširivanje. Od samog početka API treba projektovati imajući u vidu da će se u budućnosti proširivati. Vremenom će sasvim sigurno biti potrebno da se dodaju nove klase, postojećim klasama nove metode, a postojećim metodama novi parametri. Kako API raste, rastu i šanse da dođe do konflikta između postojećih i novih elemenata API-ja.
Održavanje kompatibilnosti sa prethodnim verzijama je najviši prioritet. To praktično znaći da softver koji je pisan sa nekom od prethodnih verzija API-ja, u idealnom slučaju bi trebalo bez ikakvih izmena da radi i sa novom verzijom. Ovo je često veoma teško 100% postići, ali tome treba težiti.

Praktični saveti za razvoj dobrog API-ja:

- definisati i upoznati zahteve koji treba da budu rešeni pomoću API;
- definisati API pre početka implementacije;
- testirati API na realnim slučajevima korišćenja;
- napisati nekoliko primera korišćenja API-ja;
- prikupljati i analizirati mišljenja korisnika AP-ja.

4.4. Analiza i projektovanje novih funckonalnosti

U ovom odeljku opisano je projektovanje novih funkcionalnosti Neuroph frameworka, u skladu sa zahtevima identifikovanim u poglavlju 3.3. , predviđenim načinima proširenja opisanim u odeljku 4.2 , i principima i preporukama za projektovanje framework-a datim u poglavlju 4.3. Projektovane su sledeće funkcionalnosti:

1. funkcije transfera;
2. normalizacija podataka;
3. tehnike za generisanje slučajnih težinskih koeficijenata;
4. ulazni adapteri;
5. izlazni adapteri;
6. nove vrste neuronskih mreža i algoritama za učenje;
7. podrška za merenje performansi (*benchmark*).

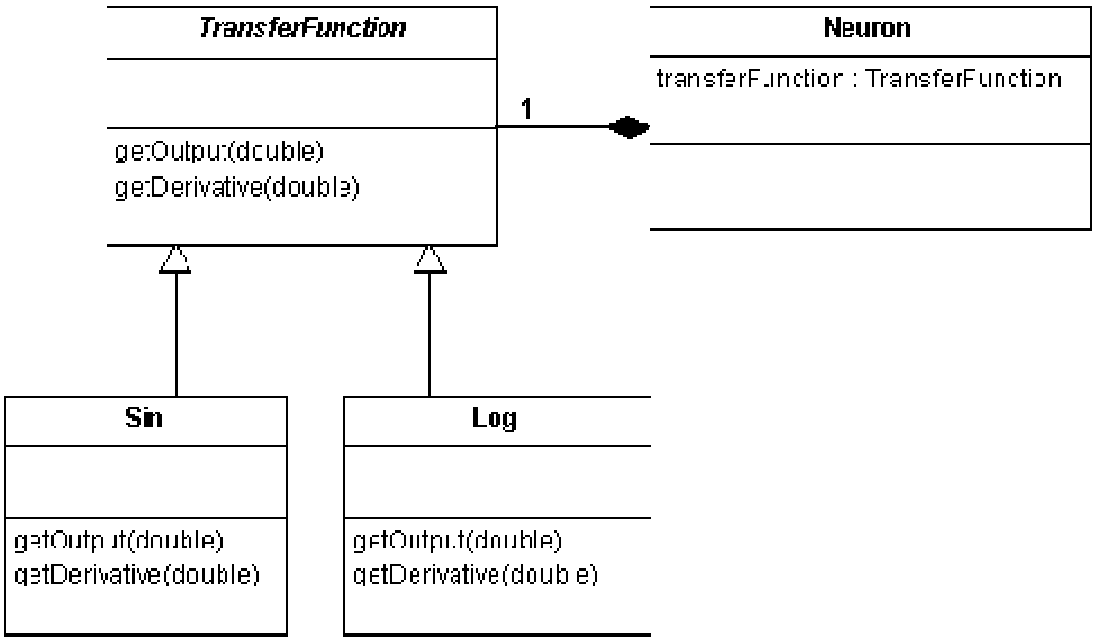
4.4.1. Nove funkcije transfera

Zahtev: Dodati nove vrste fukcija transfera za neurone: sin i log.

Svrha: Funkcija transfera sadrži glavnu logiku za izračunavanje neurona, i različite vrste neuronskih mreža koriste različite vrste funkcije transfera u neuronima. Dodavanjem novih vrsta funkcija transfera poboljšava se podrška za kreiranje novih vrsta neuronskih mreža.

Potrebne funkcionalnost: Funkcija transfera kao ulaz dobija težinsku sumu ulaza u neuron, a kao izlaz daje vrednost funkcije koju predstavlja. Pored toga potrebno je omogućiti izračunavanje prvog izvoda funkcije koju predstavlja jer se ta vrednost koristi u algoritmima za učenje.

Rešenje: Dodavanje novih vrsta funkcija transfera po potrebi, je predviđeno u osnovi strukture programskog okvira Neuroph. Neuroph definiše apstraktnu klasu *org.neuroph.core.transfer.TransferFunction* koja je osnovna klasa za sve sve funkcije transfera. Kreiranje novih funkcija transfera se realizuje nasleđivanjem klase TransferFunction, i redefinisanjem metoda *getOutput* i *getDerivative* koje izracunavaju i vracaju vrednost i prvi izvod funkcije za dati ulaz. Pored toga potrebno je dodati nove funkcije u registar funkcija transfera *org.neuroph.util.TransferFunctionTypes*.Na slici 14. dat je dijagram klasa koje realizuju funkcije transfera za neurone.



Slika 14. Klase *Sin* i *Log* nasleđuju apstraktnu klasu *TransferFunction*

Dodavanje novih funkcija transfera je očekivani zahtev od strane korisnika programskog okvira i zato je isprojektovan tako da se to može jednostavno izvesti nasleđivanjem postojeće apstraktne klase.
Zahvaljujući korišćenju mehanizma refleksije u pomoćnm klasama za kreiranje objekata (*NeuronFactory*), ceo framework može da koristi nove klase bez dodatnih izmena. Opisano rešenje je primer za dobro definisan scenario proširenja, koji korisniku daje jasnu tačku proširenja (*extension point*), usmerenu na logiku odnosno problem koji rešava, bez potrebe za velikim izmenama u ostatku programskog okvira.

4.4.2. Normalizacija skupa podataka za trening

Zahtev: Dodati podršku za normalizaciju podataka za trening. Normalizacija podrazumeva svođenje ulaznih vrednosti na interval [0,1] ili [-1, 1] jer se u okvru neuronskih mreža koriste vrednosti na tom intervalu.

Svrha: Pretvaranje realnih vrednosti iz skupa podataka za trening u vrednosti koje se koriste za učenje neuronskih mreža

Potrebne funkcionalnosti: Potrebno je obezbediti sledeće vrste normalizacije:

- **Max** – normalizacija u odnosu na najvecu vrednost u skupu
  $x_i = x_i / max\_x$
- **MaxMin** – normalizacija u odnosu na interval najmanje i najveće vrednosti u skupu
  $x_i = (x_I – min\_x) / (max\_x – min\_x)$
- **Decimalno skaliranje** – sve vrednosti u skupu se podele sa najmanjim 10^n tako da sve vrednosti budu manje od 1
  $x_i = x_i / 10^n$

Pored toga, potrebno je:
- obezbediti jednostavan programski interfejs (API) za korisnike;
- predvideti mogućnosti proširenja, tj. dodavanja novih metoda normalizacije od strane korisnika.

Rešenje: Realizacija navedenih zahteva moguća je na tri načina:

1. Klasi TrainingSet dodati odgovarajuće metode za sve tri vrste normalizacije.
U tom slučaju normalizacija podataka u instanci podataka za trening *trainingSet*, bi se vršila na sledeći način:

```
trainingSet.normalizeToMax()
trainingSet.normalizeToMinMax()
trainingSet.normalizeByDecimalScaling()
```

Dobra strana ovog rešenja je jednostavan i intuitivni programski interfejs (API) za krajnjeg korisnika. Loše strane ovog rešenja je nedostatak podrške za proširenje u

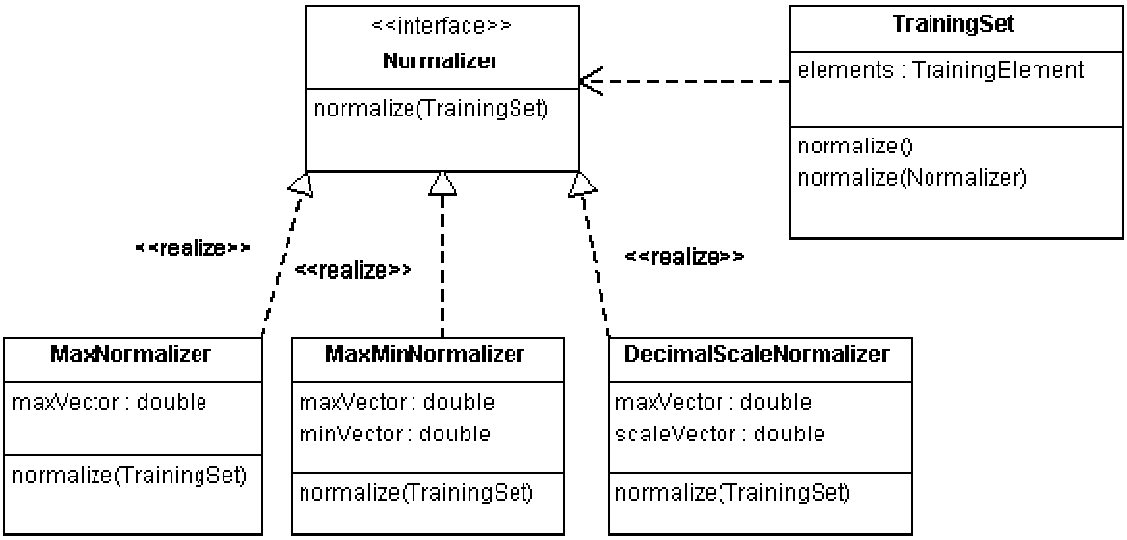
smislu implementacije novih vrsta normalizacije u OO maniru i prooptерećivanje klase *TrainingSet*. U slučaju da je potrebno dodati još neke nove vrste normalizacije, to bi se vršilo dodavanjem nove metoda klasi TrainingSet.

2. Kreiranje posebne klase sa statičkim metodama za normalizaciju
Jedan način na koji bi se delimično poboljšalo prethodno rešenje, je uvođenje nove klase *Normalizer*, koja bi sadržala statičke metode za normalizaciju i prihvatale kao parametar instancu klase *TrainingSet*. Normalizacija pomoću ove klase bi izgledala na sledeći način:

```
Normalizer.normalizeToMax(ts)
Normalizer.normalizeToMinMax(ts)
Normalizer.decimalScale(ts)
```

Na ovaj način klasa *TrainingSet* bi se rasteretila (ne bi potencijalno sadržala gomilu metoda za normalizaciju koje bi korisnici vremenom dodavali), i sva logika za normalizaciju bi bila na jednom mestu u posebnoj klasi. Programski interfejs je opet intuitivan, mada sada zahteva korišćenje još jedne klase za izvršenje operacije. Međutim mogućnost za nadogradnju je i dalje nezadovoljavajuća, jer se svodi na dodavanje novih metoda i izmenu osnovnih klasa progrmaskog okvira, što vremenom može da dovede do grananja i nekompatibilnosti između prilagođenih verzija framework-a koje koriste različiti korisnici.

3. Kreiranje interfejsa za normalizaciju, posebnih klasa za normalizaciju koje implementiraju taj interfejs i kombinovanje sa prethodna dva predloga rešenja.
Uvođenje interfejsa za normalizaciju i implementacija tog interfejsa za svaku metodu normalizacije je u potpunosti sa principima OO projektovanja. Na taj način otvorena je mogućnost da korisnici kreiraju nove klase za normalizaciju koje implementiraju odgovarajući interfejs, a da se pri tom te klase bez problema uklapaju svuda gde je predviđeno korišćenje normalizacije. To praktično znači i da ova proširenja ne zahtevaju izmene koda na drugim mestima u programskom okviru, izuzev dodavanja novih klasa. Na slici 15. dat je dijagram klasa projektovanog rešenja.



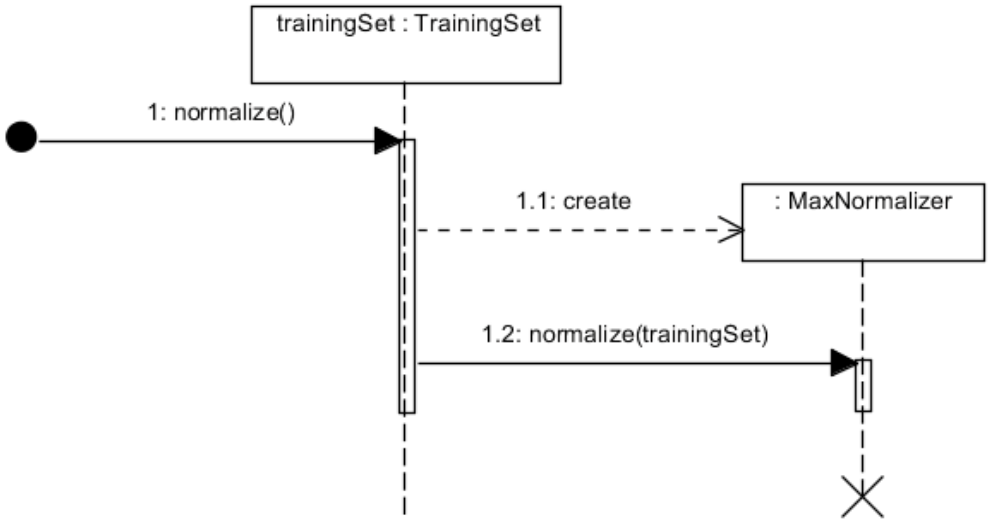
Slika 15. Dijagram klasa koje realizuju podršku za normalizaciju podataka

Da bi se obezbedio jednostavan i intuitivan programski interfejs, koji će korisniku omogućiti da jednim pozivom metode izvrši željenu operacije klasi TrainingSet dodate su odgovarajuče metode, koj eomogućavaju normalizaciju na sledeći način:

```
trainingSet.normalize(new MaxNormalizer()); // max normalizacija
trainingSet.normalize(new MaxMinNormalizer(); // max min
normalizacija
trainingSet.normalize(new DecimalScaleNormalizer();//decimalno
skaliranje
trainingSet,normalize(); // podrazumevana, max normalizacija
```

Na slici 16 dat je dijagram sekvenci izvršavanja operacije normalizacije, koja se izvršava na sledeći način:

1. Neka klijentska klasa poziva metodu za normalizaciju klase *TrainingSet*
2. Kreira se istanca klase koja neposredno izvršava normalizaciju, u ovom slučaju *MaxNormalizer*
3. Poziva se odgovarajuća metoda za normalizaciju koja predstavlja implementaciju interfejsa *Normalizer*



Slika 16. Dijagram sekvenci za slučaj izvršavanja podrazumevane Max normalizacije

Navedeno rešenje u potpunosti ispunjava nefunkcionalne zahteve, a to je da bude lako prošitivo, i da bude intuitivno za krajnje korisnike što su dva osnovna principa projektovanja celog Neuroph framework-a.

4.4.3. Tehnike za generisanje slučajnih brojeva

Zahtev: Dodati Neuroph-u podršku za generisanje slučajnih brojeva koje se koristi za inicijalizaciju težina veza između neurona.
Svrha: Istraživanja su pokazala da primena odre]enih tehnika za generisanje slučajnih brojeva za inicijalizaciju neuronskih mreža, obezbeđuje bolje rezultate prilikom učenja neuronskih mreža [ref widrow]. Pored toga, podrška za ove tehnike obezbediće osnovu za dalja istraživanaj u ovom smeru.

Potrebne funkcionalnosti: Po uzoru na *Encog* framework, potrebno je napraviti sledeće tehnike za inicijalizaciju težinskih koeficijenata slučajnim brojevima:

1. Range – sve težine se generišu kao slučajni brojevi na zadatom intervalu [min, max]

   weight = min + Math.random() * (max - min)

2. Distort – postojeće težine se modifikuju za slučajno generisanu vrednost koja se kontroliše parametrom

   weight = weight + (factor - (Math.random() * factor * 2))

3. Consistent – generiše se uvek isti niz slučajnih brojeva prema zadatom ključu (*seed*)

4. Gaussian – slučajni brojevi se generišu prema normalnoj/Gausovoj raspodeli prema Box/Muller-ovom algoritmu

5. Nguyen-Widrow – metoda namenjena višeslojnim perceptronima, koja uzima u obzir broj ulaza i broj skrivenih neurona. Prema iskustvu iz Encog-a, daje najbolje rezultate.

   ```
   beta = 0.7 * Math.pow(hiddenNeuronsCount, 1.0 / inputNeuronsCount)
   norm = Math.sqrt(sum(weight*weight));
   weight = beta * weight / norm
   ```

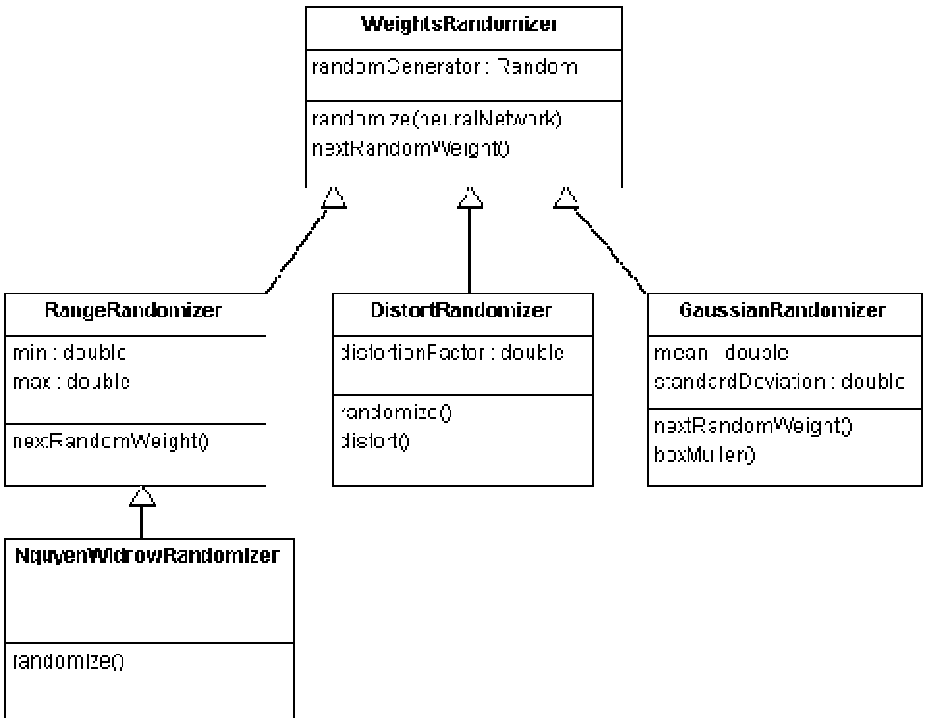
Rešenje: Iako Neuroph trenutno podržava inicijalizaciju težina slučajnim brojevima, postojeće rešenje nije dovoljno fleksibilno i nisu podržane napredne tehnike. Postojeće rešenje je realizovano pomoću tri metode u okviru klase *NeuralNetwork* i to:

1. *randomizeWeights()* – inicijalizuje težine na intervalu [-1, 1] i pomoću generatora slučajnih brojeva iz Java-e

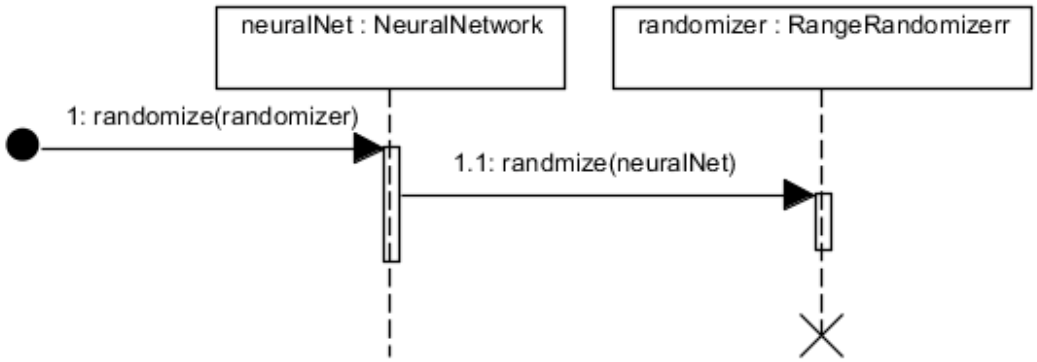
2. *randomizeWeights(double min, double max)* – inicijalizuje težine na zadatom intervalu takođe pomoću genratora slučajnih brojeva iz Java-e

3. *randomizeWeights(Random random)* – inicijalizuje težine pomoću zadate instance Java generatora slučajnih brojeva koja se prosleđuje kao parametar. Ovakav pristup omogućava generisanje istih sekvenci slučajnih brojeva prema zadatom ključu (*seed*)

Prateći postojeće rešenje, nove tehnike za generisanje slučajnih brojeva za inicijalizaciju težina, dodavale bi se kreiranjem novih metoda. Međutim to rešenje nije dobro jer bi dovelo do preopterećivanja glavne domenske klase metodama koje nisu uvek potrebne, moglo bi da bude uzrok nekompatibilnosti usled grananja različitih prilagođenih verzija, a ukoliko bi se dodavale u izvedenoj klasi ove funkcionalnosti ne bi bile na raspolaganju drugim vrstama neuronskih mreža.
Međutim, rešenje koje uvodi treća verzija metode *randomizeWeights(Random random)* iskorišćena je kao osnova za dalju nadogradnju. I u ovom slučaju je važno ispoštovati princip u projektovanju Neuroph-a, da se operacija može završiti jednim pozivom metode u kontekstu relevantne klase, i tako obezbediti jednostavan i intuitivan programski interfejs.
Iskorišćen je isti pristup kao i kod dodavanja funkcionalnosti normalizacije, odnosno klasi NeuralNetwork je dodata metoda koja kao parametra prima objekat klase koja predstavlja tehniku za generisanje slučajnih brojeva, pri čemu se različite tehnike realizuju kao podklase osnovne klase za generisanje slučajnih brojeva. Na slici 17 dat je dijagram klasa, a na 18 dijagram sekvenci koji ovo ilustruju.



Slika 17. Dijagram klasa koje relizuju generisanje slučajnih vrednosti težinskih koeficijenata



Slika 18. Dijagram sekvenci za generisanje slučajnih težinskih koeficijenata

Primeri korišćenja klasa za generisanje slučajnih vrednosti za težinske koeficijnte u Java kodu:

```
neuralNet.randomize(new RangeRandomizer(-2, 2));
neuralNet.randomize(new DistortRandomizer(0.7));
neuralNet.randomize(new NguyenWidrowRandomizer (0.1, 0.7));
neuralNet.randomize(new GaussianRandomizer (0.7, 0.1));
```

Iz datih primera Java koda vidi se da dato rešenje omogućava izvršenje operacije inicijalizacije težina slučajnim vrednostima pozivom jedne metode, kojoj se kao

parametar prosleđuje klasa koja implementira odgovarajuću tehniku. Dodavanje novih tehnika za generisanje slučajnih vrednosti težina može se lako realizovati nasleđivanjem osnovne klase WeightsRandomizer i preklapanjem odgovarajućih metoda.

4.4.4. Ulazni adapteri

Zahtev: Dodati podršku za učitavanje ulaznih podataka za neuronsku mrežu iz raznih izvora: ulaznog *stream*-a, fajla, URL-a, i baze podataka.

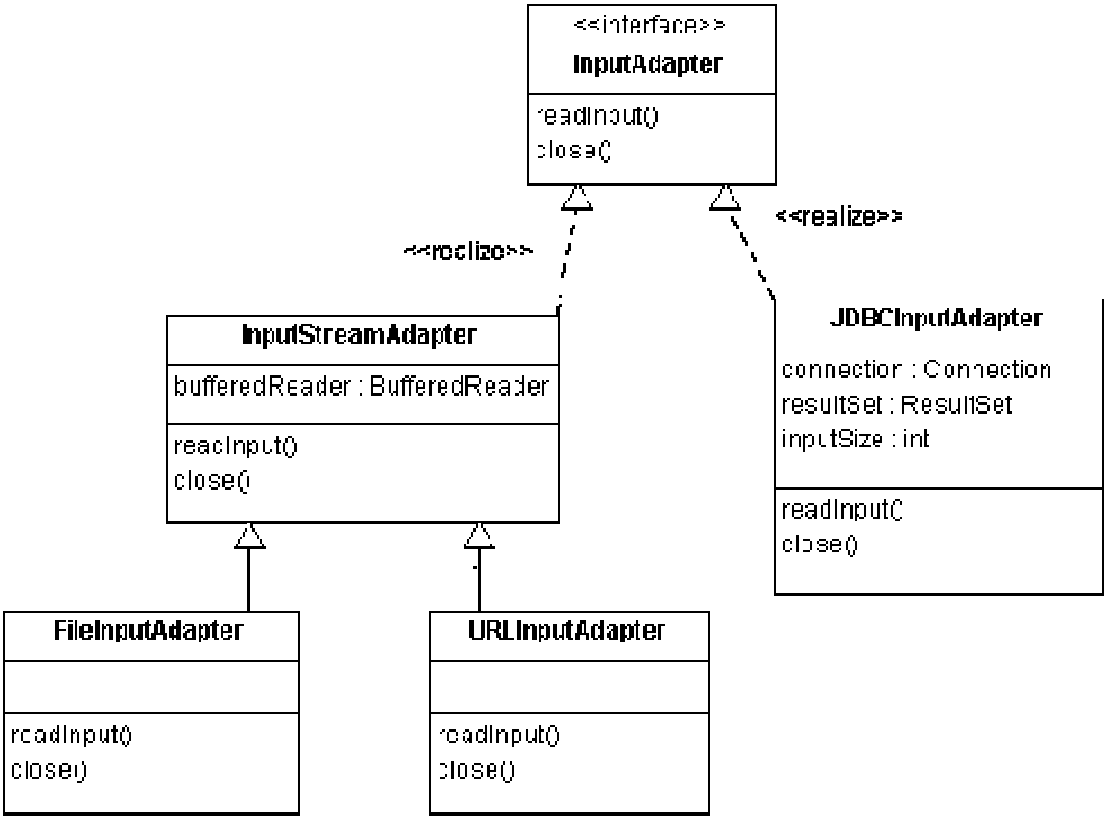
Svrha: Obezbediti jednostavnu integraciju i korišćenje Neuroph-a sa raznim izvorima podataka. Osnovni, tipični izvori podataka su kao što je navedeno u zahtevu: ulazni stream, fajl, URL i baza podataka.

Potrebne funkcionalnosti: Potrebno je obezbediti sledeće funkcionalnosti:

- Čitanje ulaza za neuronsku mrežu iz zadatog ulaznog stream-a;
- Čitanje ulaza za neuronsku mrežu iz zadatog fajla;
- Čitanje ulaza za neuronsku mrežu sa zadatog URL-a;
- Čitanje ulaza za neuronsku mrežu iz baze podataka pomoću zadatog upita.

Potrebno je projektovati fleksibilno i proširivo rešenje koje se lako može nadograditi za druge izvore podataka, po uzoru na standardni java.io API.

Rešenje: Osnovni princip za projektovanje rešenja u skladu sa navedenim zahtevima, je da postoji standardni interfejs za čitanje iz bilo koje vrste izvora podataka, i da za svaku vrstu izvora podataka postoji posebna implementacija tog interfejsa. To omogućava da isti programski kod koji koristi Neuroph neuronsku mrežu, može bez izmena da radi sa različitim izvorima podataka, pri čemu je potrebno samo da mu se prosledi odgovarajući adapter za čitanje podataka.
U osnovi rad sa ulaznim *stream-om*, fajl-om i URL-om se svode na rad sa *stream-ovima*, dok je rad za bazom standardno podrazumeva rad preko JDBC interfejsa.



Slika 19. Dijagram klasa koje realizuju ulazne adaptere

Na slici 19 dat je dijagram klasa koje realizuju ulazne adaptere, i ispunjavaju postavljene zahteve.
Svi ulazni adapteri implementiraju (ili nasleđuju) interfejs *InputAdapter* koji predstavlja opšti programski interfejs za sve izvore podataka koji se koriste kao ulaz za neuronsku mrežu. Interfejs se sastoji iz dve metode, i to:

1. *readInput* metode kojojm se čita i vraća sledeći ulaz za neuronsku mrežu i
2. *close* metode kojojm se zatvara izvor podataka nakom što su svi podaci iščitani.

Ovaj interfejs je definisan po uzoru na rad sa *stream-ovima* u standardnom java.io paketu.
Klasa *InputStreamAdapter* implementira čitanje iz ulaznih *stream-ova*, i predstavlja osnovnu klasu za klase FileInputAdapter i URLInputAdapter koje implementiraju čitanje iz fajlova i sa URL-a respektivno.
Klasa *JDBCInputAdapter* obezbeđuje čitanje ulaznih podataka za neuronsku mrežu iz baze preko *InputAdapter* interfejsa, koristeći standardnu Java JDBC podršku.
Dato rešenje u potpunosti odgovara na postavljeni zahtev da se omogući čitanje ulaznih podataka za Neuroph neuronsku mrežu iz ulaznog stream-a, fajla, URL-a i baze podataka. Takođe, po potrebi omogućava jednostavno proširivanje, odnosno dodavanje novih adaptera koji implemenetiraju interfejs *InputAdapter*, i obezbeđuju čitanje iz drugih vrsta izvora podataka.

4.4.5. Izlazni adapteri

Zahtev: Dodati podršku za upisivanje izlaznih podataka iz Neuroph neuronskih mreža u različite sisteme za prenos i skladištenje podataka, i to: ulazni *stream*, fajl, URL i bazu podataka.

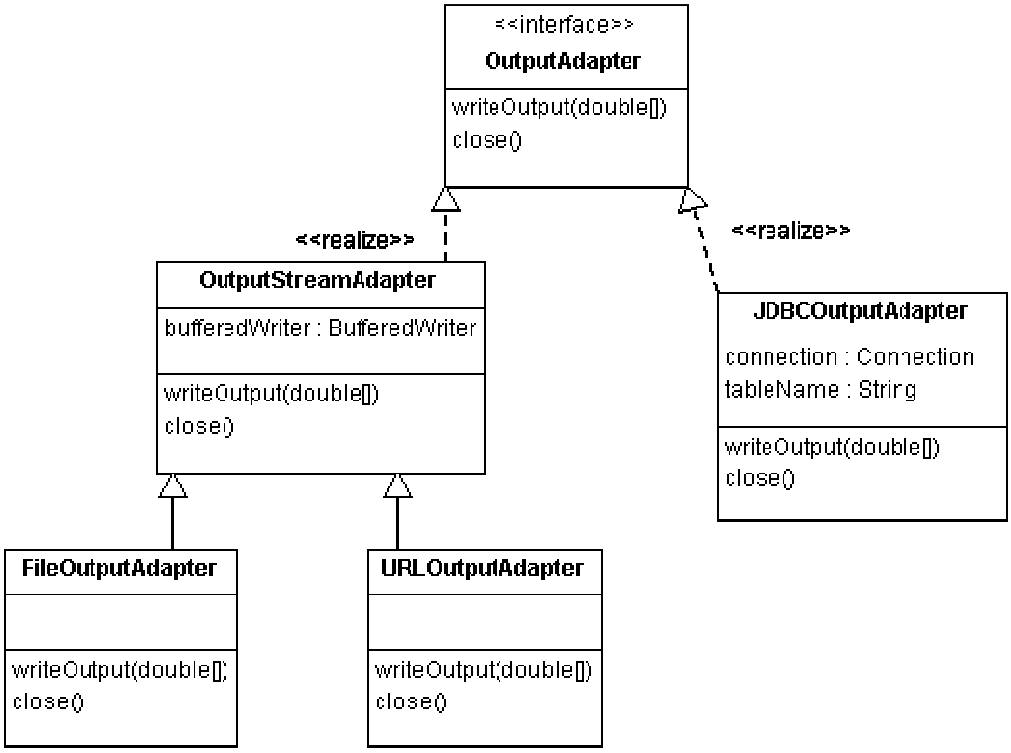
Svrha: Obezbediti jednostavnu integraciju i korišćenje Neuroph-a sa raznim načinima za prenos i skladištenje podataka. Osnovni, tipični načini prenosa i skladištenja podataka su kao što je navedeno u zahtevu: izlazni stream, fajl, URL i baza podataka.

Potrebne funkcionalnosti: Potrebno je obezbediti sledeće funkcionalnosti:

- Upisivanje izlaza neuronske mreže u zadati izlazni stream;
- Upisivanje izlaza neuronske mreže u zadati fajl;
- Upisivanje izlaza neuronske mreže na zadati URL;
- Upisivanje izlaza neuronske mreže u zadatu tabelu u bazi podataka.

Potrebno je projektovati fleksibilno i proširivo rešenje koje se lako može nadograditi i za druge načine prenosa i skladištenja podataka, po uzoru na standardni *java.io* API.

Rešenje: Rešenje za dizajn izlaznih adaptera dato je po istom principu kao i za ulazne adaptere, jer je su zahtevi praktično potpuno isti, s jedinom razlikom, što su kod ulaznih adaptera u pitanju operacije čitanja, a kod izlaznih operacije upisivanja. Ovaj princip je ispoštovan po uzoru na standardni *java.io* paket. U osnovi rešenja je standardni interfejs za upisivanje u bilo kakav izlaz, a za za svaku specifičnu vrstu izlaza postoji posebna implementacija tog interfejsa. Ovakav pristup omogućava da isti programski kod koji koristi Neuroph neuronsku mrežu, može bez izmena da radi sa različitim načinima za prenos i skladištenje podataka, pri čemu je samo potrebno da mu se prosledi odgovarajući adapter. U osnovi, rad sa ulaznim stream-om, fajl-om i URL-om se svode na rad sa *stream-ovima*, dok je rad za bazom standardno podrazumeva rad preko JDBC interfejsa. Na slici 20 dat je dijagram klasa koje realizuju izlazne adaptere.



Slika 20. Dijagram klasa koje relizuju izlazne adaptere

Svi izlazni adapteri implementiraju (ili nasleđuju) interfejs *OutputAdapter* koji predstavlja opšti programski interfejs za sve načine upisivanja podataka koji sa izlaza neuronske mreže. Interfejs se sastoji iz dve metode, i to:

1. *writeOutput* metode koja upisuje izlaz zadat kao ulazni parametar metode i
2. *close* metode kojojm se zatvara izlaz za upis podataka nakom što su svi podaci upisani.

Ovaj interfejs je definisan po uzoru na rad sa izlaznim *stream-ovima* u standardnom java.io paketu.

Klasa *OutputStreamAdapter* implementira upisivanje u izlazne *stream-ove*, i predstavlja osnovnu klasu za klase FileOutputAdapter i URLOutputAdapter koje implementiraju upis u fajlove i na URL respektivno.

Klasa *JDBCOutputAdapter* obezbeđuje upis izaznih podataka iz neuronske mreže u bazu preko *OutputAdapter* interfejsa, koristeći standardnu Java JDBC podršku.

Dato rešenje u potpunosti odgovara na postavljeni zahtev da se omogući upis izlaznih podataka iz Neuroph neuronske mreže u izlazni *stream*, fajl, URL i bazu podataka. Takođe, po potrebi omogućava jednostavno proširenje, odnosno kreiranje novih adaptera koji implemenetiraju interfejs *OutputAdapter* i omogućavaju upis u druge vrste izlaza/skladišta podataka.

4.4.6. Nove vrste neuronskih mreža i algoritama za učenja

Zahtev: Dodati podršku za nove vrste neuronskih mreža i algoritama za učenje, i dodati nove opcije postojećim. Dodati mreže i algoritme koji postoje kod framework-a Encog i JOONE.

Svrha: Obezbediti naprednije algoritme za učenje koji će omogućiti brže rešavanje problema, eksperimentisanje i uporedno poređenje nekoliko rešenja. Pored toga, sam framework će dobiti podršku za razvoj još većeg broja neuronskih mreža, što je njegova osnovna namena.

Potrebne funkcionalnosti: Potrebno je kreirati sledeće funkcionalnosti koje postoje kod framework-a Encog i JOONE a trenutno ih nema u Neuroph-u:

- Algoritam za učenje *ResilientPropagation*
- Neuronsku mrežu tipa *PCANetwork* i odgovarajući algoritam za učenje *GeneralizedHebbianLearning*
- Neuronsku mrežu tipa *InstarOutstar* i odgovarajući algoritam za učenje *InstarOutstarLearning*
- Neuronsku mrežu tipa *BoltzmanNetwork*, pri čemu za učenje koristiti već postojeći algoritam *HebianLearning*
- Neuronsku mrežu tipa *CounterPropagation* i odgovarajući algoritam za učenje *CounterPropagationLearning*

Rešenje: Neuroph framework je projektovan tako, da se nove vrste neuronskih mreža i algoritmi za učenje mogu nadograditi na postojeće osnovne klase uz maksimalno iskorišćenje postojećeg koda. Zahvaljujući tome sve zahtevane funkcionalnosti se mogu izvesti nadogradnjom postojećih klasa. Međutim da bi se došlo do optimalnog rešenja, neophodno je napraviti i neke izmene u osnovnim klasama kako bi odgovorile na zahteve novih vrsta neuronskih mreža. Tako, pored uvođenja novih funkcionalnosti dolazi i do evolucije osnovnog framework-a uz obavezno očuvanje kompatibilnosti sa prethodnim verzijama.

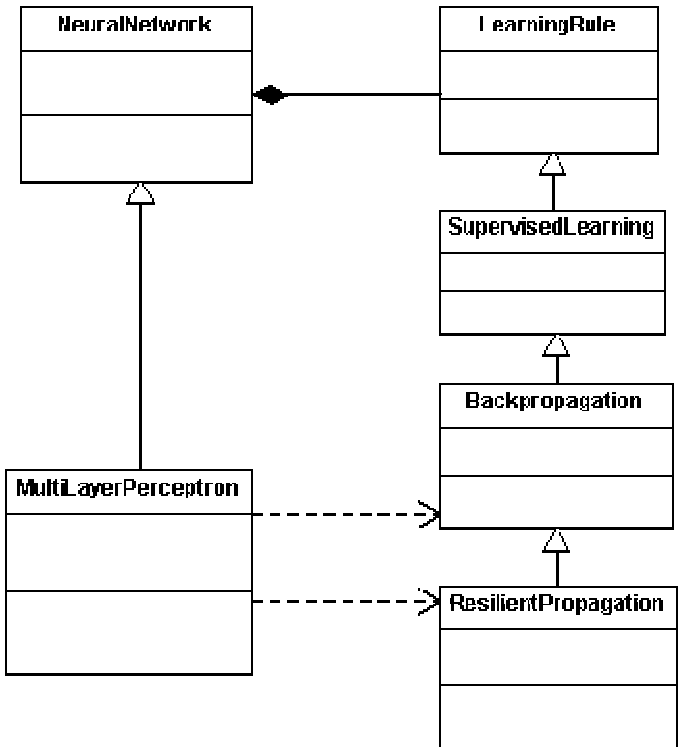
Zahtevane funkcionalnosti realizovane su na sledeći način:

1. Algoritam za učenje ResilientPropagation nasleđuje postojeći algoritam Backpropagation i koristi se za učenje neuronske mreže tipa MultiLayerPerceptron (slika 21). Algoritam ResilientPropagation predstavlja varijaciju osnovnog Backpropagation algoritma i zbog toga je osnovni princip rada oba algoritma isti, pa je ta osnova iskorišćena za kreiranje novog algoritma.

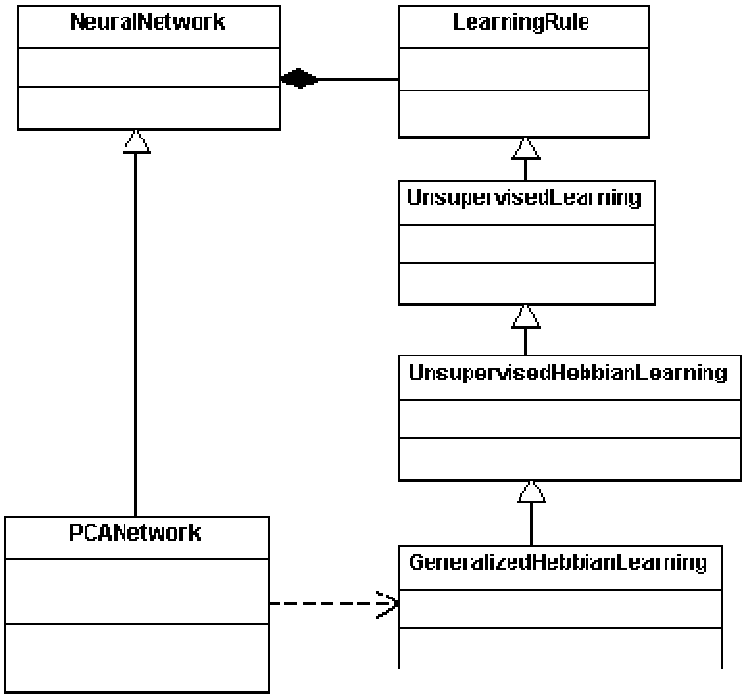
2. *PCANetwork* nasleđuje klasu *NeuralNetwork* i kao algoritam za učenje koristi *GeneralizedHebbianLearning* koji nasleđuje postojeću klasu *UnsupervisedHebbianLearning* (slika 22)

Na isti način kao i u prethodna dva slučaja (nasleđivanjem osnovne klase NeuralNetwork i odgovarajućeg algoritma za učenje) kreirane su i ostale vrste neuronskih mreža i to:

- *CounterPropagationNetwork* nasleđuje *NeuralNetwork* i predstavlj akombinaciju dve već postojeće mreže: kompetitivna mreže (*CompetitiveNetwork*) i Perceptron-a. Algoritam za učenje kombinuje *CompetitiveLearning* za učenje kompetitivnog sloja neurona, i supervizornog učenja za izlazni sloj neurona;
- *InstarOutstar* nasleđuje *NeuralNetwork* i koristi *InstarOutstarLearning* koji je implementiran po uzoru na *InstarLearning* i *OutstarLearning;*
- *BoltzmanNetwork* nasleđuje *NeuralNetwork*, implementiran je po uzoru na postojeću *BAM* mrežu kojoj je dodat jedan skriveni sloj neurona, i za učenje koristi *BinaryHebbianLearning*.



Slika 21. Dijagram klasa za algoritam ResilientPropagation



Slika 22. Dijagram klasa neuronske mreže PCANetwork i njenog algoritma za učenje GeneralizedHebbianLearning

4.4.7. Testiranje performansi – benchmark

Zahtev: Dodati podršku za testiranje performansi neuronskih mreža i algoritama za učenje.

Svrha: Podrška za testiranje performansi omogućava uporednu analizu performansi različitih implementacija algoritama, i poređenje sa drugim rešenjima. Time omogućava analizu i optimizaciju performansi softverskog framework-a.

Analiza problema: Testiranje performansi softvera za neuronske mreža je složen problem i može posmatrati sa na dva nivoa. Prvi nivo je algoritamski nivo, koji podrazumeva efikasnost algoritma za učenje, za određenu primenu ili klasu problema. Na ovom nivou performanse se ogledaju u broju iteracija treninga potrebnih da neuronska mreža 'nauči' određeni skup podataka.
Drugi nivo je implementacija, odnosno performanse određene softverske realizacije algoritama, i podrazumeva brzinu izvršavanja algoritma, odnosno protoka podataka i podešavanja parametara mreže.
Poređenje na algoritamskom nivou se u Neuroph-u lako može izvršiti na osnovu podataka o potrebnom broju iteracija, koji se neposredno može dobiti preko javnog programskog interfejsa (API) klasa koje realizuju algoritme za učenje.
Na nivou implementacije performanse se mogu meriti tzv. *micro-benchmark*-om, koji predstavlja često korišćeni način merenja performansi softvera. Micro-benchmark predstavlja merenje realnog vremena potrebnog za izvršavanje određene operacija ili skupa operacija u celini. Merenje vremena se vrši programski tako što se uzme sistemsko vreme u milisekundama pre i posle izvršenja određene operacije, i njihova razlika predstavlja potrebno vreme izvršavanja. Ovaj jednostavan princip i pored izvesnih nedostataka daje rezultate koji su korisni za optimizaciju softvera u cilju pobolj[anja performansi.
Obzirom da poređenje performansi na algoritamskom nivou već moguće sa postojećim funkcionalnostima, a da ne postoji nikakvo rešenje za testiranje performansi na nivou implementacije,
za razvoj je izabran micro-benchmark za testiranje implementacije neuronskih mreža iz Neuroph-a,
Testiranje performansi pomoću micro-benchmarka karakterišu razni problemi koje treba imati u vidu prilikom razvoja i analize rezultata. Pre svega, problem je nepredvidivi način izvršavanja programa na Java virtualnoj mašini (JVM) koji je posledica dinamičkog kompajliranja i automatskog upravljanja memorijom [Goetz 2004].
Java programi se prilikom kompajliranja prevode u Java byte code, koji se tek prilikom izvršavanja na određenoj hardverskoj platformi prevodi u mašinksi kod koji se izvršava.
Prve generacije JVM su koristile interpretere, što znači da se Java byte code uopšte i nije prevodio u mašinski kod u celini, već su se Java byte code naredbe jedna po jedna prevodile u mašinski kod i izvršavale tokom izvršavanja programa. Ovakav pristup veoma brzo je pokazao loše performanse. Pri tom, uopšte nije bila moguća optimizacija koda koju vrše statički kompajleri kao npr. kod programa pisanih u C i C++,

Sledeća generacija JVM, koristila je Just In Time (JIT) kompajlere kako bi se ubrzalo izvršavanje. Kod JIT kompilacije Java byte code se prevodio u mašinski kod odmah po startovanju programa. Dakle kompilacija se vrši na početku izvršavanja programa kako bi se izbeglo prevođenje u mašinske instrukcije tokom izvršavanja. Međutim, kako bi se skratilo vreme startovanja programa, JIT kompajler nije prevodio ceo program u mašinske instrukcije, već samo one delove progama za koje je pretpostavljao da će se izvršavati. Pored toga, optimizacija koda je bila minimalna opet kako bi se skratilo vreme startovanja programa.
Aktuelna, najnaprednija tehnika kompajliranja i izvršavanja programa na JVM je *HotSpot* dinamičko kompajliranje. HotSpot je tehnika koja kombinuje interpretaciju, profajling i dinamičko kompajliranje. Suština ovog pristupa je da se nakon izvesnog broja izvršavanja Java byte code-a, izvrši kompajliranje i optimizacija delova koda koji se najviše izvršavaju. Tokom izvršavanje interpretacijom, JVM prikuplja podatke o tome koji se delovi programa najviše izvršavaju, i na osnovu tih podataka određuje koji delovi programa će se kompajlirati. Pored toga, HotSpot kompajler ima dve moda: klijentski (*-client*) i serverski (*-server*) mod. U serverskom modu, kompajler vrši optimizaciju tako da obezbedi maksimalne performanse pri opterećenju aplikacija koje se dugo izvršavaju, dok se u klijentskom modu optimizacija vrši u smeru skraćivanja vremena startovanja i korišćenja memorije, i pri tom se vrši manji broj optimizacija.
Ono što je posebno interesantno kod HotSpot kompajliranje je tzv. kontinuirano kompajliranje i optimizacija. To znači da određeni segment programa koji je interpretiran određeni broj puta, i zatim iskompajliran, posle nekog vremena može biti ponovo iskompajliran i optimizovan ako postoji mogućnost da se taj segment bolje optimizuje.
Ovakav način izvršavanja Java programa treba imati u vidu jer je velika razlika u izvršavanju interpretiranog i iskompajliranog koda, a i samo kompajliranje i rekompajliranje tokom izvršavanja utiče na rezultate merenja brzine.
Zbog toga se koristi tzv. tehnika zagrevanja (*warmup*) koja podrazumeva izvršavanje testiranog koda određeni broj iteracijija, kako bi se JVM stabilizovala, odnosno iskompajlirala i optimizovala sve kritične delove programa, a zatim izvršiti merenje vremena.
Pored dinamičkog kompajliranja, problem predstavlja automatsko upravljanje memorijom i rad *GarbageCollector-a (GC),* koji ima zadatak da čisti memoriju od objekata koji se ne koriste. Ukoliko testirani kod intenzivno koristi memoriju, kreira mnogo novih objekata, aktiviraće se GC , čiji rad će takođe uticati na rezulatet merenja. Zato je potrebno JVM pokrenuti sa podešavanjima sa dovoljno *heap* i *stack* memorije.
Uz sve navedene probleme sa JVM, treba minimizovati uticaj drugih procesa koji se izvršavaju na računaru tokom testiranja performansi, odnosno isključiti sve procese i aplikacije koje potencijalno mogu da utiču na rezultate merenja.
I na kraju, zbog velikog broja faktora koji utiču, a koje nije moguće u potpunosti kontrolisati, testiranje treba ponoviti više puta, a zatim izvući statistiku – srednju vrednost i standardnu devijaciju.

Potrebne funkcionalnosti: U okviru rešenja za testiranje performansi potrebno je obezbediti:

1. merenje vremena potrebnog za izvršavanjeprograma ili dela programa
2. fleksibilan i prošitiv model za kreiranje korisničkih testova performansi

3. rešenje za probleme testiranja performansi opisane u fazi analize – dinamičko kompajliranje, upravljanje memorijom, i ‘zagrevanje’ testiranja;
4. generisanje objedinjenih rezultata/izveštaja o testiranju performansi;
5. pored tehni;kog modela, dato re[enje treba da obezbedi i metodolo[ki okvir za testiranje performansi.

Rešenje: Rešenje za micro-benchmark Neuroph-a, projektovano je tako da omogućava jednostavno kreiranje raznih testova peroformansi od strane korisnika i razvojnog tima Neuoph-a. Rešenje obuhvata glavne elemente koje imaju slični micro-benchmark alati (npr. JBench, Japex), a pri tom se išlo na to da bude što jednostavniji za korišćenje i malim brojem klasa.
Prilikom projektovanja rešenja, uzete su u obzir sve specifičnosti prilikom testiranja na JVM opisane prilikom analize problema. Na slici 23 dat je dijagram toka procesa testiranja, koji je osnova za realizaciju postavljenih zahteva. Osnovni koraci testiranja su:

1. Priprema testa
2. Zagrevanje (WarmUp)
3. Test performansi sai merenjem vremena izvršavanja
4. Generisanje statistike i izveštavanje

Priprema testa

Priprema test obuhvata pripremu podataka za testiranje i kreiranje svih objekata potrebnih za izvršavanje testa. U ovom konkretnom slučaju za Neuroph, to je kreiranje skupa podataka za trening i neuronske mreže.

Zagrevanje (WarmUp)

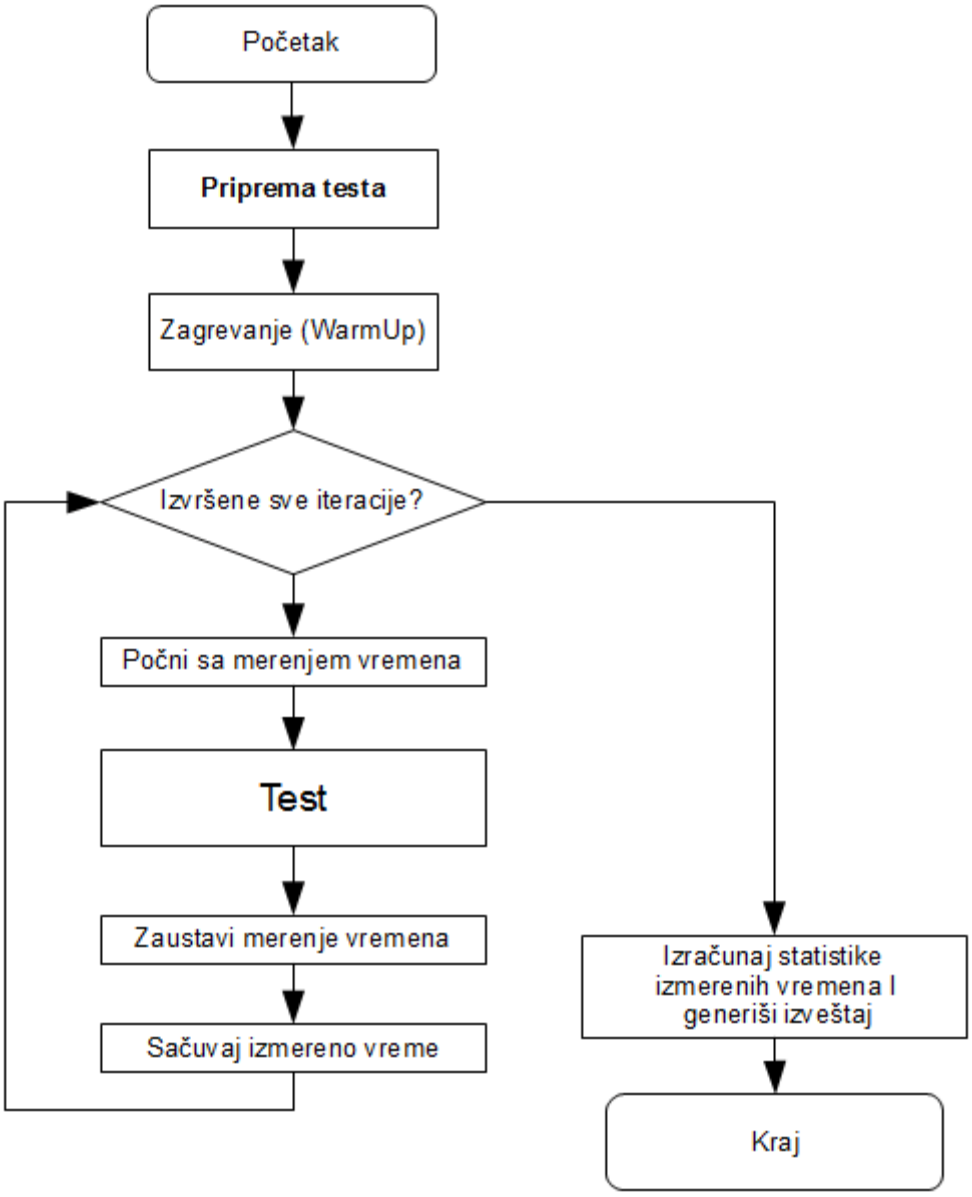
Zagrevanje, je pripremna faza tokom koje je programski kod koji se testira potrebno izvršiti određeni broj iteracija, kako bi test i JVM došli u stabilno stanje.Stabilno stanje je stanje u kome se tokom izvršavanja program više neće rekompajlirati i optimizovati, jer će se sve optimizacije izvršiti tokom ‘zagrevanja’. Takođe će biti kreirani svi objekti i programu će biti dodeljeno dovoljno memorije za izvršavanje.

Test performansi sa merenjem vremena izvršavanja

Sam test performansi podrazumeva izvršavanje programskog koda koji se testira određen broj iteracija, pričemu se za svaku iteraciju pomoću sistemskog vremena meri vreme izvršavanja.

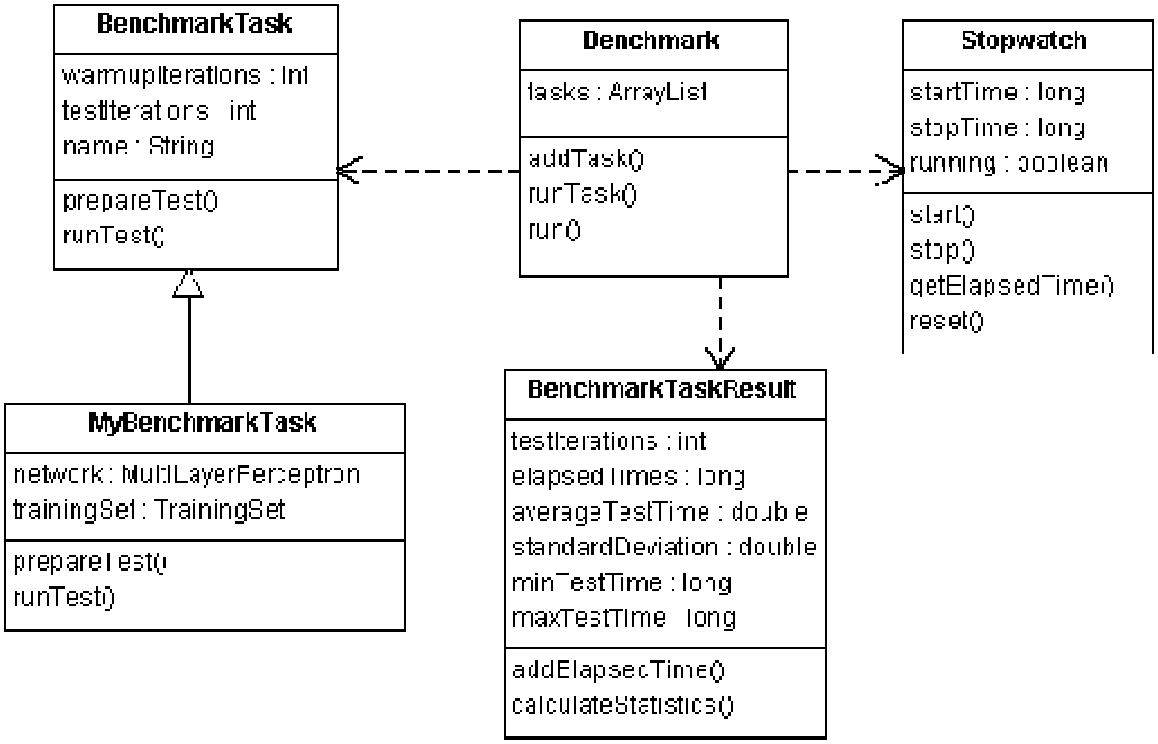
Generisanje statistika i izveštavanje

Na kraju testa, kada se izvrše sve iteracije testiranja, generiše se jednostavna statistika: min, max, srednja vrednost i standardna devijacija.



Slika 23. Dijagram toka procesa testiranja performansi pomoću *micro-benchmark* testova

Na slici 24. Dat je dijagram klasa koje realizuju micro-benchmark za Neuroph prema defnisanom dijagramu toka.



Slika 24. Dijagram klasa koje relizuju micro-benchmark za Neuroph

Klasa *Stopwatch* predstavlja štopericu kojom se meri sistemsko vreme tokom izvršavanja testa performansi.

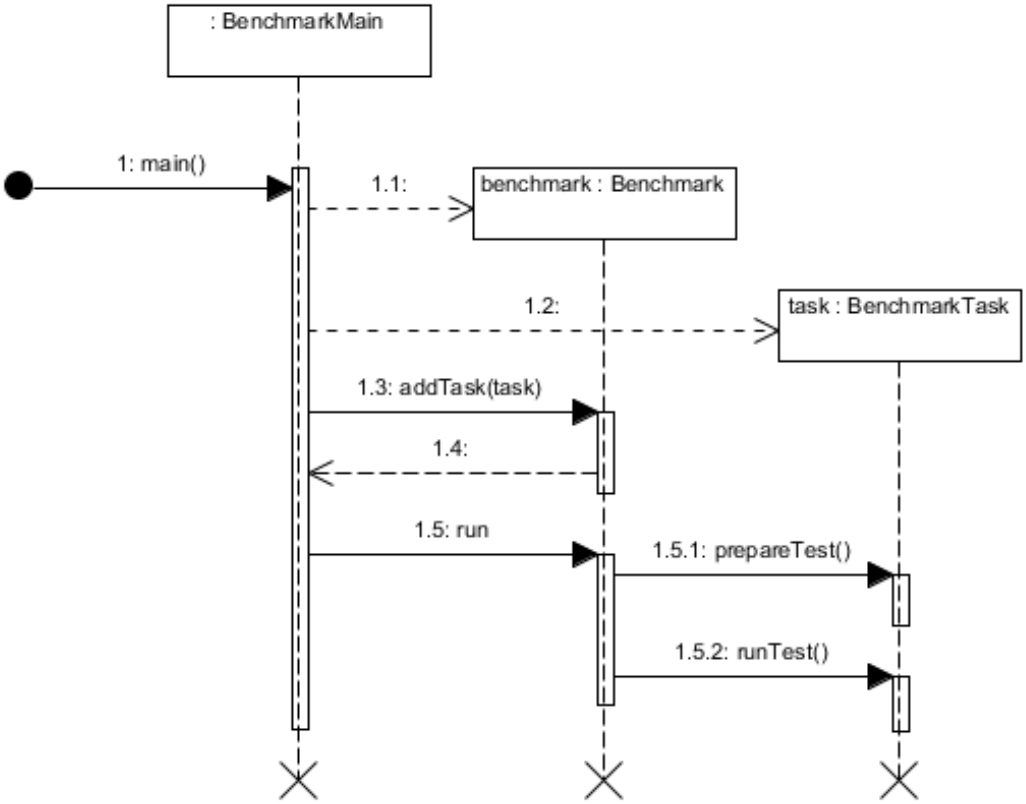
Klasa *BenchmarkTask* je apstraktna osnovna klasa za sve testove performansi. Ona sadrži podešavanja za broj iteracija za fazu zagrevanja i testiranja, a očekuj eod izvedenih klasa da definišu apstraktne metode *prepareTest* i *runTest* koje treba da implementiraju logiku za pripremu i izvršavanje testa. Klasa MyBenchmarClass predstavlja konkretnu jednu implementaciju testa performansi.

Klasa *Benchmark* sadrži statičke metode koje sadrže logiku toka izvršavanja testa za jedan ili više testova – metode *runTask* i *runTasks.*

Klasa BenchmarkTaskResults sadrži izmerena vremena prilikom testiranja performansi, i obezbeđuje metode za generisanje statistike i izveštavanje.

Kreiranje novih, korisnički definsanih testova, predviđeno je nasleđivanjem klase BenchmarkTask, i definisanjem apstraktnih metoda. Pored toga nisu potrebne nikakve dodatne izmene, i ostale klase za testiranje se koriste bez izmena, Na taj način, obezbeđen je fleksibilni proširiv model za testiranje.

Na slici 25 dat je dijagram sekvenci prilikom testiranja performansi.



Slika 25. Dijagram sekvenci za testiranja performansi

5. IMPLEMENTACIJA

U ovom poglavlju dat je programski kod i opisani su detalji implementacije najvažnijih klasa koje su isprojektovane u četvrtom poglavlju. Uz svaki listing programskog koda, ukazano je na najbitnije elemente implementacije, razlog za primenu određenih rešenja i ukazano je na saglasnost sa principima razvoja softverskih *framework*-a definisanih u 4. poglavlju. Uz svaku novu funkcionalnost, dat je i kratak primer koji demonstrira način korišćenja. Implementirane su sledeće funkcionalnosti:

1. sinusna i logaritamska funkcija transfera;
2. tehnike normalizacije podataka Max, MaxMin, i decimalno skaliranje:
3. tehnike za generisanje slučajnih težinskih koeficijenata Range, Distort, Gaussian, NguyenWidrow;
4. ulazni adapteri za fajl, URL, JDBC, stream;
5. izlazni adapteri za fajl, URL, JDBC, stream;
6. nove vrste neuronskih mreža i algoritama za učenje ResilientPropagation, PCANetwork,GeneralizedHebbianLearning;
7. merenje performansi *(benchmark*).

5.1. Implementacija novih funkcija transfera

Na listingu 1 dat je programski kod klase *Sin* koja predstavlja nove vrste funkcija transfera za sinusnu funkciju. U skladu sa rešenjem projektovanim u odeljku 4.5.1 ova klasa:

1. nasleđuje apstraktnu klasu TransferFunction;
2. redefiniše odgovarajuće metode za izračunavanje vrednosti, i prvog izvoda ovih funkcija.

```
public class Sin extends TransferFunction {

    @Override
    public double getOutput(double net) {
        return Math.sin(net);
    }

    @Override
    public double getDerivative(double net) {
      return Math.cos(net);
    }
}
```

Listing 1. Programski kod funkcije transfera Sin

Na listingu 2, dat je primer korišćenja ove klase, prilikom kreiranja instance neurona sa sinusnom funkcijom transfera.

```
// kreiranje neurona sa sinusnom funkcijom transfera
Neuron neuron =
 new Neuron(new WeightedInput(), new Sin());
```

Listing 2. Primer korišćenja: kreiranje neurona sa sinusnom funkcijom transfera

U listing 3.1, u prilogu 3, dat je programski kod klase *Log* koja realizuje logaritamski funkciju transfera po istom principu.

5.2. Implementacija normalizacije podataka za trening

Na listingu 3 dat je programski kod interfeja *Normalizer* koji predstavlja interfejs za razne raspoložive vrste normalizacije, a takođe predstavlja i osnovu za dodavanje novih vrsta normalizacije po potrebi. Interfejs definiše metodu *normalize*, koja kao parametar prihvata skup podataka za trening, i predstavlja javni programski interfejs (API) za normalizaciju.

```
public interface Normalizer {
    public void normalize(TrainingSet trainingSet);
}
```

Listing 3. Interfejs za normalizaciju podataka

Na listingu 3.2. u prilogu 3 dat je programski kod klase *MaxNormalizer* koja predstavlja implementaciju interfejsa *Normalizer* za algoritam normalizacije u odnosu na maksimalnu vrednost u skupu.
Na listingu 4 dat je primer normalizacje skupa podataka za trening. Iz listinga se vidi da se sama normalizacija se vrši jednim pozivom metode, a na isti način se koristi i u situaciji kada se kao parametar prosleđuje neka nova klasa za normalizaciju definisana od strane korisnika.

```
// podrazumevana (max) normalizacija
trainingSet.normalize();

// eksplicitna max normalizacija
trainingSet.normalize(new MaxNormalizer());

// max-min normalizacija
trainingSet.normalize(new MaxMinNormalizer();
```

Listing 4. Primer koda za normalizaciju skupa podataka za trening

Pored max normalizacije, u okviru Neuroph framework-a takođe su implementirane max-min normaizacija i decimalno skaliranje.

5.3. Implementacija tehnika za generisanje slučajnih brojeva

Na listingu 5 dat je programski kod klase *WeightsRandomizer* koja predstavlja osnovnu klasu koju nasleđuju sve tehnike za generisanje slučajnih težinskih koeficijenata. Klasa sadrži atribute koji predstavljaju neuronsku mrežu (*neuralNetwork*) i standardni Java generator slučajnih brojeva (*randomGenerator*). Glavne metode u ovoj klasi su:

- *nextRandomWeight*, koja nema ulaznih parametara, a vraća sledeću slučajnu vrednost težinskog koeficijenta. U ovoj osnovnoj klasi ova metoda koristi java random, metoda se preklapa
- *randomize*, koja iterativno prolazi kroz sve slojeve neurona, i veza između neurona u neuronskoj mreži zadatoj kao ulazni parametar i postavlja vrednosti težina koristeći metodu *nextRandomWeight*.

```
public class WeightsRandomizer {
    protected NeuralNetwork neuralNetwork;
    protected Random randomGenerator;

    public WeightsRandomizer() {
        this.randomGenerator = new Random();
    }

    public Random getRandomGenerator() {
        return randomGenerator;
    }

    public void randomize(NeuralNetwork neuralNetwork) {
        this.neuralNetwork = neuralNetwork;
        for (Layer layer : neuralNetwork.getLayers()) {
            for (Neuron neuron : layer.getNeurons()) {
                for (Connection connection :
                            neuron.getInputConnections()) {
        connection.getWeight().setValue(nextRandomWeight());
                }
            }
        }
    }

    protected double nextRandomWeight() {
        return randomGenerator.nextDouble();
    }
}
```

Listing 5. Osnovna klasa za generisanje slučajnih vrednosti težinskih koeficijenata

Ova klasa implementira osnovni mehanizam inicijalizacije težina pomoću standardnog Java generatora slučajnih brojeva, a druge tehnike inicijalizacije se jednostavno mogu implementirati na sledeći način:

1) nasleđivanjem ove klase;
2) redefinisanjem metode *nextRandomWeight*.

Najvažniji elementi dizajna ove klase su:

- javni interfejs koji se sastoji od metode *randomize;*
- tačka za dalja proširenja i modifikacije koju predstavlja metoda *nextRandomWeight;*
- Java generator slučajnih brojeva sa protected vidljivošću, tako da stoji na raspolaganju i izvedenim klasama.

Na listingu 3.3. u prilogu 3., dat je programski kod klase *RangeRandomizer* koja generiše slučajne vrednosti za inicijalizaciju težina, u okviru zadatog intervala [min, max]. Ova klasa nasleđuje klasu *WeightsRandomizer* i redefiniše metodu *nextRandomWeight* koja neposredno generiše slučajne vrednosti. Slučajne vrednosti u okviru određenog intervala min, max se generišu pomoću jednostavne formule i generatora slučajnih vrednosti nasleđenog iz osnovne klase, a interval se zadaje u konstruktoru prilikom kreiranje instanci ove klase. Sama procedura dodele generisanih slučajnih vrednosti se takođe nalazi u osnovnoj klasi u metodi *randomize*.

Ova klasa je odličan primer kako se osnova rešenja za generisanje slučajnih težinskih koeficijenata vrlo jednostavno može proširiti, uz veliko iskorišćenje postojećeg koda iz osnovnih (nasleđenih) klasa.

Na listing 3.4. u prilogu3., dat je programksi kod klase DistortRandomizer koja modifikuje postojeće težinske koeficijente prema slučajnom faktoru. Klasa *DistortRandomizer* takođe nasleđuje klasu *WeightsRandomizer,* ali za razliku od klase *RangeRandomizer* koja redefiniše metodu *nextRandomWeight*, *DistortRandomizer* redefiniše metodu *randomize*, i dodaje jednu novu metodu – *distortWeight*. Ovakvo rešenje je neophodno, jer je za generisanje nove vrednosti neke težine, potrebno znati i njenu trenutnu vrednost, i za to se koristi metoda *distortWeight,* koja trenutnu vrednost težine dobija kao ulazni parametar. Iz prethodna dva primera se vidi fleksibilnost projektovanog rešenja, odnosno mogućnost prilagođavanja osnovne klase u različitim scenarijima. Ovakva flekcibilnost je od izuzetnog značaja za dalju evoluciju API-a, i dodavanje novih metoda normalizacije.

Na listingu 3.5. u prilogu 3. dat je programski kod klase *NguyenWidrowRandomizer* koja generiše slučajne vrednosti težina *Nguyen-Widrow* metodom. Ova klasa nasleđuje klasu *RangeRandomizer* (jer generiše slučajne vrednosti o okviru zadatog intervala), i redefiniše metodu *randomize*.

Na listingu 6 dat je programski kod koji demonstrira korišćenje klasa za generisanje slučajnih težinskih koeficijenata u neuronskoj mreži,

```
MultiLayerPerceptron nnet = new MultiLayerPerceptron(2, 3, 1);
nnet.randomizeWeights(new NguyenWidrowRandomizer(0.3,0.7));
// nnet.randomizeWeights(new RangeRandomizer(0.1, 0.9));
// nnet.randomizeWeights(new GaussianRandomizer(0.4, 0.3));
```

Listing 6. Primer programskog koda za generisanje slučajnih težinskih koeficijenata za neuronsku mrežu

5.4. Ulazni adapteri

Na listingu 7 dat je programski kod interfejsa za ulazne adaptere u kladu sa rešenjem projektovanim u poglavlju 4.5.4. Interfejs ima dve metode:

1) `readInput` za čitanje sledećeg ulaza za neuronsku mrežu;
2) `close` za zatvaranje izvora.

Metoda koja vrši čitanje `readInput` treba da vraća niz *double* vrednosti koje predstavljaju ulaz za neuronsku mrežu. Metoda `close` treba da zatvori ulazni izvor, odnosno oslobodi zauzete resurse nakon što su pročitani svi podaci.

```
public interface InputAdapter {
    public double[] readInput();
    public void close();
}
```

Listing 7. Interfejs za ulazne adaptere

Na listingu 3.6, u prilogu 3., dat je programski kod klase *InputStreamAdapter* koja predstavlja implementaciju interfejsa *InputAdapter* za rad sa ulaznim *stream-ovima*. Ova klasa predtsvlja osnovnu klasu za čitanje iz ulaznih stream-ova, i koristi *BufferedReader* objekat za baferovano čitanje iz karakter *stream-ova*. Ima dva konstruktora, koji kao ulazne parametre prihvataju ulazni *stream-a* (*InputStream*) ili objekat klase *BufferedReader*.
Metode za čitanje i zatvaranje stream-a u potpunosti obavljaju svoju funkciju čitanja iz karakter stream-ova i čak ih nije potrebno redefinisati u izvedenim klasama koje u osnovi koriste stream. Konstruktor prihvata parametar tipa *BufferedReader* služi za inicijalizaciju iz izvedenih klasa.

Na listingu 8 dat je programski kod klasa *FileInputAdapter* koja nasleđuju klasu *InputStreamAdapter* i obezbeđuju podršku za pisanje u fajlove. Ova klasa nasleđuje bez izmena metode za čitanje i zatvaranje stream-a, i samo pozivaju konstruktor osnovne klase kojoj prosleđuju odgovarajući *BufferedReader* objekat. Time je postignuto maksimalno iskorišćenje postojećeg koda za čitanje iz osnovne klase, i veoma jednostavna implementacija čitanja iz fajla, koje se zapravo svodi na čitanje iz ulaznog stream-a već obezbeđenog u osnovnoj klasi.

```
  public class FileInputAdapter extends InputStreamAdapter {

    public FileInputAdapter(File file) throws
FileNotFoundException {
        super(new BufferedReader(new FileReader(file)));
    }

    public FileInputAdapter(String fileName) throws
FileNotFoundException {
        this(new File(fileName));
    }
}
```

Listing 8. Klasa *FileInputAdapter* koja implementira ulazni adapter za čitanje iz fajla

Na listingu 3.7. u prilogu 3. dat je programski kod klase *URLInputAdapter* koja takođe nasleđuje klasu *InputStreamAdapter*,i obezbeđuje čitanje iz stream-a otvorenog nad zadatim URL-om, po istom principu kao i *FileInputAdapter*.

Na listingu 9 dat je primer korišćenja ulaznog adaptera za čitanje iz fajla.

```
MultiLayerPerceptron neuralNet = new MultiLayerPerceptron(2, 3,
1);
FileInputAdapter fileIn = new
FileInputAdapter("net_input.txt");
double[] input;
while( (input = fileIn.readInput()) != null) {
     neuralNet.setInput(input);
     neuralNet.calculate();
     double[] output = neuralNet.getOutput();
}
fileIn.close();
```

Listing 9. Primer korišćenja ulaznog adaptera za čitanje ulaza za neuronsku mrežu iz fajla

Na listingu 3.8. u prilogu 3., dat je programski kod klase *JDBCInputAdapter* koja koja implementira ulazni adapter za čitanje ulaza za neuronsku mrežu iz baze podataka. Konstruktor klase prihvata izvor podataka, tj. konekciju ka bazi podataka (objekat klase *java.sql.Connection*) i SQL upit u vidu *String-a* koji predstavlja upit koji daje ulazne podatke. Za razliku od prethodnih adaptera, ovaj adapter ne koristi *stream*, i definiše sopstvene metode za čitanje i zatvaranje izvora podataka (u skladu sa specifikacijom interfejsa) tako da rade sa bazom podataka.

5.5. Izlazni adapteri

Na listingu 10 dat je programski kod interfejsa za izlazne adaptere u kladu sa rešenjem projektovanim u odeljku 4.5.5. Interfejs ima dve metode:

1) `writeOutput` za upis izlaza iz neuronske mreže (double niz);
2) `close` za zatvaranje upisa.

Metoda `writeOutput` koja vrši upisivanje prihvata ulazni parametar niz *double* vrednosti koje predstavljaju izlaz iz neuronske mreže. Metoda `close` treba da zatvori izlaz i oslobodi zauzete resurse nakon što su upisani svi podaci.

```
public interface OutputAdapter {
    public void writeOutput(double[] output);
    public void close();
}
```

Listing 10. Interfejs za izlazne adaptere

Na listingu 3.9. u prilogu 3, dat je programski kod klase *OuputStreamAdapter* koja predstavlja implementaciju interfejsa *OutputAdapter* za rad sa izlaznim *stream-ovima*. Ova klasa predtsvlja osnovnu klasu za upis u izlazne stream-ove, i koristi *BufferedWriter* objekat za baferovan upis u karakter *stream-ove*. Ima dva konstruktora, koji kao ulazne parametre prihvataju ulazni *stream-a* (*OutputStream*) ili objekat klase *BufferedWriter*. Klasa *OutputStreamAdpter* implementira metode navedene u interfejsu OutputAdapter i ove metode koriste i sve izvedene klase (koje su takođe izlazni adapteri) koje rade sa izlaznim stream-ovima.
Na listingu 11 dat je programski kod klase *FileOutputAdapter* koja obezbeđuje upisivanje u fajlove. Ova klasa bez izmena nasleđuju metode za čitanje i zatvaranje stream-a, i samo poziva konstruktor osnovne klase kome prosleđuju odgovarajući *BufferedWriter* objekat. Time je postignuto maksimalno iskorišćenje postojećeg koda za upisivanje iz osnovne klase, i veoma jednostavna implementacija pisanja u fajl, koja se zapravo svodi na pisanje u izlazni *stream* u osnovnoj klasi.

```
public class FileOutputAdapter extends OutputStreamAdapter {

    public FileOutputAdapter(File file) throws
FileNotFoundException, IOException {
        super(new BufferedWriter(new FileWriter(file)));
    }
    public FileOutputAdapter(String fileName) throws
FileNotFoundException, IOException {
        this(new File(fileName));
    }}
```

Listing 11. Klasa *FileOutputAdapter* koja implementira adapter za upisivanje u fajl

Na listingu 3.10. u prilogu 3, dat je programski kod klase *URLOutputAdapter* koja takođe nasleđuje klasu *OuputStreamAdapter*,i obezbeđuje pisanje u stream-a otvorenog nad zadatim URL-om, po istom principu kao i *FileOutputAdapter*.

Na listingu 12 dat je primer korišćenja ulaznog adaptera za čitanje iz fajla.

```
FileOutputAdapter fileOut = new
FileOutputAdapter("net_output.txt");         double[] output =
neuralNetwork.getOutput();
fileOut.writeOutput(output);
fileOut.close();
```

Listing 12. Primer korišćenja adaptera za upisivanje izlaza neuronske mreže u fajl

Na listingu 3.11. u prilogu 3, dat je programski kod klase *JDBCOutputAdapter* koja koja implementira izlazni adapter za upis izlaza neuronske mreže u bazu podataka. Konstruktor klase prihvata konekciju ka bazi podataka (objekat klase *java.sql.Connection*) i naziv tabele u koju treba upisati podatke. Za razliku od prethodnih izlaznih adaptera, ovaj adapter ne koristi *stream*, i definiše sopstvene metode za čitanje i zatvaranje izvora podataka (u skladu sa specifikacijom interfejsa) tako da rade sa bazom podataka.

5.6. Algoritmi za učenje neuronskih mreža

U ovom odeljku dati su neki detalji implementacije klasa *ResilientPropagation*, *PCANetwork* i *GeneralisedHebianLearning* pomoću kojih su relizovane nove vrste algoritama za učenje i neuronskih mreža.
Na listingu 13. dat je deo implementacije klase ResilientPropagation, zbog dužine listing kompletan kod nije dostupan u radu, ali se može preuzeti sa Interneta u okviru izvornog koda projekta.

```
public class ResilientPropagation extends BackPropagation {
    private double decreaseFactor = 0.5;
    private double increaseFactor = 1.2;
    private double initialDelta = 0.1;
    private double maxDelta = 1;
    private double minDelta = 1e-6;
    private static final double ZERO_TOLERANCE = 1e-27;

    public ResilientPropagation() {
        super();
        super.setBatchMode(true);
    }

    /**
     * Calculate and sum gradients for each neuron's weight,
the actual weight update is done in batch mode
     * @see resillientWeightUpdate
     */
    @Override
    protected void updateNeuron(Neuron neuron) {
        for (Connection connection :
neuron.getInputConnections()) {
            double input = connection.getInput();
            if (input == 0) {
                continue;
            }
            // get the error for specified neuron,
            double neuronError = neuron.getError();
            // get the current connection's weight
            Weight weight = connection.getWeight();
            // ... and get the object that stores reislient
training data for that weight
            ResilientWeightTrainingtData weightData =
(ResilientWeightTrainingtData) weight.getTrainingData();
            // calculate the weight gradient (and sum gradients
since learning is done in batch mode)
            weightData.gradient += neuronError * input;
        }
    }
```

```
    public class ResilientWeightTrainingtData {
        public double gradient;
        public double previousGradient;
        public double previousWeightChange;
        public double previousDelta = initialDelta;
    }
}
```

Listing 13. Klasa ResilientPropagation koja implementira istoimeni algoritam za učenje

Nekoliko najzanimljivijih detalja u vezi sa implementacijom ovog algoritma su:

- uvođenje interne klase `ResilientWeightTrainingtData` koja sadrži sve vrednosti koje ovaj specifični algoritam koristi za izračunavanje promena težina, i objekti te klase se pre početka treninga ubacuju u sve Weights objekte neuronske mreže (tzv. *dependency injection patern*)
- algoritam se uvek izvršava u batch modu – u konstruktoru ima `super.setBatchMode(true);`
- redefiniše metodu `updateNeuron` koju nasleđuje još iz klase LMS, i u kojoj samo računa sumu gradijentata za sve ulazne veze odnosno težine neurona. Stvarni izračunavanje promena tećine vrši se u batch modu u metodi *resillientWeightUpdate.*

Na listinzima 14 i 15 dat je programski kod klasa PCANetwork i GeneralizedHebbianLearning koja realizuju mrežu tipa PCA sa odgovarajućim algoritmom za učenje.

```
public class PCANetwork extends NeuralNetwork {

    public PCANetwork(int inputNeuronsCount, int
outputNeuronsCount) {
        this.createNetwork(inputNeuronsCount,
outputNeuronsCount);
    }

     private void createNetwork(int inputNeuronsCount, int
outputNeuronsCount) {
           // set network type
           this.setNetworkType(NeuralNetworkType.PCA_NETWORK);
           // init neuron settings for input layer
           NeuronProperties inputNeuronProperties = new
NeuronProperties();
           inputNeuronProperties.setProperty("transferFunction",
TransferFunctionType.LINEAR);
           // create input layer
           Layer inputLayer =
LayerFactory.createLayer(inputNeuronsCount,
inputNeuronProperties);
```

```
            this.addLayer(inputLayer);
            NeuronProperties outputNeuronProperties = new
NeuronProperties(Neuron.class, WeightedSum.class,
Linear.class);
            // createLayer output layer
            Layer outputLayer =
LayerFactory.createLayer(outputNeuronsCount,
outputNeuronProperties);
            this.addLayer(outputLayer);
            // create full conectivity between input and output
layer
            ConnectionFactory.fullConnect(inputLayer,
outputLayer);
            // set input and output cells for this network
            NeuralNetworkFactory.setDefaultIO(this);

             this.setLearningRule(new
GeneralizedHebbianLearning());
}}
```

Listing 14. Klasa PCANetwork koja implementira istoimenu vrstu neuronskih mreža

Ove dve klase su tipičan primer kako se u okviru Neuroph framework-a grade nove vrste neuronskih mreža i algoritama za učenje na predviđenim tačkama proširenja, uz veliko iskorišćenje postojeće strukture i logike. Princip proširenja podrazumeva nasleđivanje odgovarajućih postojećih osnovnih klasa uz dodavanje specifičnosti nove mreže odnosno algoritma za učenje.
Tako klasa nove neuronske mreže obično definiše metodu `createNetwork` koja kreira odgovarajuću arhitekturu mreže (slojeve i neurone), dok se u okviru algoritma za učenje redefinišu metode koje vrše podešavanje težinskih koeficijenata – `updateNeuronWeights.`

```
public class GeneralizedHebbianLearning extends
UnsupervisedHebbianLearning {
    @Override
    protected void updateNeuronWeights(Neuron neuron) {
        double output = neuron.getOutput();
        for(Connection connection :
neuron.getInputConnections()) {
            double input = connection.getInput();
                        double netInput = neuron.getNetInput();
            double weight = connection.getWeight().getValue();
             double deltaWeight = (input - netInput) * output *
                                  this.learningRate;
             connection.getWeight().inc(deltaWeight);
     }}}
```

Listing 15. Klasa GeneralizedHebbianLearning koja impelmentira jednu varijantu algoritma *Hebbian Learning*

Listing klase GeneralizedHebbianLearning pokazuje koliko je u okviru Neuroph framework-a jednostavno implementirati varijacije nekog algoritma jednostavnim nasleđivanjem i redefinisanjem metode za izračunavanje promena težina - `updateNeuronWeights`.

5.7. Testiranje performansi – benchmark

Testiranje performansi realizovano je pomoću sledećih klasa:

org.neuroph.benchmark.Stopwatch
org.neuroph.benchmark.*BenchmarkTask*
org.neuroph.benchmark.*Benchmark*
org.neuroph.benchmark.*BenchmarkTaskResults*

Klasa *Stopwatch* predstavlja 'štopericu', koja se koristi se za merenje vremena tokom testa performansi. Ima metode *start* i *stop* koje uzimaju trenutno sistemsko vreme u milisekundama pozivom *System.currentTimeMillis(),* i markiraju vreme početka i završetka izvršanja programa koji se testira. Metoda *getElapsedTime* vraća proteklo vreme između poziva start i stop metoda, u milisekundama. Na listingu 3.12. u prilogu 3, dat je kompletan programski kod klase Stopwatch

Klasa *BenchmarkTask* koja predstavlja apstraktnu osnovnu klasu za testove performansi koje kreiraju korisnici. Apstraktne metode *prepareTest* i *runTest* u izvedenim klasama treba da implementiraju pripremu testa, i samo izvršavanje testa. Metoda *prepareTest* treba da pripremi podatke i inicijalizuje sve potrebne objekte, dok metoda *runTest* sadrži segment programskog koda kome se testiraju performanse. Klasa ima atribute *warmupIterations* i *testIterations* koji predstavljaju podešavanje, koliko iteracija treba da se izvrši u fazi 'zagrevanja', i testiranja. Na listingu 3.13. u prilogu 3, dat je kompletan programski kod klase *BenchmarkTask.*

Klasa *Benchmark* realizuje osnovni tok procesa izvršavanje testa i merenje vremena. Glavna metoda u klasi je *runTask* koja predstavlja implementaciju toka izvršavanja testa, Ova metoda kao parametar prihvata zadatak (BenchmarkTask) koji se testira, i zatim:

1. izvršava pripremu testa;
2. izvršava zagrevanje testa tako što izvrši metodu *runTest* zadati broj iteracija (*warmupIterations*)
3. izvršava testiranje performansi, odnosno merenje vremena izvršavanja zadati broj iteracija (*testIterations*)

Klasa takođe može sadrži i kolekciju testova performansi (više objekata klase *BenchmarkTask*) i da ih redom izvršava jedan po jedan. U tom slučaju testovi se dodaju pomoću metode *add()* a izvršavanje svig testova se pokreće pomoću metode *run()*. Na listingu 3.14 u prilogu 3, dat je kompletan programski kod klase *Benchmark.*

Na listingu 16, dat je programski kod klase *MyBenchmarkTask* koja predstavlja korisnički definisan test performansi za neuronski mrežu iz Neuroph-a, Klasa *MyBenchmarkTask* nasleđuje klasu *BenchmarkTask* i implementira apstraktne metode *prepareTest i runTest.*
Metoda prepareTest kreira slučajno generisani skup podataka za trening, i neuronski mrežu tipa *MultiLayerPerceptron* koja se testira. Metoda *runTest* u startuje trening neuronske mreže.

```
public class MyBenchmarkTask extends BenchmarkTask {
    private MultiLayerPerceptron network;
    private TrainingSet trainingSet;

    public MyBenchmarkTask(String name) {
        super(name);
    }

    public void prepareTest() {
        int trainingSetSize = 100;
        int inputSize = 10;
        int outputSize = 5;

        this.trainingSet = new TrainingSet(inputSize,
outputSize);

        for (int i = 0; i < trainingSetSize; i++) {
            double input[] = new double[inputSize];
            for( int j=0; j<inputSize; j++)
                input[j] = Math.random();

            double output[] = new double[outputSize];
            for( int j=0; j<outputSize; j++)
                output[j] = Math.random();

            SupervisedTrainingElement element =
                                new
SupervisedTrainingElement(input, output);
            trainingSet.addElement(element);
        }

        network = new
MultiLayerPerceptron(inputSize,8,7,outputSize);
    ((MomentumBackpropagation)network.getLearningRule())
.setMaxIterations(11000);
    }

  public void runTest() {
        network.learnInSameThread(trainingSet);
    }
}
```

Listing 16. Primer implementacije testa performansi za Neuroph neuronske mreže

Na listingu 17. dat je primer testiranja performansi pomoću testa *MyBenchmarkTask*. Iz primera se vidi da je programski interfejs vrlo jednostavan i intuitivan.
Prvo je potrebno kreirati instancu testa performansi koji treba izvršiti, i podesiti parametre za broj iteracija za zagrevanje i testiranje. Nakon toga kreira se instanca klase *Benchmark* koja obezbeđuje logiku za izvršavanje testova, dodaje se test jednostavnim pozivom metode *addTask*, i na kraju startuje izvršavanje testa metodom *run*.

```
public class BenchmarkMain {
    public static void main(String[] args) {
        BenchmarkTask task1 = new
MyBenchmarkTask("MyFirstBenchmark");
        task1.setWarmupIterations(2);
        task1.setTestIterations(5);

        Benchmark benchmark = new Benchmark();
        benchmark.addTask(task1);

        benchmark.run();
    }
}
```

Listing 17. Primer korišćenja izlaznog adaptera za upisivanje izlaza neuronske mreže u fajl

6. PRIMERI KORIŠĆENJA

U ovom poglavlju dati su primeri koji opisuju način rešavanja problema pomoću neuronskih mreža i Neuroph framework-a. Dati primeri predstavljaju osnovno korisničko uputstvo, koje je neizostavni deo svakog softvera. Glavna namena takvog kratkog korisničkog uputstva je da kroz konkretne realne primere u osnovnim crtama upozna korisnika sa softverom, i omogući mu da što pre počne da ga primenjuje.

6.1. Način rešavanja problema pomoću neuronskih mreža i Neuroph framework-a

Dugoročni cilj Neuroph frameworka je da podrži sve faze u procesu rešavanja problema pomoću neuronskih mreža. Opšta procedura rešavanja problema pomoću neuronskih mreža obuhvata sledeće faze

1. priprema podataka za trening;
2. trening neuronskih mreža;
3. testiranje neuronskih mreža;
4. primena istreniranih neuronskih mreža.

Priprema podataka za trening podrazumeva obradu podataka koji opisuju problem koji se rešava i koji će se koristiti za trening neuronske mreže. Priprema podataka se vrši kako bi se ostvarili bolji rezulti prilikom treninga neuronsih mreža i obuhvata razne metode normalizacije, redukcije dimenzija i filtriranja podataka. Neuroph framework trenutno obezbeđuje nekoliko metoda za normalizaciju koje su implementirane u okviru ovog istraživanja, i PCA metodu za redukciju dimenzionalnosti.

Trening neuronskih mreža podrazumeva primenu algoritma za trening neuronskih mreža sa pripremljenim podacima za trening, tokom koga se vrši automatsko podešavanje parametara neuronske mreže kako bi dobila željenu funkcionalnost. Svrha treninga je pronalaženje optimalnih parametara neuronske mreže koja daje zadovoljavajuće rešenje, i podrazumeva isprobavanje raznih kombinacija parametara. Neuroph obezbeđuje softversku alatku sa grafičkim korisničkim interfejsom NeurophStudio koja omogućava eksperimentisanje prilikom treninga neuronskih mreža, kao i Java API koji omogućava programiranje specifičnih procedura treninga.

Testiranje neuronskih mreža podrazumeva proveru funkcionalnosti neuronske mreže sa podacima koji nisu bili korišćeni prilikom treninga. Trening i testiranje se obično vrše sa podskupovima celokupnog skup podataka za trening. Npr. 70% skupa podataka za trening se koristi baš za trening, a preostalih 30% za testiranje. Ovaj odnos može biti i drugačiji u zavisnosti od konkretnog problema, pri čemu se najbolji odnos utvrđuje eksperimentalnim putem. Neuroph omogućava jednostavno kreiranje skupova podataka za trening i testiranje pomoću klase *TrainingSet*.

Primena istreniranih neuronskih mreža podrazumeva kreiranje softverskih komponenti koje implementiraju istreniranu neuronsku mrežu i omogućavaju njeno korišćenje u aplikacijama od interesa. Neuroph omogućava čuvanje istreniranih neuronskih mreža u vidu serijalizovanih Java objekata, i njihovo korišćenje u drugim aplikacijama. Da bi neka aplikacija koristila neuronsku mrežu kreiranu sa Neuroph-om, dovoljno je da ima referencu na jar fajl Neuroph framework-a.

Neuroph se za rešavanje problema pomoću neuronskih mreža može koristiti na tri načina:
1. korišćenjem razvojnog okruženja za neuronske mreže;
2. korišćenjem specijalizovanih *wizarda* za određenu vrstu problema;
3. korišćenjem Neuroph Java API-a, direktno u programskom kodu.

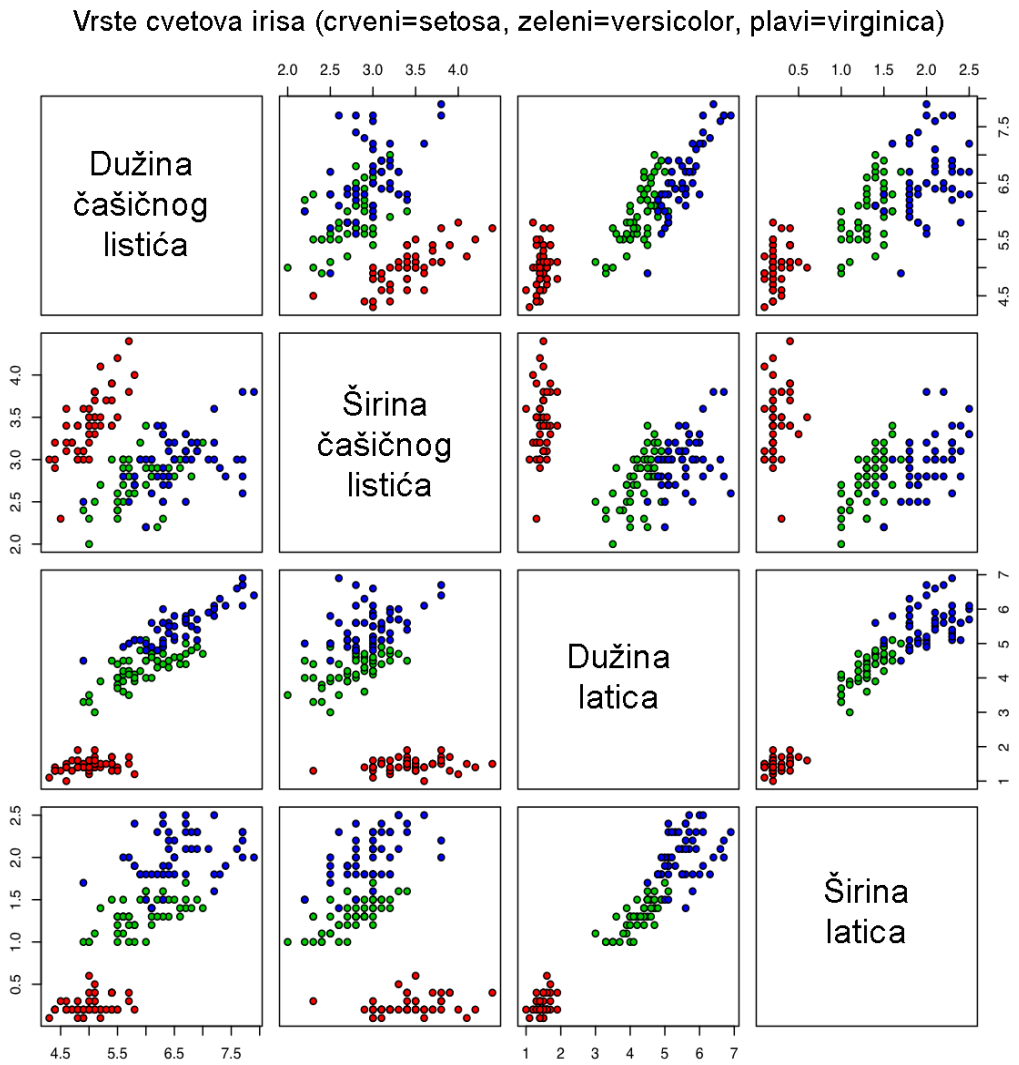
Primeri dati u daljem tekstu, neposredno prikazuju načine pod 1 i 2, koji pri tom ispod korisničkog interfejsa koriste API Neuroph framework-a.

6.2. Klasifikacija cvetova Iris-a

Opis primera: Primer problema klasifikacije cvetova iris-a demonstrira primenu razvojnog okruženja NeurophStudio zasnovanom na Neuroph framework-u, za rešavanje problema klasifikacije.

Opis problema: Problem klasifikacije cvetova irisa je dobro poznat problem u oblasti klasifikacije i predstavlja često korišćeni primer prilikom testiranja i poređenja različitih algoritama za klasifikaciju. Problem se sastoji u tome da se na osnovu četiri karakteristike cvetova (dužine i širine latica, i dužine i širine čašičnog listića) odredi kojoj od 3 vrste irisa neki cvet pripada. Suština problema je u tome što je jedna klasa linerano separabilna od druge dve, dok te druge dve međusobno nisu linearno separabilne, što se vidi na slici 26.

Podaci: Skup podataka za trening neuronske mreže sadrži podatke o 3 vrste irisa, po 50 uzoraka za svaku vrstu, odnosno ukupno 150 uzoraka. Za svaki uzorak dati su sledeći podaci:
1. dužina čašičnog listića u cm
2. širina čaštičnog listića u cm
3. dužina latice u cm
4. širina latice u cm
5. klasa irisa kojoj cvet priprada: setosa, versicolour, virginica



Slika 26. Klasifikacija cvetova irisa

Planiranje treninga

Planiranje treninga neuronskih mreža podrazumeva planiranje procedure pripreme podataka, treninga i testiranja. Za rešavanje problema klasifikacije cvetova irisa planirano je kreiranje tri para trening i test skupova podataka za trening na osnovu raspoloživog uzorka i to na način prikazan u tabeli 15:

Tabela 15. Skupovi podataka za trening i testiranje

Rb.	Skup za trening	Skup za testiranje
1.	70% uzorka	30%
2.	80%	20%
3.	90%	10%

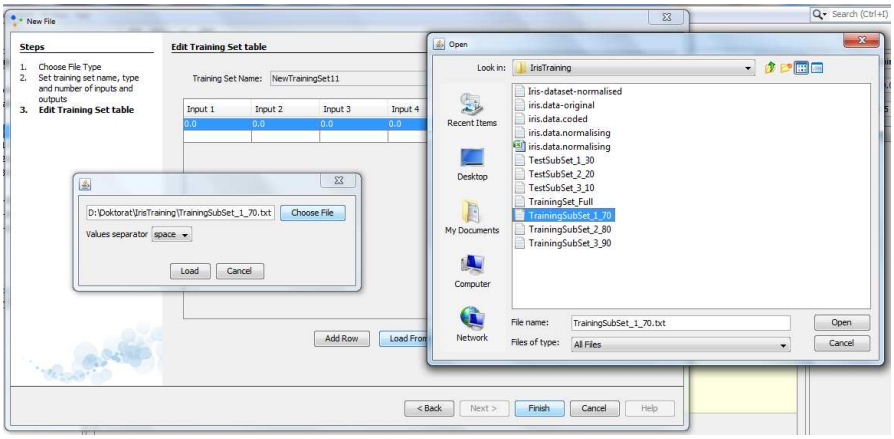
Za rešavanje problema koristi se neuronska mreža tipa višeslojni perceptron (Multi Layer Perceptron) i algoritam za učenje Backpropagation. Ovaj tip neuronske mreže se uspešno primenjuje za razne probleme klasifikacije.
Trening izvršiti sa tri različita skupa parametara za algoritma za trening i neuronske mreže kako bi se sagledali dobijeni rezultati sa različitim parametrima.

Priprema podataka

Priprema podataka u ovom primeru podrazumeva normalizaciju podataka iz uzorka, i kreiranje skupova podataka za trening i testiranje u skladu sa projektovanim treningom, odnosno tri para trening i test podataka i to u odnosu (70, 30), (80, 20) i (90, 10).
Podaci iz uzorka su normalizovani metodom normalizacije u odnosu na maksimalni i minimalni element. Priprema podataka je obavljena pomoću klase API-ja Neuroph framework-a, korišćenjem metoda `normalize` i `createTrainingAndTestSubset` iz klase `TrainingSet` na sledeći način:

```
TrainingSet trainingSet =
TrainingSetImport.importFromFile("iris.dataset.txt", 4, 3, " ");
trainingSet.normalize(new MaxMinNormalizer());
TrainingSet[] trainAndTestSet =
trainingSet.createTrainingAndTestSubsets(70, 30);
trainAndTestSet [0].saveAsTxt("TrainingSubSet_1_70.txt", " ");
trainAndTestSet [1].saveAsTxt("TestSubSet_1_30.txt", " ");
```

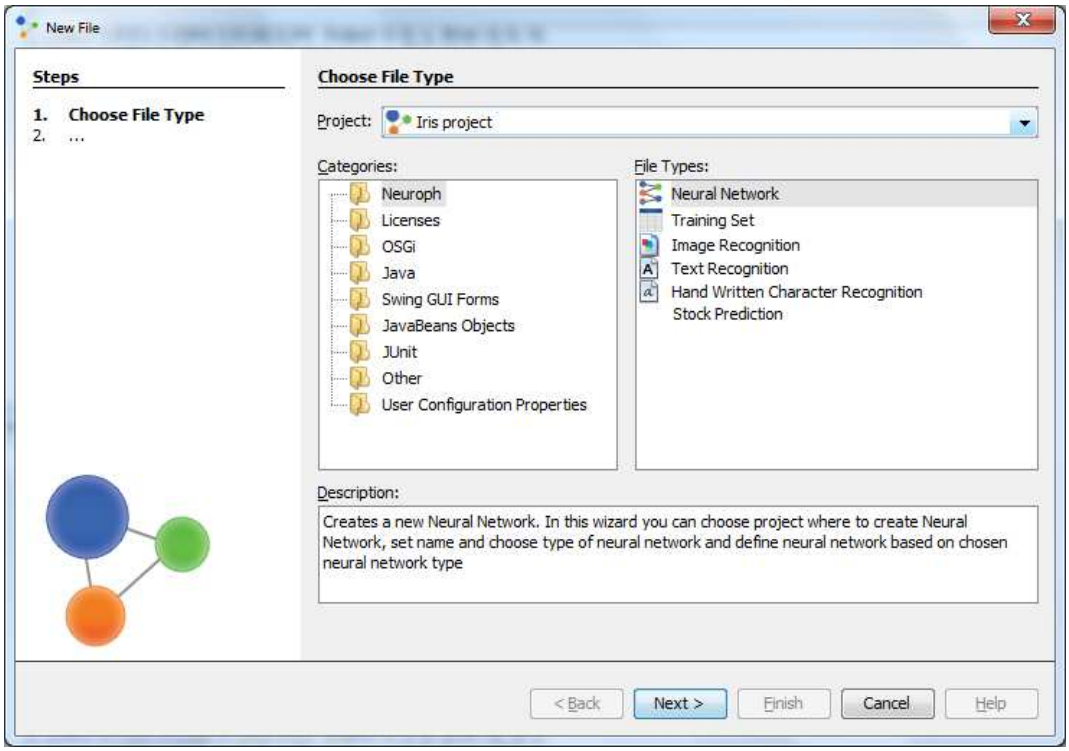
Dati listing Java koda kreira dva fajla koj sadrze 70 i 30% slučajno odabranih uzoraka iz celokupnog skupa podataka, i predstavljaju skupove podataka za trening i testiranje.
Nakon toga kreirani tekstualni fajlovi sa podacima za trening i testiranje uvezeni su u razvojno okruženje za neuronske mreže NeurophStudio (slika 27).
Dijalog za uvoz podataka i kreiranje skupova za trening i testiranje pokreće se iz glavnog menija aplikacije Neuroph Studio File > New > Neuroph > Training Set



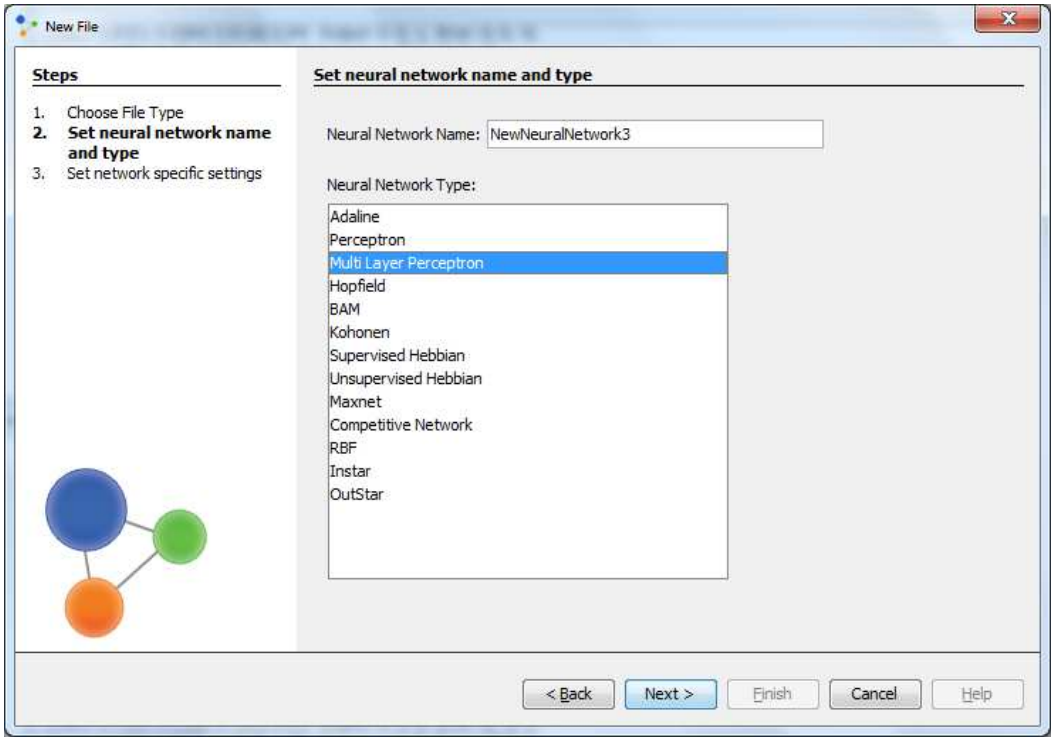
Slika 27. Dijalog za uvoz podataka za trening u aplikaciju Neuroph Studio

Trening neuronske mreže

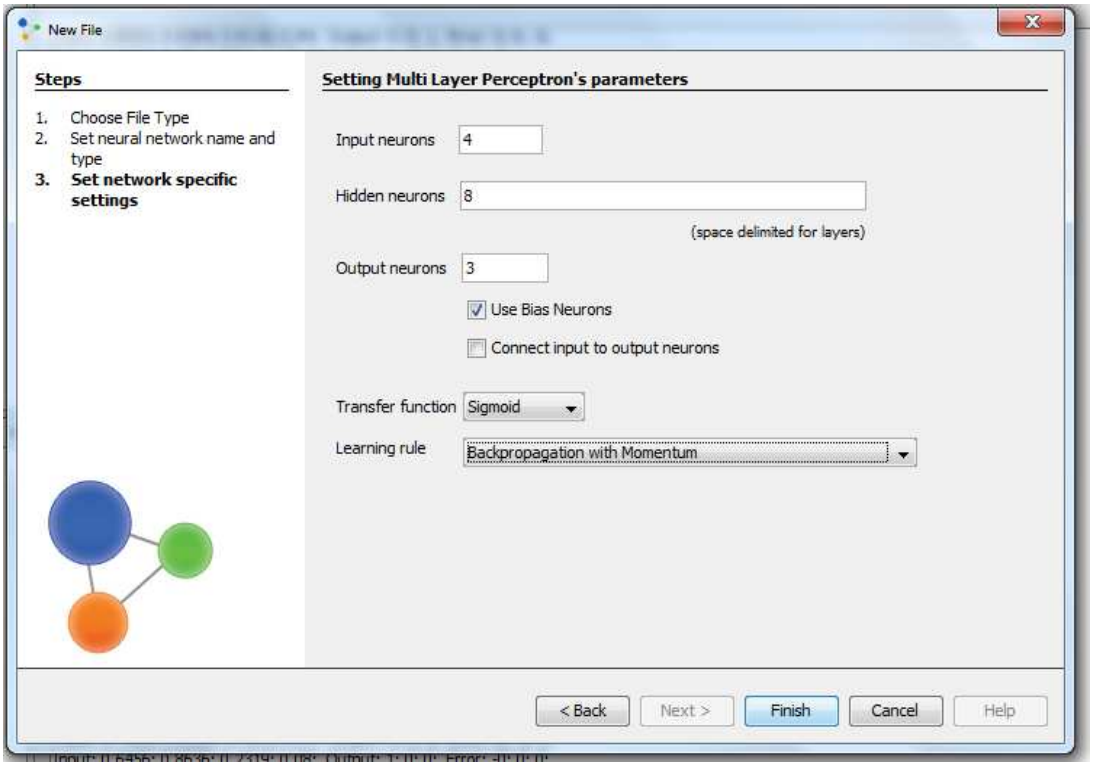
Nakon uvoza podataka, potrebno je kreirati neuronsku mrežu koja će se istrenirati. Neuronske mreže se u aplikaciji Neuroph Studio kreiraju pomoću wizarda za kreiranje neuronskih mreža koji se pokreće iz glavnog menija sa File > New > Neuroph > Neural Network (slike 28, 29, 30).



Slika 28. Pokretanje wizarda za kreiranje neuronske mreže

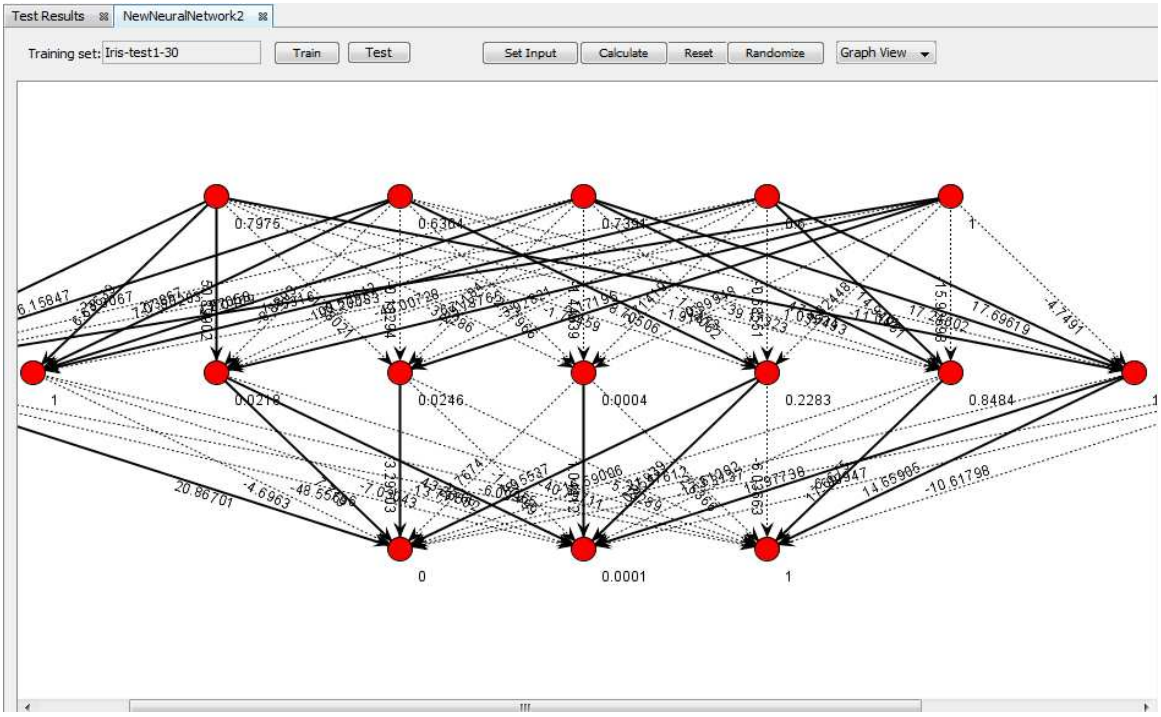


Slika 29. Wizard za kreiranje neuronske mreže – korak 1, izbor vrste neuronske mreže



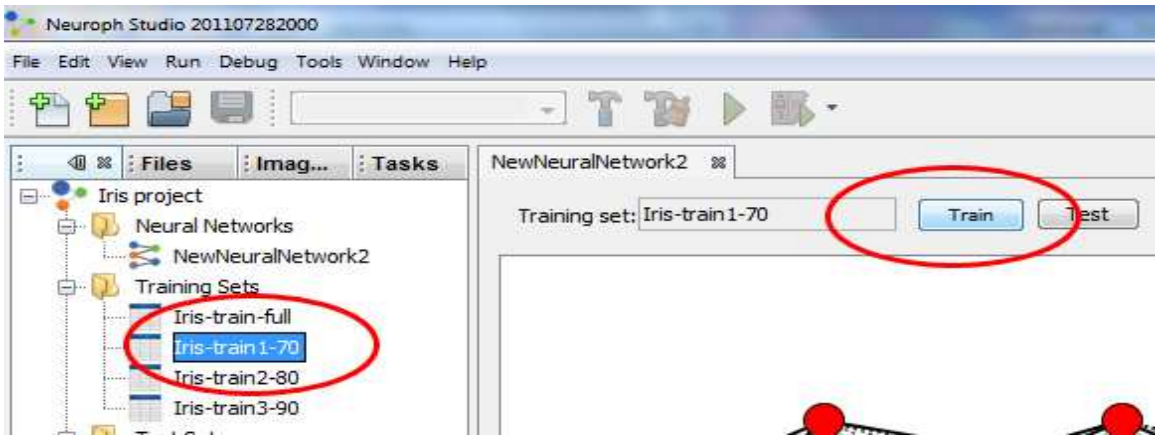
Slika 30. Wizard za kreiranje neuronske mreže – korak 2, podešavanje parametara neuronske mreže

Nakon pokretanja wizarda potrebno je izabrati vrstu neuronske mreže (slika 29), a zatim podestiti specifične parametre izabrane neuronske mreže (slika 30). Na slici 31. dat je vizuelni prikaz kreirane neuronske mreže



Slika 31. Vizuelni prikaz neuronske mreže kreirane pomoću wizarda

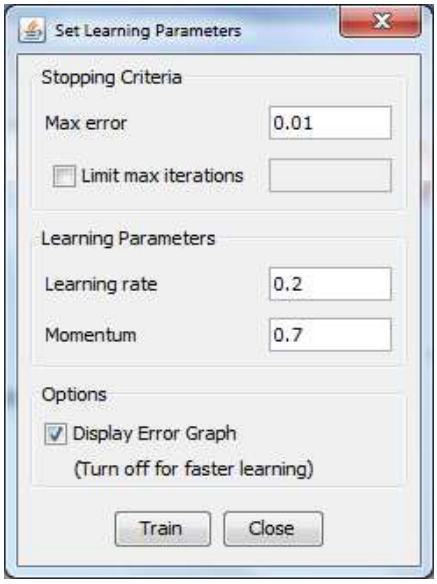
Sam trening neuronske mreže se pokreće izborom odgovarajućeg skupa za trening iz Poject prozorcica i klikom na dugme train (slika 32).



Slika 32. Pokretanje treninga neuronske mreže

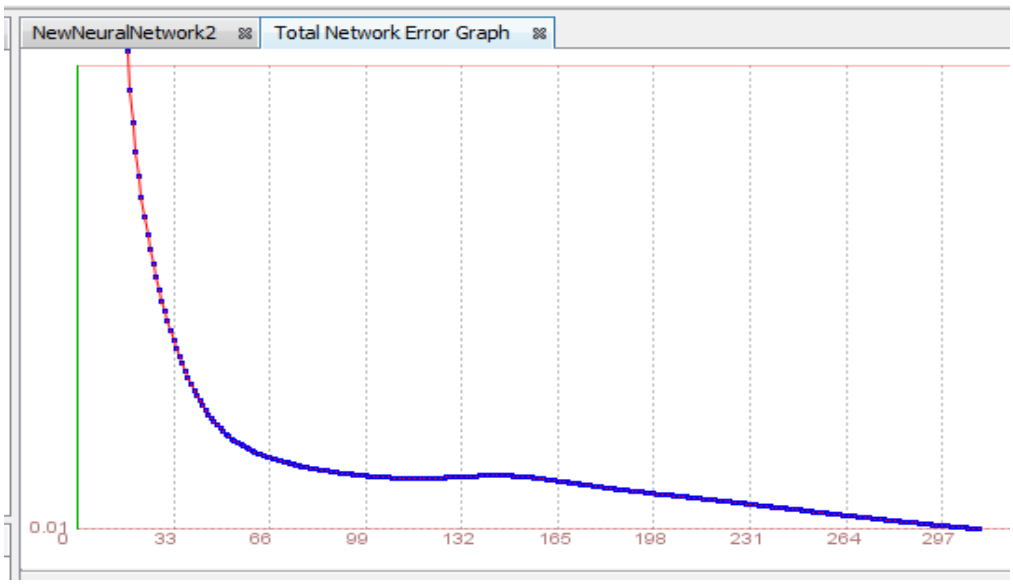
Nakon pokretanja treninga otvara se dijalog u kome je potrebno podesiti parametre algoritma za trening neuronske mreže (slika 33). Na slici su date početne

eksperimentalne vrednosti parametara za learning rate 0.2, maksimalu grešku 0.01 (1%) i momentum aparametar 0.7.



Slika 33. Dijalog za podešavanje parametara algoritma za trening

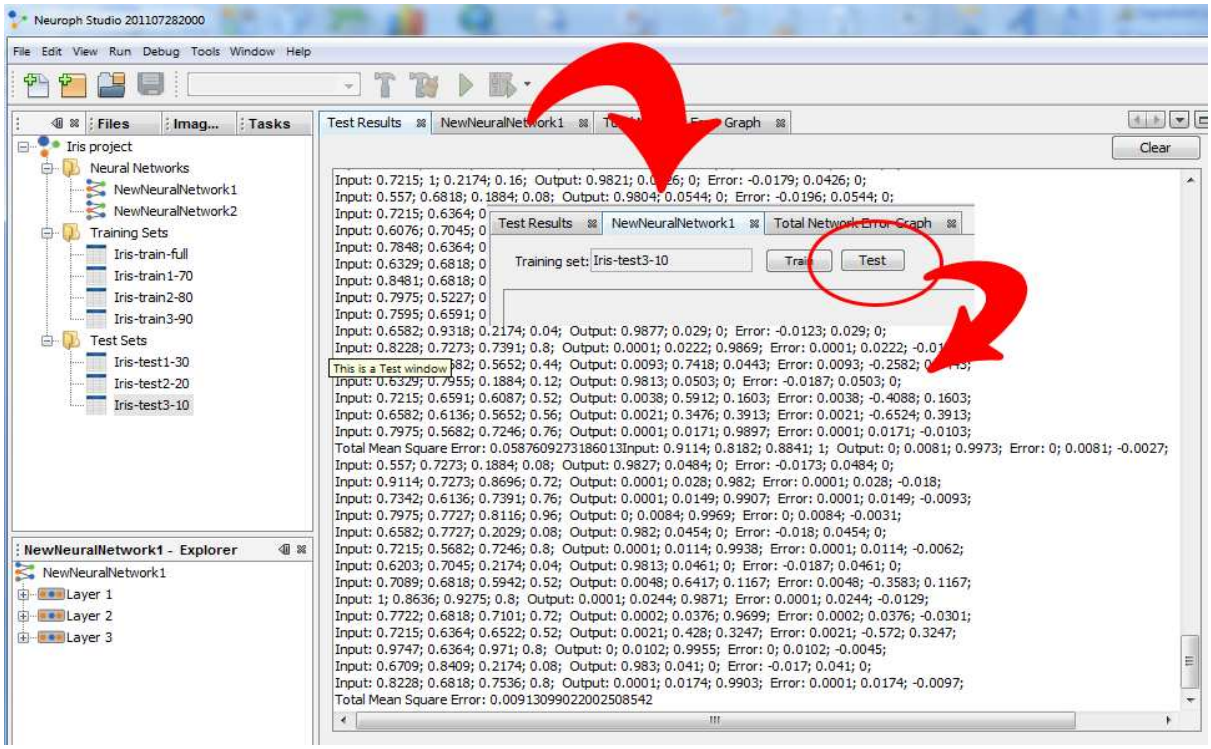
Nakon zadavanja parametara, počinje izvršavanje algoritma za trening koji vrši automatsko podešavanje neuronske mreže kako bi se minimizovala odstupanja u funkcionalnosti mreže koja se iskazuju ukupnom srednjom kvadratnom greškom mreže. Na slici 34. Dat je grafik greške za izvršeni trening u okviru ovog primera. Trening se izvršava sve dok vrednost srednje kvadratne greške ne bude ispod zadate vrednosti parametra Max error, koja se zadaje u prethodnom dijalogu (slika 33).



Slika 34. Grafik srednje kvadratne greške tokom treninga neuronske mreže

Testiranje neuronske mreže

Testiranje neuronske mreže za izabrani test set vrši se klikom na dugme Test (slika 35), nakon čega se otvara prozor sa prikazom grešaka za sve pojedinače elemente iz skupa za testiranje, ako i ukupna srednja greška za sve podatke.



Slika 35. Testiranje neuronske mreže

U tabeli 16. Dati su rezultati treninga i testiranja za sve eksperimentalne parove trening i test skupova podataka.

Tabela 16. Rezultati treninga i testiranja neuronskih mreža

Rb.	**Trening i test skupovi**	**Broj iteracija treninga**	**Srednja kvadratna greška prilikom testiranja**
1.	70% trening 30% test	321	0.017
2.	80% trening 20% test	4497	0.033
3.	90% trening 10% test	6373	0.00008

Dobijeni rezultati pokazuju zanimljive karakteristike neuronskih mreža:

1. Malo povećanje skupa za trening (10%) višestruko je povećalo broj iteracija potrebnih za trening (više od 10 puta) poredeći treninge 1 i 2.
2. Iako je u treningu 2. Korišćeno više podataka za trening, srednja kvadratna greška je veća (0.033 > 0.017)
3. Treći trening takođe je trajao duže, ali za rezultat je imao najmanju grešku prilikom testiranja u odnosu na druga dva slučaja.

Ovaj primer pokazuje nedeterminističku prirodu neuronskih mreža, i kako se eksperimentalnim putem dolazi do najboljih rezultata. Da bi se ovi rezultati potvrdili, u realnom scenariju je potrebno ponoviti ih više puta.

Ovaj primer demonstrira kako okruženje za razvoj neuronskih mreža Neuroph Studio i Neuroph framework obezbeđuju sve funkcionalnosti za celokupan proces treniranja neuronskih mreža. Moguća su mnoga unapređenja, koja bi se odnosila na automatizaciju celog procesa trening ai testiranja.

Primena istrenirane neuronske mreže za klasifikaciju

Iz aplikacje Neuroph Studio istrenirana mreža se snima kao serijalizovani Java objekat, koji se može dalje koristiti u korisničkoj Java aplikaciji. Sledeći listing prikazuje primer kako se u java kodu, učitava i koristi snimljena neuronska mreža:

```
// učitaj snimljenu neuronsku mrežu
NeuralNetwork myNeuralNet = NeuralNetwork.load("myNeuralNet.nnet");
// postavi ulazni vektor koji treba kalsifikovati
myNeuralNet.setInput(0.81, 0.72, 0.77, 0.92);
// procesiraj ulaz/izračunaj mrežu
myNeuralNet.calculate();
// uzmi izlaz neuronske mreže tj. klasifikaciju
double[] output = myNeuralNet .getOutput();
```

6.3. Prepoznavanje slika

Opis primera: Primer demonstrira primenu specijalizovanog wizarda za kreiranje neuronskih mreža za prepoznavanje slika, u okviru razvojnog okruženja NeurophStudio

Opis problema: Problem prepoznavanja slika (i prepoznavanja uopšte) je pored klasifikacije, još jedan tipičan primer primene neuronskih mreža. Samo prepoznavanje slika ne može se formalno-matematički opisati i ne postoji odgovarajući algoritamski postupak.. Takođe, prilikom prepoznavanja moguća su razna odstupanja / varijacije originalne slike, što sve dodatno otežava prepoznavanje. Zbog svega toga neuronske mreže, predstavljaju interesantno rešenje za ovu vrstu problema.
U ovom primeru obrađen je problem prepoznavanja crno-belih silueta životinja

Podaci: Skup podataka za trening čini 5 pojedinačnih slika sa siluetama životinja koje treba prepoznati: pas, mačka, zec, riba i ptica (slika 36). Sve slike su dimenzija 100x100 piksela (tačaka) i od RGB informacija o boji svakog piksela (tačke) formira se ulazni vektor za neuronsku mrežu.

Slika 36. Slike za prepoznavanje

Planiranje treninga

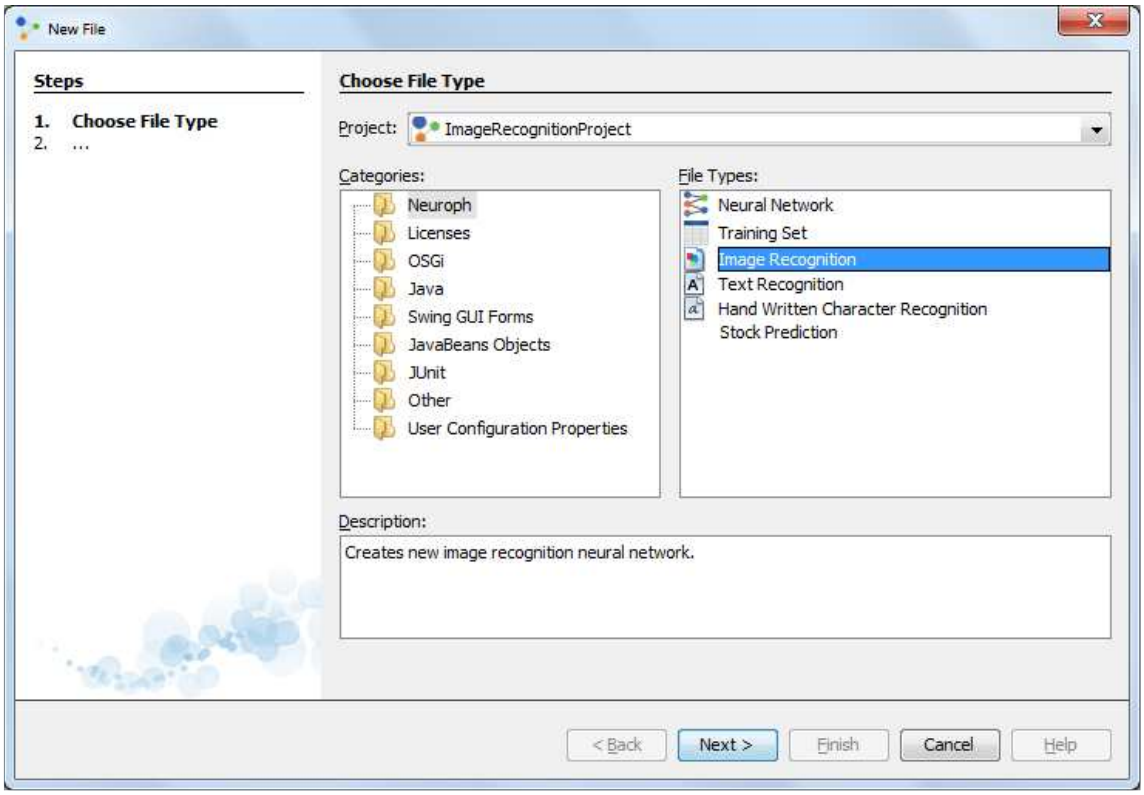
Za rešavanje problema prepoznavanja slika, u skup podataka za trening, pored slika koje treba da budu prepoznate, potrebno je uključiti i dodatne slike koje će sprečiti pogrešno prepoznavanje, a sastoje se od potpuno crvenih, plavih i zelenih površina 100x100 piksela.
Za ovu vrstu može takođe se može koristiti neuronska mreža tipa višeslojni perceptron (Multi Layer Perceptron) i Backpropagation algoritam za učenje.

Priprema podataka

Priprema podataka za trening podrazumeva normalizaciju slika, tj. svođenje slika na iste dimenzije, a po potrebi i dodatno ujednačavanje boja, koje u ovom sličaju nije potrebno. Po potrebi se mogu primeniti i razne tehnike preprocesiranja slika kao npr. detekcija ivica, ekstrakcija karakterističnih detalja i sl.

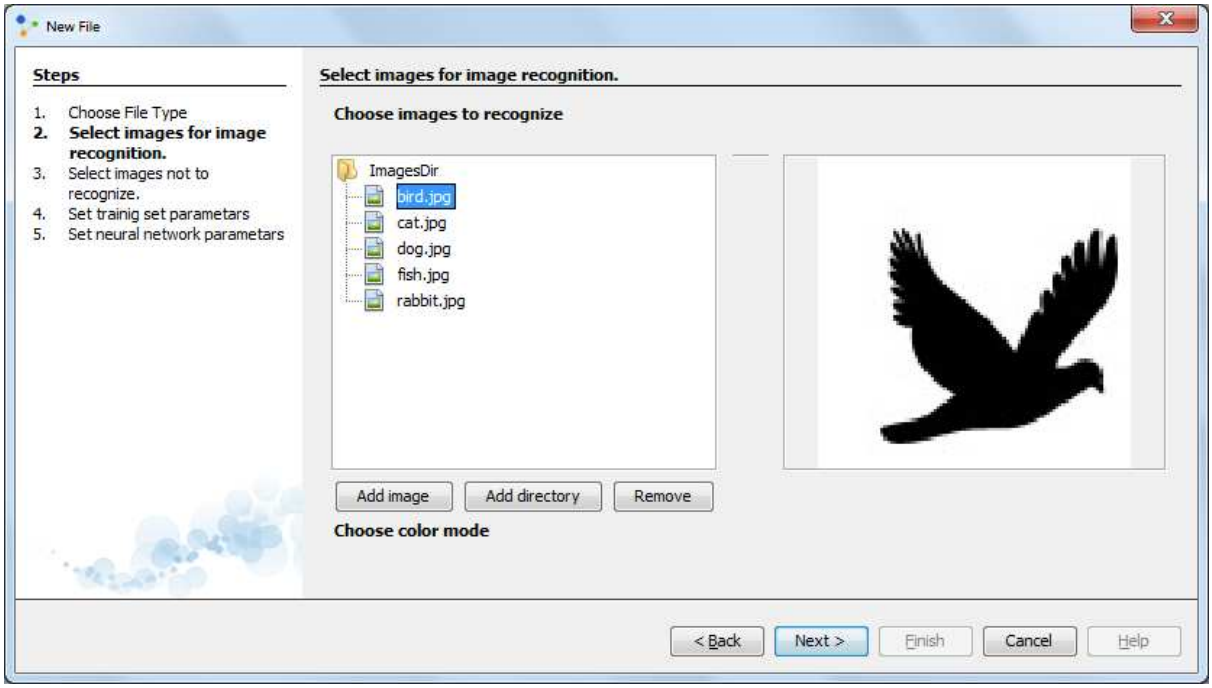
Trening neuronske mreže

Sama neuronska mreža i generisanje skupa podataka za trening na osnovu slika vrši se pomoću specijalizovanog wizard-a u okviru aplikacije Neuroph Studio. Wizard se pokreće iz glavnog menija aplikacije File > New > Neuroph > Image Recognition (slika 37.)



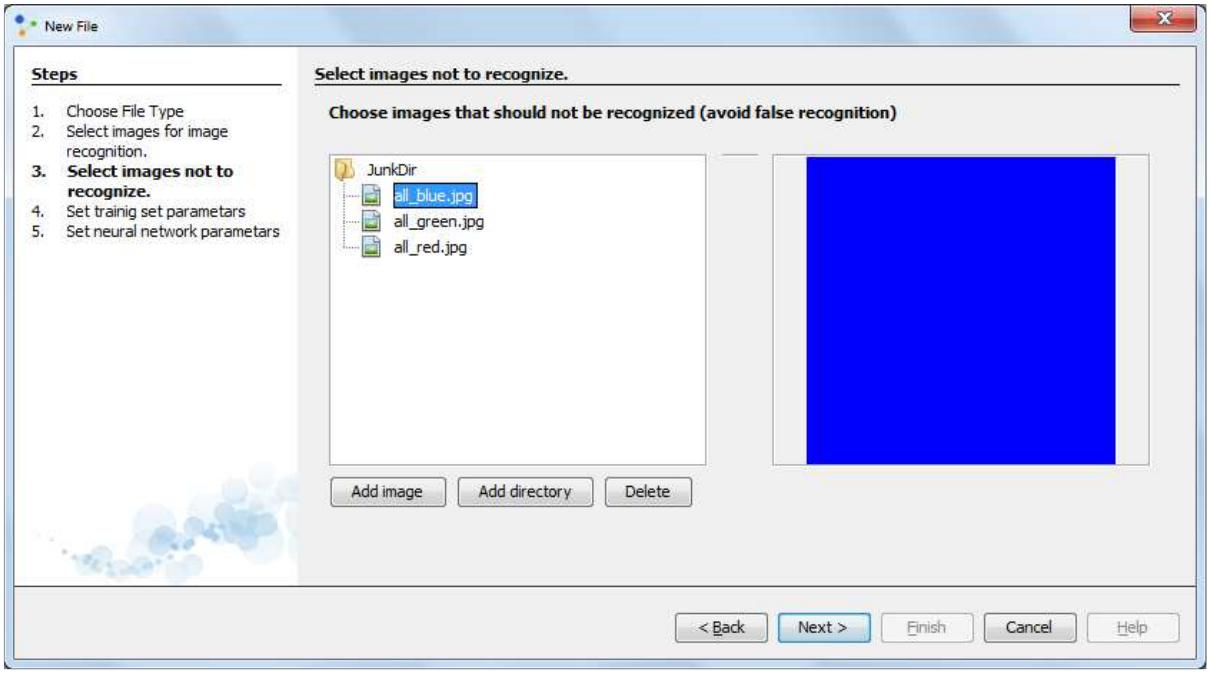
Slika 37. Pokretanje wizarda za prepoznavanje slika

U prvom koraku wizarda biraju se slike za prepoznavanje (slika 38). Dodavanje slika se vrši pomoću dugmeta *Add Image*, a moguće je i dodavanje celog direktorijuma sa slikama pomoću *Add directory*.



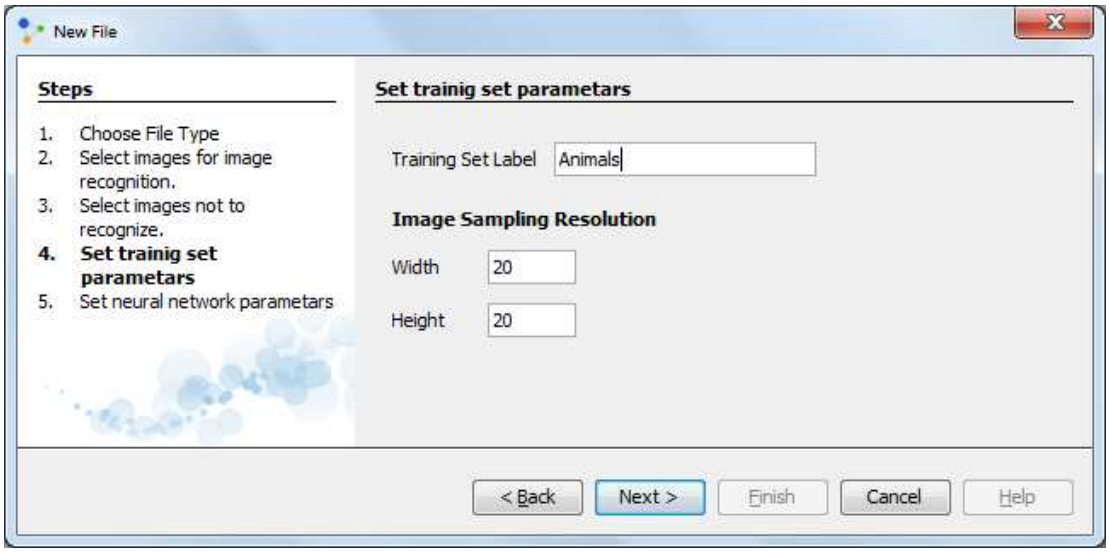
Slika 38. Wizard za prepoznavanje slika – korak 1: izbor slika za prepoznavanje

Zatim se u drugom koraku wizarda, zadaju slike koje pomažu da se izbegne pogrešno prepoznavanje (slika 39). Ovo su obično površine ispunjene osnovnim komponentama crvene, plave i zelene boje, a po potrebi mogu se dodati i druge.



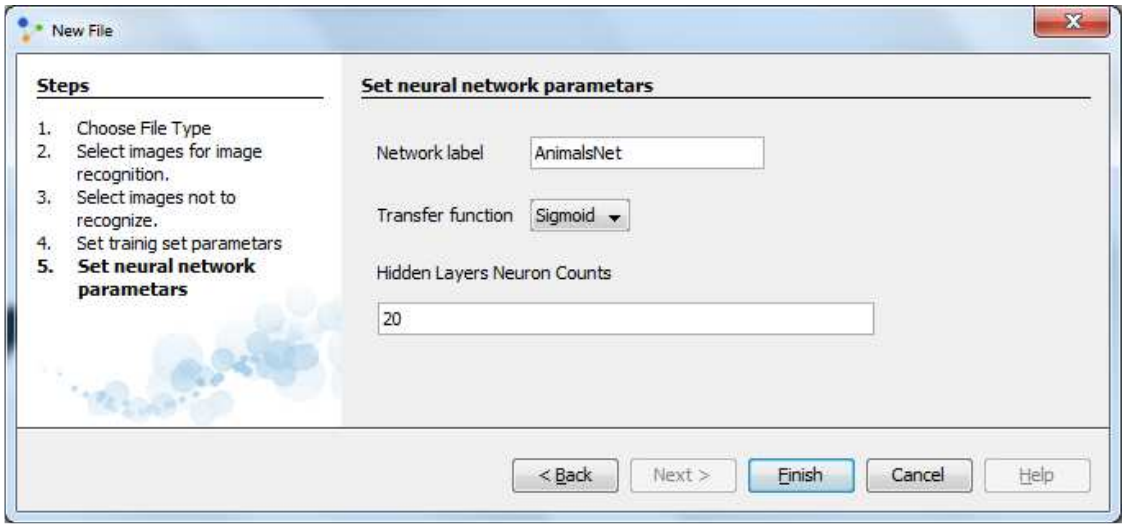
Slika 39. Wizard za prepoznavanje slika – korak 2: izbor slika koje pomažu izbegavanje pogrešnog prepoznavanja

U sledećem koraku zadaju se naziv skupa podataka koji se kreira na osnovu izabranih slika u prethodnim koracima, i rezolucija za prepoznavanje. Rezolucija za prepoznavanje predstavlja dimenzije (širina i visina) na koju će slike za trening i prepoznavanje biti skalirane da bi se poboljšale performanse (slika 40). Što je slika manja to su trening i prepoznavanje brži, ali u smanjivanju treba ići do granice dok se dobijaju zadovoljavajući rezultati, što se utvrđuje eksperimentalnim putem.



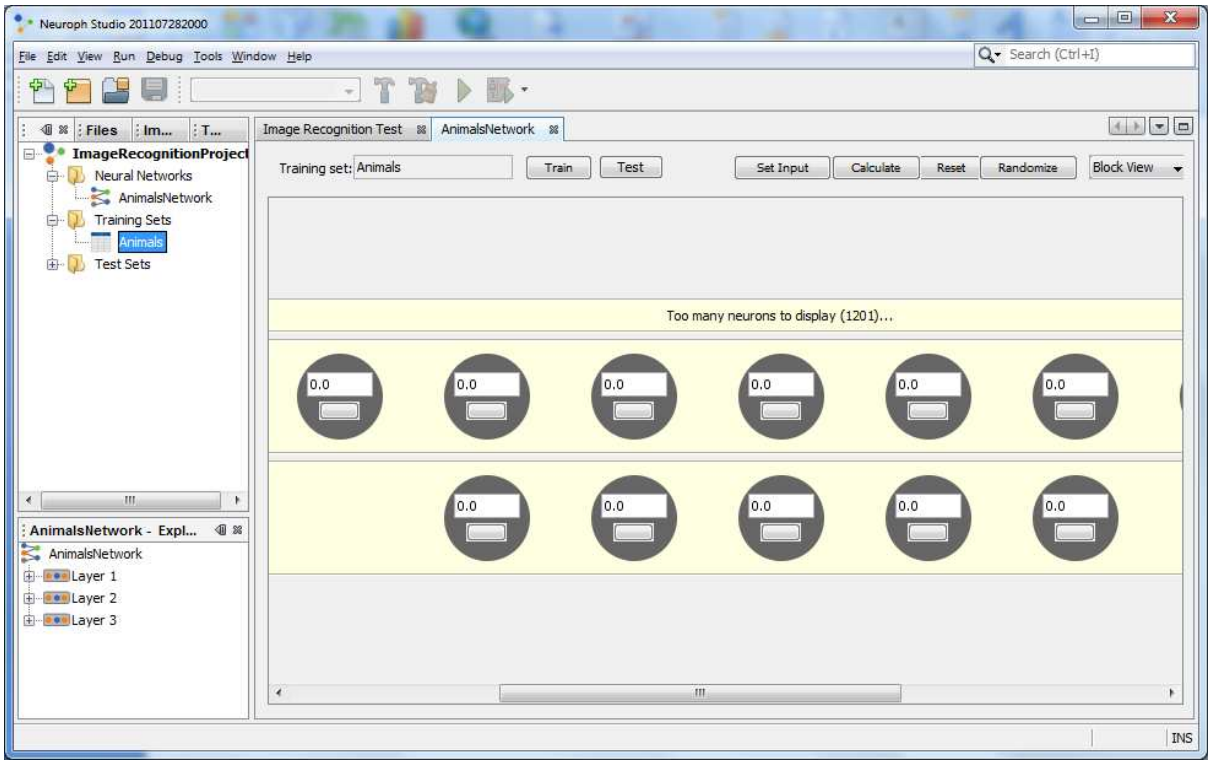
Slika 40. Wizard za prepoznavanje slika – korak 3: kreiranje skupa podataka za trening, zadavanje rezolucije za prepoznavanje

Poslednji korak wizarda je zadavanje podešavanja neuronske mreže i to vrste funkcije transfera koja će se koristiti i broj skrivenih neurona (slika 41). Ova podešavanja se odnose na neuronsku mrežu tipa Multi Layer Perceptron sa Momentum Backpropagation algoritmom za učenje. Najbolje vrednosti za broj skrivenih neurona zavisi od konkretnog problema i utvrđuje se eksperimentalnim putem.



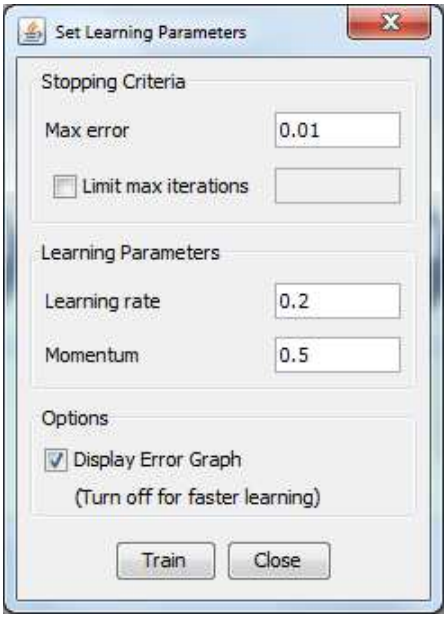
Slika 41. Wizard za prepoznavanje slika – korak 4: zadavanje parametara neuronske mreže

Nakon prolaska kroz wizard, automatski se kreiraju skup podataka za trening od izabranih slika, i neuronska mreža sa zadatim podešavanjima (slika 42). Sama logika za kreiranje podataka za trening n osnovu slika i neuronske mreže za prepoznavanje slika implementirani su u okviru Neuroph frameworka koji obezbeđuje jednostavan API za ove operacije, dok wizard u okviru Neuroph Studio-a obezbeđuje grafički interfejs za unos slika i podešavanja.



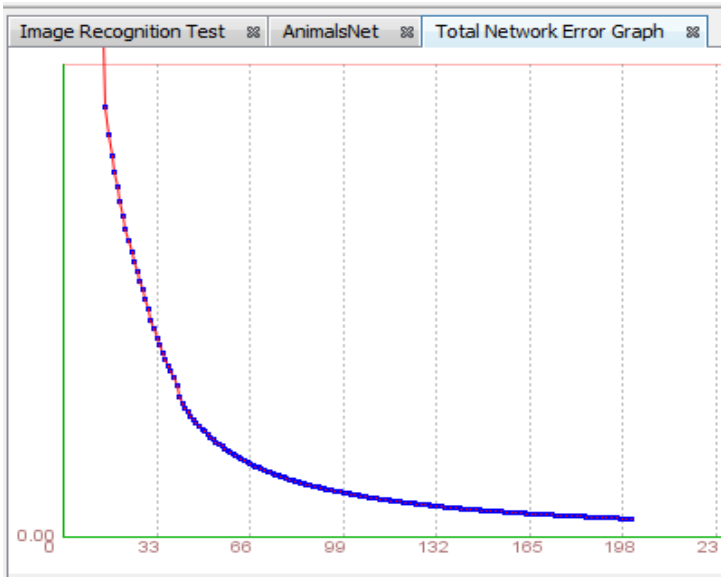
Slika 42. Kreirana neuronska mreža za prepoznavanje slika

Nakon kreiranja skupa podataka za trening i neuronske mreže, pokreće se trening mreže klikom na dugme *Train*, nakon čega se otvara dijalog za podešavanej parametara treninga (slika 41). Za ovaj problem, eksperimentalnim putem, pronađene su vrednosti parametara koje daju dobre rezultate (slika 43).



Slika 43. Dijalog za zadavanje parametara za trening

Na slici 44 dat je grafik srednje kvadratne greške tokom treninga, na kome se vidi da je greška mreže, spuštena na prihvatljivi nivo manji od 0.01 za oko 214 iteracija.



Slika 44. Grafik srednje kvadratne greške tokom treninga mreže

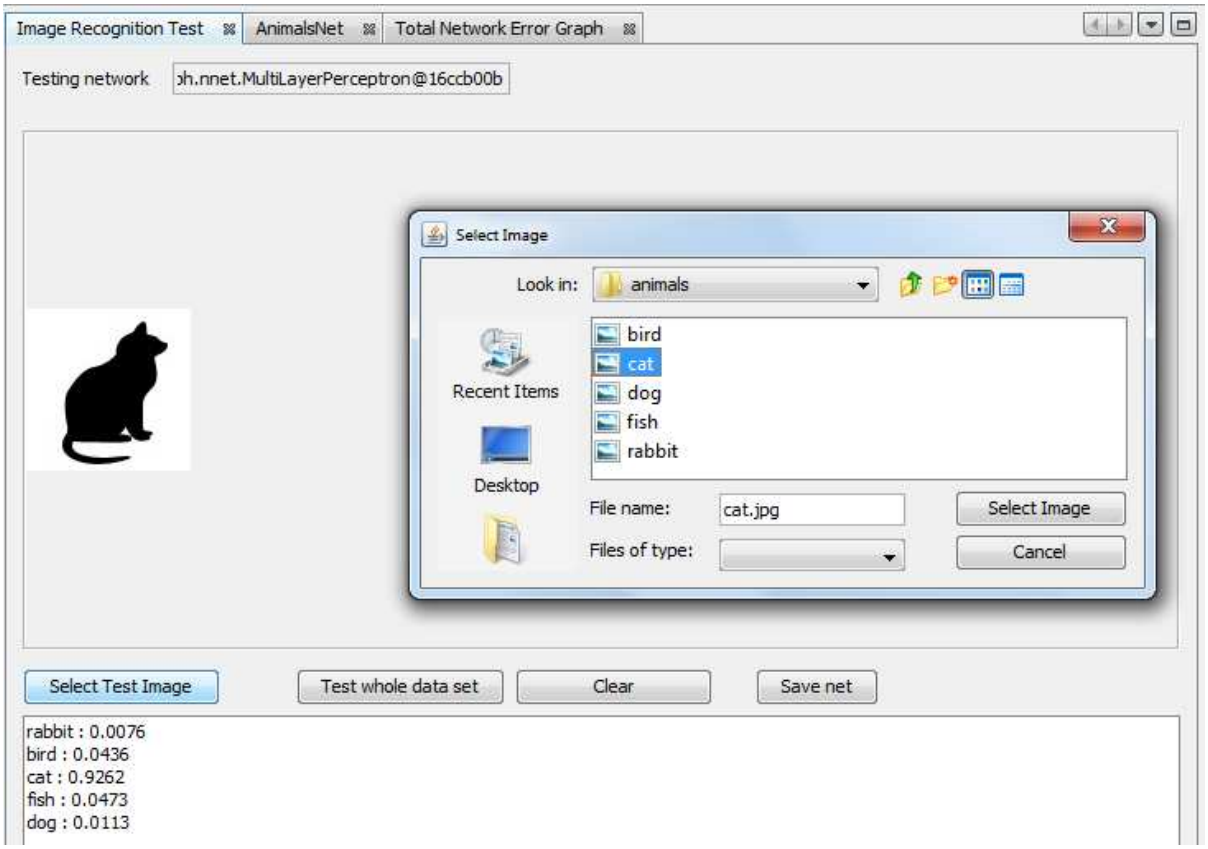
U tabeli 17. dati su rezultati treninga za nekoliko neuronskih mreža sa različitim brojem skrivenih neurona, i parametrima algoritma za trening.

Tabela 17. Rezultati treninga sa različitim podešavanjima neuronske mreže i algoritma za trening

Broj skrivenih neurona	**Learning rate**	**Momentum**	**Broj iteracija**
20	0.2	0.5	214
20	0.2	0.7	423
20	0.5	0.5	72
20	0.7	0.5	beskonačno
30	0.2	0.5	302

Testiranje neuronske mreže

Nakon treninga neuronske mreže, u posebnom prozoru za testiranje vrši se provera prepoznavanja slika (slika 45). Nakon izbora slike, u posebnom polju ispisuju se vrednosti sa izlaza neuronske mreže, tj. rezultati prepoznavanja. Svakom izlaznom neuronu dodeljena je oznaka tj. naziv slike, i neuron sa najvećom izlaznom vrednošću predstavlja prepoznatu sliku. U primeru na slici 43. Prepoznata je slika mačke sa verovatnoćom 0.9, dok je verovatnoća da je naslici zec 0.007.



Slika 45. Prozor za testiranje neuronske mreže za prepoznavanje slika

Primena istrenirane neuronske mreže za prepoznavanje slika

Neuronska mreža istrenirana za prepoznavanje slika može se sačuvati kao serijalizovani Java objekat, i kasnije koristiti u drugim Java aplikacijama. Na listingu je dat programski kod sa objašnjenjima u vidu komentara koji demonstrira korišćenje neuronske mreže za prepoznavanje slika.

```
// učitaj snimljenu neuronsku mrežu
NeuralNetwork nnet =
NeuralNetwork.load("MyImageRecognition.nnet");
// uzmi plugin za prepoznavanje slika iz neuronske mreže
ImageRecognitionPlugin imageRecognition =
(ImageRecognitionPlugin)nnet.getPlugin(ImageRecognitionPlugin.cl
ass);

 try {
    // samo prepoznavanje slika se vrši pomoću metode
    // recognizeImage
      HashMap output = imageRecognition.recognizeImage(
                       new File("someImage.jpg"));
      // ispisi rezultate prepoznavanja
       System.out.println(output.toString());
 } catch(IOException ioe) {
        ioe.printStackTrace();
 }
```

7. EVALUACIJA

U ovom poglavlju data je evaluacija Neuroph framework-a nakon razvoja novih funkcionalnosti. Evaluacija obuhvata sledeće elemente:

1. uporedni pregled funkcionalnosti i *framework-a* u celini;
2. jednostavnost korišćenja i zadovoljstvo korisnika;
3. testiranje performansi.

Dati model evaluacije se može primeniti kao opšti model za evaluaciju softvera iz oblasti inteligentnih sistema.

7.1. Pregled funkcionalnosti i karakteristika framework-a u celini

Tokom ovog rada na ovoj distertaciji razvijene su nove funkcionalnosti Neuroph framework-a koje značajno unapređuju njegove mogućnosti primene, ali i učvršćuju poziciju u odnosu na konkurentske framework-e, pre svega Encog. Razvijene funkcionalnosti obuhvataju:

1. nove vrste neuronskih mreža i algoritmi za učenje: *ResilientPropagation, PCANetwork,GeneralizedHebbianLearning, CounterpropagationNetwork, CounterpropagationLearning, InstarOutstarNetwork, InstarOutstarLearning, BoltzmanNetwork;*
2. podrška za normalizaciju podataka: *Max, MaxMin, DecimalScale;*
3. tehnike za generisanje slučajnih težinskih koeficijenata: *Range, Distort, Gaussian, NguyenWidrow;*
4. ulazno/izlazni adapteri: *File, URL, JDBC, Stream;*
5. merenje performansi (*benchmark*);
6. nove funkcije transfera: *Sin, Log;*
7. izmene u strukturi osnovnih klasa, zahvaljujući kojima je omogućena još veća fleksibilnost i podrška za razne varijacije algoritama za učenje.

U tabelama 18, 19 i 20 dat je uporedni pregled glavnih funkcionalnosti za Neuroph, Encog i Joone *framework-e*. Sa X su označene funkcionalnosti Neuroph-a koje su razvijene u okviru ovog istraživanja. U prilogu 1 dato je još detaljnije poređenje dodatnih funkcionalnosti..

Tabela 18. Uporedna tabela podržanih vrsta neuronskih mreža

Vrste neuronskih mreža			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Adaline	+	+	
Perceptron	+	+	+
Multi Layer Perceptron	+	+	+
Hebbian Network	+	+	
Hopfield neural network	+	+	
Bidirectional Associate memory (BAM)	+	+	
Boltzmann machine	X	+	
Counterproragation neural network (CPN)	X	+	
Recurrent-Elman		+	+
Recurrent-Jordan		+	+
Recurrent-SOM		+	+
RBF network	+	+	
Maxnet	+		
Competitive Network	+		
Kohonen SOM	+	+	+
Instar	+		
Outstar	+		
Adaptive resonance theory (ART1)		+	
Time Delay neural network			+
Neuro Fuzzy Perceptron	+		
Support Vector Machine (SVM)		+	
PCA neural network	X		+

Tabela 19. Uporedna tabela podržanih algoritama za učenje neuronskih mreža

Algoritmi za učenje			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
LMS	+	+	
PerceptronLearning	+		
Binary delta rule	+		
Smooth Delta Rule	+	+	
Hebbian learning (supervised and unsupervised)	+	+	
Simulated Annealing		+	
Backpropagation	+	+	+
Auto Backpropagation	+	+	
Resilient backpropagation	X	+	+
Manhattan Update Rule Propagation		+	
Levenberg Marquardt (LMA)		+	
Scaled Conjugate Gradient		+	
Competitive learning	+	+	
Hopfield learning	+	+	
Kohonen	+	+	+
Instar learning	+	+	
Outstar learning	+	+	
Instar/Outstar	X	+	
Genetic algorithm training		+	
NEAT	+	+	

Tabela 20. Uporedna tabela raznih dodatnih funkcionalnosti

Dodatne funkcionalnosti			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Normalizacija podataka	X	+	
Benchmark	X	+	
Tehnike generisanja slučajnih brojeva za inicijalizaciju težina	X	+	+
Ulazni adapteri za bazu podataka, URL, stream	X	+	+
Izlazni adapteri za bazu podataka, fajl, stream	X		+

Razvoj navedenih novih funkcionalnosti, značajno je približio Neuroph njegovom glavnom cilju, a to je da u potpunosti podrži, automatizuje i obezbedi alate za sve operacije tokom procesa rešavanja problema pomoću neuronskih mreža. Iz primera datih u 6. poglavlju vidi se da je već sada Neuroph veoma blizu tog cilja, i da pored

samog framework-a, obezbeđuje i veoma moćno razvojno okruženje za neuronske mreže Neuroph Studio, zasnovano na NetBeans platformi.
Posebno je značajna evolucija same osnove frameworka, jer se uvođenjem novih zahteva ukazala potreba za restruktuiranjem i modifikacijom osnovnih klasa supervizornih algoritama za učenje. Pored direktne potrebe da odgovori na zahtevane funkcionalnosti, ove modifikacije učinile su moguće i razne druge varijacije algoritama za učenje u budućnosti, što Neuroph framework pretvara u istraživačku platformu za neuronske mreže.
Sa svim ovim karakteristikama, Neuroph je odlično rešenje za istraživanja i edukaciju u oblasti neuronskih mreža, za brzi razvoj prototipova i eksperimentisanje sa raznim vrstama neuronskih mreža, i modifikacijama algoritama za učenje. Neuroph se takođe može koristiti i u produkciji, za rešavanje realnih problema, a u slučaju treninga veoma velikih količina podataka treba razmotriti i korišćenje Encog-a kao rešenja sa boljim performansama. Međutim, sa razvojem podrške za merenje performansi, omogućeno je sistematsko testiranje performansi Neuroph framework-a, što će biti glavni parametri u daljim fazama razvoja. Poboljšanje performansi na nivou implementacije, dodavanje naprednih algoritama za učenje i podrška za paralelno procesiranje su pravci budućeg razvoja.
Ukoliko Neuroph framework sagledamo u kontekstu postavljenih ciljeva razvoja definisanih u odeljku 3.2, i funkcionalnosti identifikovanih za razvoj u poglavlju 3.3. može se reći da su u potpunosti ostvareni planirani rezultati, i to:

1. razvijene su dodatne funkcionalnosti koji su Neuroph učinile konkurentnim u odnosu na druge framework-e – iz uporednog pregleda se vidi da je dopunjen izvestan broj funkcionalnosti koje su u odnosu na druge framework-e nedostajale u prethodnim verzijama;
2. pri tom, nove funkcionalosti su nadograđene na već postojeće, ili su uklopljene u postojeću strukturu framework-a, čime je razvoj framework-a zasnovan na principima koji su se uspešno pokazali u praksi, uz održavanje kompatibilnosti sa prethodnim verzijama gde god je to bilo moguće;
3. povećane su mogućnosti za praktičnu primenu i pojednostavljeno je korišćenje – razvijen napredni algoritam za učenje (Resilient Propagation), obezbeđena podrška za normalizaciju podataka, inicijalizaciju težina, ulaz/izlaz iz raznih izvora.

Razvoj ResilientPropagation algoritma ima poseban značaj, jer on u ovom trenutku predstavlja najnapredniji algoritam za učenje u Neuroph–u, a njegovim razvojem takođe su otvorene mogućnosti da se kroz dalju nadogradnju implementiraju i drugi slični algoritmi koje Encog ima, a Neuroph trenutno ne: Manhattan Update Rule Propagation, Levenberg Marquardt i Scaled Conjugate Gradient.
Dodatne funkcionalnosti koje obuhvataju normlizaciju podataka, tehnike generisanje slučajnih težinskih koeficijenata, ulazno/izlazne adaptere i merenje performansi uopšte nisu postojale u prethodnim veryijama Neuroph-a i njihova implementacija predstavlja značajan korak i važno povećanje upotrebne vrednosti Neuroph-a.

Pored svih dosad navedenih funkcionalnih karakteristika, posebno treba istaći da integrisano razvojno okruženje približno profesionalnog nivoa kvaliteta, izdvajaju

Neuroph u odnosu na druga rešenja. Zahvaljujući svojim karakteristikama, Neuroph u praksi omogućava:

1. implementaciju specifičnih, u potpunosti prilagođenih procedura treninga neuronskih mreža;
2. razvoj novih i prilagođavanje postojećih algoritama za učenje, specifičnostima problema koji se rešavaju;
3. integraciju celog rešenja (framework-a i alata) u okviru jednog razvojnog okruženja.

S druge strane, na osnovu prikaza datog u drugom poglavlju, primera korišćenja datih u prilogu 2, i analizom samog programskog koda opravdano se može reći da je Encog i Joone teže prilagoditi specifičnim potrebama, složeniji su za razumevanje i imaju alate sa grafičkim interfejsom na nivou osnovnog grafičkog interfejsa.

7.2. Jednostavnost korišćenja i zadovoljstvo korisnika

Jedan od osnovnih principa prilikom razvoja Neuroph frameworka je da bude inuitivan i jednostavan za korišćenje za krajnje korisnike kako na nivou grafičkog interfejsa, tako i na nivou framework-a. Na osnovu brojnih komentara korisnika na forumu Neuroph projekta, i uopšte velikom broju korisnika, ovaj pristup se pokazao kao vrlo pozitivan, i na tome će biti stavljen akcenat i u daljem razvoju.
Kako bi se detaljnje sagledala iskustva i mišljenja korisnika Neuroph-a, kreiran je upitnik za evaluaciju sa sledećim pitanjima:

Nivo znanja korisnika (user level)
1. Dobro poznajem teoriju neuronskih mreža
2. Imam praktičnog iskustva u rešavanju problema pomoću neuronskih mreža
3. Imam iskustva u radu sa programskim jezikom Java
4. Imam iskustva u radu sa drugim frameworc-ima za neuronske mreže

Jednostavnost ovladavanja softverom
5. Brzo sam naučio/la da ga koristim
6. Ima dobra prateća uputstva/dokumentaciju
7. Osnovna logička struktura i organizacija framework-a su jasni na prvi pogled
8. Uloge pojedinačnih komponenti u sistemu su jasne na prvi pogled

Jednostavnost korišćenja softvera
9. Softver je jednostavan za korišćenje
10. Intuitivno je jasno gde se nalaze određene funkcionalnosti
11. U svakom trenutku mi je jasno šta softver očekuje od mene
12. U svakom trenutku mi je jasno koju akciju treba da izvršim da bih postigao ono što želim

Upotrebna vrednost
13. Odgovara mojim zahtevima/potrebama
14. Omogućava mi da brže obavim planirane zadatke
15. Ima sve potrebne funkcionalnosti

Zadovoljstvo korisnika
16. Zadovoljan sam softverom.
17. Softver radi upravo onako kako ja želim.
18. Softver je zabavan za korišćenje.

Upitnik je kreiran po uzoru na slične upitnike za evaluaciju softvera, ali je prilagođen potrebama evaluacije Neuroph-a, odnosno prikupljanju podataka od interesa. Upitnik se sastoji iz 5 delova, sa ukupno 17 pitanja u vidu iskaza, na koja korisnik odgovara na tzv. Likert-ovoj skali od 1 do 5, pri čemu 1 znači da se apsolutno ne slaže, a 5 da se apsolutno slaže:

1) u potpunosti se ne slažem
2) ne slažem se
3) neutralan sam
4) slažem se
5) u potpunosti se slažem

Prvi deo upitnika prikuplja podatke o nivou znanja/isksutva korisnika, a ostali delovi utiske prilikom korišćenja softvera koji se uopšteno obuhvataju jednostavnost ovladavanje i korišćenja, a zatim i celokupni utisak i zadovoljstvo korisnika.

Upitnik je bio postavljen na Web strani Neuroph projekta, i korisnici su pozvani da ga popune i na taj način pomognu njegovo dalje usavršavanje. Upitnik je u periodu od 3 meseca popunilo 320 ispitanika, i rezultati su predstavljeni u tabeli 18.

Tabela 18. Rezultati upitnika za evaluaciju Neuroph-a

Pitanja	**Odgovori**						
	U potpunosti se ne slažem	Ne slažem se	Neutralan sam	Slažem se	U potpunosti se slažem	Srednja vrednost	Standardna devijacija
1	112	74	3	89	42	2.61	1.51
2	129	65	2	71	53	2.54	1.58
3	1	3	1	81	234	4.7	0.56
4	86	54	0	124	56	3.03	1.53
5	23	27	31	120	119	3.89	1.20
6	15	19	13	108	165	4.22	1.08
7	16	27	16	107	164	4.27	1.16
8	18	14	18	148	122	4.07	1.06
9	17	29	21	176	77	3.83	1.06
10	21	38	8	130	123	3.93	1.21
11	20	29	22	112	137	3.99	1.19
12	24	27	20	114	135	3.97	1.22
13	24	22	14	157	103	3.92	1.15
14	17	24	29	163	87	3.87	1.06
15	15	54	17	106	128	3.87	1.24
16	7	11	5	133	164	4.36	0.86
17	7	13	3	129	168	4.37	0.87
18	11	17	4	113	175	4.33	0.99

Na osnovu rezultata upitnika mogu se izvesti sledeći zaključci:

1. Korisnici Neuroph-a su:
 - sa (41%) i bez (58%) teorijskog znanja iz oblasti neuronskih mreža (P1);
 - sa (39%) i bez (61%) praktičnog iskustva u rešavanju problema pomoću neuronskih mreža (P2);
 - skoro svi sa (98%) iskustvom u programskom jeziku Java (P3).

Dakle, Neuroph koriste iskusni korisnici ali i početnici u oblasti neuronskih mreža, sa iskustvom u Java programiranju.

2. U pogledu jednostavnosti ovladavanja softverom, dobijeni su veoma dobri rezultati jer se značajna većina ispitanika se izjasnila da su
 - softver brzo naučili (75%) (P5, slika 46);
 - softver ima kvalitetnu dokumentaciju (85%) (P6);
 - da je organizacija i logika softvera inuitivna (P7 - 84% i P8 - 82%).

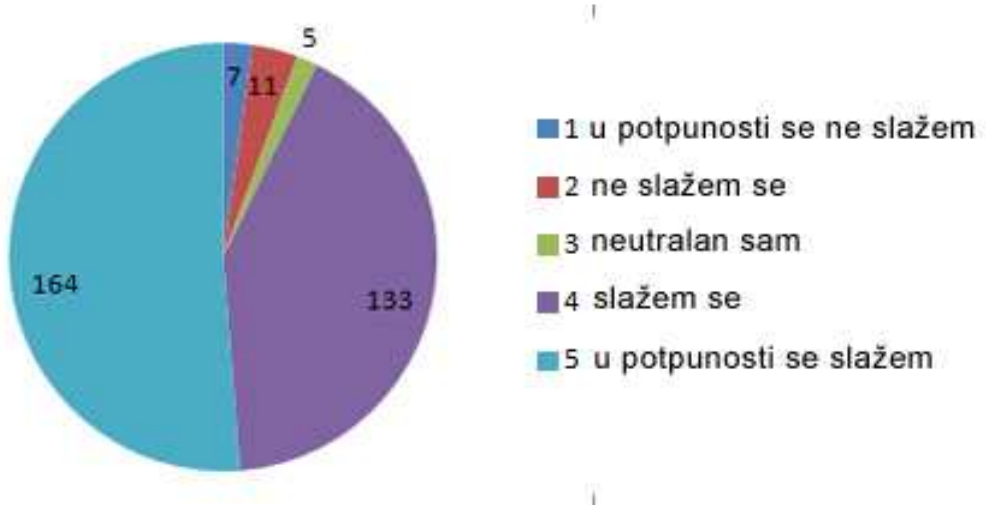
3.U pogledu jednostavnosti korišćenja takođe se velika većina ispitanika izjasnila da je softver jednostavan za korišćenje i da je intuitivno jasno gde se nalaze koje funkcionalnosti (79%)(P9 i P10, slika 47.). U vezi sa tim 78% ispitanika je pozitivno ocenilo interakciju sa softverom (P11 i P12)

4. U pogledu upotrebne vrednosti ispitanici su se u većini izjasnili da:
- softver odgovara njihovim zahtevima/potrebama (81%) (P13);
- omogućava im da brže obavljaju planirane zadatke (78%) (P14)
- ima sve potrebne funkcionalnosti (73%) (P15)

5. I na kraju u pogledu opšteg utiska takođe su dobijeni veoma pozitivni rezultati jer je preko 90% ispitanika reklo da su zadovoljni softverom i da softver radi upravo onako kako oni žele (P16, P17 i P18) (slika 48).

Slika 46. Grafički prikaz odgovora na pitanje 5 – ‘Brzo sam naučio da koristim softver

Slika 47. Grafički prikaz odgovora na pitanje 9 – ‘Softver je jednostavan za korišćenje’



Slika 48. Grafički prikaz odgovora na pitanje 16 – 'Zadovoljan sam softverom'

Korisnicima, koji su pozitivno odgovorili da imaju iskustva u radu sa drugim frameworc-ima za neuronske mreže dato je da popune isti upitnik za framework koji su koristili. Obzirom da Joone framework više nema aktivan razvoj i značajan broj aktuelnih korisnika, on ovde nije detaljnije evaluiran, već je pažnja orijentisana na Encog. U tabeli 19 dati su rezultate ankete sa korisnicima Neuroph-a koji su koristili i Encog.

Tabela 19. Rezultati upitnika za evaluaciju Encog-a

Pitanja	**Odgovori**						
	U potpunosti se ne slažem	Ne slažem se	Neutralan sam	Slažem se	U potpunosti se slažem	Srednja vrednost	Standardna devijacija
1	21	23	2	39	35	1.26	1.58
2	11	22	2	34	51	1.41	1.68
3	1	3	1	67	48	1.62	1.71
4	0	0	0	15	105	1.83	1.88
5	16	25	1	43	35	1.30	1.59
6	7	13	1	56	43	1.48	1.67
7	23	20	2	39	36	1.27	1.60
8	19	26	2	30	43	1.29	1.62
9	34	35	1	23	27	1.04	1.44
10	30	37	3	24	26	1.06	1.43
11	19	23	2	34	42	1.30	1.62
12	17	25	2	37	39	1.30	1.61
13	7	15	3	50	45	1.47	1.67
14	6	21	3	55	35	1.41	1.62
15	5	6	2	65	42	1.54	1.68
16	7	9	5	63	36	1.48	1.65
17	7	9	1	71	32	1.48	1.64
18	6	8	3	73	30	1.48	1.63

Na osnovu rezultata upitnika mogu se izvesti sledeći zaključci:

1. Korisnici Encog-a su:
- sa (64%) i bez (36%) teorijskog znanja iz oblasti neuronskih mreža (P1);
- sa (71%) i bez (29%) praktičnog iskustva u rešavanju problema pomoću neuronskih mreža (P2);
- skoro svi (99%) sa iskustvom u programskom jeziku Java (P3).

Na osnovu ovih rezultata može se zaključiti da Encog kao i Neuroph koriste iskusni korisnici i početnici u oblasti neuronskih mreža, sa iskustvom u Java programiranju. Međutim, treba primetiti da je kod Encog-a veći procenat korisnika sa teorijskim znanjem i praktičnim iskustvom u oblasti neuronskih mreža nego kod Neuroph-a, što ukazuje na činjenicu da se početnici u većem procentu odlučuju za Neuroph.

2. U pogledu jednostavnosti ovladavanja softverom, dobijeni su sledeći rezultati:
- softver su brzo naučili (54%) (P5);
- softver ima kvalitetnu dokumentaciju (83%) (P6);
- organizacija i logika softvera su inuitivni (P7 - 66% i P8 - 61%).

Ukoliko se ovi rezultati uporede sa rezultatima istih pitanja za Neuroph, vidi se da veći procenat korisnika (75% Neuroph i 54% Encog) smatra da su Neuroph brzo naučili, i da su njegova organizacija i logika intuitivni (84% Neuroph i 66% Encog), dok oba softvera imaju otprilike isti nivo kvaliteta dokumentacije (85% Neuroph i 83% Encog). Pri tom treba napomenuti da Encog ima više primera datih u vidu programskog koda, dok Neuroph ima više primera datih u vidu wizarda,i sa grafičkim interfejsom u okviru aplikacije Neuroph Studio.

3. U pogledu jednostavnosti korišćenja približno polovina ispitanika (42%) izjasnilo se da je softver
jednostavan za korišćenje i da je intuitivno jasno gde se nalaze koje funkcionalnosti (P9 i P10). Interakciju sa softverom pozitivno je ocenilo 63% ispitanika (P11 i P12).
U poređenju sa Neuroph-om to je nešto manji procenat - 79% (P9 i P10) i 78% (P11 i P12). Ovaj rezultat ukazuje da je Neuroph u pogledu jednostavnosti korišćenja i interakcije sa korisnikom u izvesnoj prednosti u odnosu na Encog-a.

4. U pogledu upotrebne vrednosti ispitanici su se u većini izjasnili da:
- softver odgovara njihovim zahtevima/potrebama (79%) (P13);
- omogućava im da brže obavljaju planirane zadatke (75%) (P14) ;
- ima sve potrebne funkcionalnosti (89%) (P15).

Ovde se primećuje da se u poređenju sa Neuroph-om, veći broj ispitanika (89%) izjasnio da Encog ima sve potrebne funkcionalnosti (dok je kod Neuroph-a 73%) (P15). Ovaj rezultat ukazuje na zahteve korisnika softvera za dodavanjem novih funkcionalnosti koje su potrebne u praksi, iako već sada visok i približno isti procenat korisnika oba softvera, smatra da softver odgovara njihovim zahtevima i omogućava im da brže obavljaju planirane zadatke (P13 i P14).

5. Opšti utisak u vezi Encog-a je pozitivan jer je preko 80% ispitanika reklo da su zadovoljni softverom i da softver radi onako kako oni žele (P16, P17 i P18).

Na osnovu dobijenih rezultata upitnika za Neuroph i Encog mogu se izvesti sledeći zaključci:

1. Encog u većem procentu koriste iskusniji korisnici, dok Neuroph u većem procentu odgovara početnicima (P1, P2);

2. Neuroph je jednostavniji za korišćenje i lakše ga je naučiti nego Encog (P4 – P12);

3. Generalno, oba frameworka odgovaraju potrebama svojih korisnika (P13 i 14);

4. Neuroph-u treba dodati još funkcionalnosti (koje Encog već ima) kako bi se odgovorilo na zahteve korisnika, koji se javljaju u praksi (P15).

7.3. Testiranje performansi (benchmark)

U odeljku 4.4.7 prilikom projektovanja podrške za testiranje performansi, već je objašnjeno da postoje dva nivoa testiranja performansi kada se radi o neuronskim mrežama. Prvi nivo je algoritamski, i podrazumeva utvrđivanje potrebnog broja iteracija da neuronska mreža nauči određeni skup podataka, a drugi nivo je implementacioni koji podrazumeva brzinu, odnosno potrebno vreme da se izvrši jedna iteracija treninga, odnosno performanse određene softverske realizacije algoritama. U ovom poglavlju dati su rezultati testiranja i prikazano je na koji način se vrši testiranje Neuroph framework-a za određenu vrstu neuronskih mreža i nekog problema.

7.3.1 Testiranje performansi algoritama za učenje

U ovom odeljku dati su rezultati testiranja tri verzije Backpropagation algoritma pomoću Neuroph framework-a. U okviru Neuroph framework-a implementirane su tri verzije ovog algoritma:

1) Standardni Backpropagation
2) Backpropagation sa momentum parametrom
3) Resilient Propagation

Testiranje performansi podrazumeva trening neuronske iste mreže za određeni problem sa različitim algoritmima i poređenje broja potrebnih iteracija da mreža nauči određeni skup podataka.
Algoritam je testiran za problem klasifikacije, sa skupom podataka za klasifikaciju cvetova Irisa, već opisanim u odeljku 6.2. Test algoritma se sastoji u tome da se 3 neuronske mreže, sa istim brojem neurona, nezavisno treniraju sa različitim brojem neurona, a zatim se uporede dobijeni rezultati, odnosno potreban broj iteracija za trening. Na listingu 18. Dat je segment programskog koda, koji kreira 3 neuronske mreže sa odgovarajućim podešavanjima i izvršava trening.

```
MultiLayerPerceptron mlp1 =
                     new MultiLayerPerceptron(4, 24, 16, 8, 3);
mlp1.setLearningRule(new BackPropagation());

MultiLayerPerceptron mlp2 =
                     new MultiLayerPerceptron(4, 24, 16, 8, 3);
mlp1.setLearningRule(new MomentumBackpropagation());

MultiLayerPerceptron mlp3 =
                     new MultiLayerPerceptron(4, 24, 16, 8, 3);
mlp3.setLearningRule(new ResilientPropagation());

BackPropagation bp1 = (BackPropagation)mlp1.getLearningRule();
bp1.setLearningRate(0.2);
```

```
bp1.setMaxIterations(30000);
mlp1.learnInSameThread(trainingSet);
MomentumBackpropagation bp2 =
(MomentumBackpropagation)mlp2.getLearningRule();
bp2.setLearningRate(0.2);
bp2.setMomentum(0.5);
bp2.setMaxIterations(30000);
mlp2.learnInSameThread(trainingSet);
ResilientPropagation bp3 =
(ResilientPropagation)mlp3.getLearningRule();
bp3.setMaxIterations(30000);
mlp3.learnInSameThread(trainingSet);
```

Listing 18. Segment programskog koda za testiranje performansi algoritama za učenje

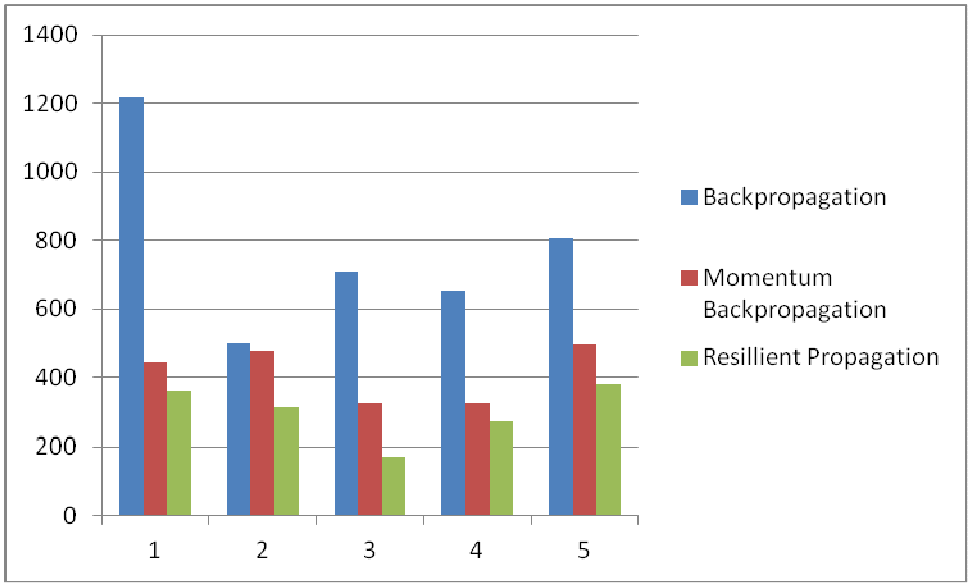
U tabelama 19 a i b dati su rezultati testiranja, za neuronsku mrežu sa (16, 8) neurona u skrivenom sloju (a), i sa (24, 16, 8) neurona u skrivenom sloju (b). Za oba slu;aja trening je izvr[en 5 puta.

Tabela 19a. Broj iteracija potrebnih da neuronska mreža sa dva skrivena sloja sa 16 i 8 neurona nauči klasifikaciju cvetova irisa

	Algoritam za trening		
Rb. treninga	**Standardni Backpropagation**	**Backpropagation with Momentum**	**Resillient Backpropagation**
1	2073	3814	522
2	2325	4116	891
3	2672	3678	285
4	3119	2768	341
5	2512	3061	428
Srednja vrednost	2540	3487	493

Tabela 19b. Broj iteracija potrebnih da neuronska mreža sa tri skrivena sloja sa 24, 16 i 8 neurona nauči klasifikaciju cvetova irisa

	Algoritam za trening		
Rb. treninga	**Standardni Backpropagation**	**Backpropagation with Momentum**	**Resillient Backpropagation**
1	1216	443	363
2	501	478	314
3	710	324	172
4	652	324	273
5	806	495	381
Srednja vrednost	777	413	301



Slika 49.Uporedni grafik potrebnog broj iteracija da neuronska mreža sa tri skrivena sloja sa 24, 16 i 8 neurona nauči klasifikaciju cvetova irisa

Iz datih rezultata testiranja vidi se da algoritam ResilientPropagation predstavlja značajno poboljšanje u odnosu na prethodne postojeće verzije Backpropagation algoritma. Test je izvršen nad dve arhitekture neuronske mreže, 5 puta i Resilient Propagation je u svim slučajevima bio 4-6 puta brži od druga dva (trening je trajao manji broj iteracija). Ovakav rezultat testiranja je bio očekivan, a cilj ovog testa je da pokaže kako se pomoću Neuroph-a mogu kreirati uporedni testovi neuronskih mreža za specifične potrebe. Takođe, efikasna implementacija *ResilientPropagation* algoritma značajna je i sa aspekta konkurentnosti sa Encog framework-om.

7.3.2. Testiranje performansi implementacije

Testiranje performansi implementacije podrazumeva merenje vremena izvršavanja različitih implementacija algoritama za učenje. Kod testiranja performansi implementacije nije potrebno pratiti broj iteracija za koje neuronska mreža nauči neki skup podataka za trening, već brzinu protoka podataka kroz neuronsku mrežu i izračunavanje promena težinskih koreficijenata. Za merenje performansi implementacije korišćen je jednostavan benchmark razvijen u okviru ovog istraživanja.
Kako bi se uporedile performanse Neuroph i Encog framework-a na nivou implementacije kreiran je test performansi koji kroz identične arhitekture neuronskih mreža, propušta slučajno generisani skup podataka za trening zadati broj iteracija. Cilj ovakvog testiranja dakle nije da neuronska mreža nauči skup podataka za trening (što se neće ni desiti), već da se izmeri vreme potrebno za izvršavanje određenog broja iteracija učenja za oba framework-a sa potpuno istim parametrima. Na listingu 19. Dat je segmet programskog koda koji izvršava testiranje.

```
BenchmarkTask task1 = new BenchmarkNeuroph("NeurophBenchmark");
task1.setWarmupIterations(30);
task1.setTestIterations(10);

BenchmarkTask task2 = new BenchmarkEncog("EncogBenchmark");
task2.setWarmupIterations(30);
task2.setTestIterations(10);

Benchmark benchmark = new Benchmark();
benchmark.addTask(task1);
benchmark.addTask(task2);

benchmark.run();
```

Listing 19. Segment programskog koda koji vrši testiranje performansi implementacije algoritama za učnje

U tabeli 20. Dati su parametri sa kojima je izvršeno testiranje.

Arhitektura neuronske mreže:	Višeslojni perceptron sa 4 ulazna neurona 16 skrivenih neurona 3 izlazna neurona
Algoritam za učenje	Backpropagation
Skup podataka za trening:	100 slučajno generisanih parova ulazno-izlaznih vektora sa vrednostima na intervalu [0, 1]
Broj iteracija treninga	10000
Neuroph	v2.6
Encog	V2.5.2

Tabela 20. Parametri sa kojima je izvršeno testiranje performansi implementacije

Samo testiranje performansi vrši se kreiranjem dve neuronske mreže, jedne sa Neuroph i jedne sa
Encog frameworkom, i izvršavanjem algoritma za trening određeni broj iteracija (10000). Procedura kreiranja i treninga mreže za oba frwmeork-a implementirana je u okviru zasebnih klasa `BenchmarkNeuroph i BenchmarkEncog.`
Kao što je već detaljnije objašnjeno u odeljku 4.4.7. ovakav način testiranja pomoću tzv. *microbenchmarka*, zahteva izvršavanje određenog broja iteracija 'zagrevanja' testa, tokom kojih se stabilizuje izvršavanje programa i JVM (Java Virtual Machine).
Stabilizacija podrazumeva izvršavanje optimizacije i HotSpot kompajliranja svih kritičnih metoda, rezervaciju potrebne memorije i stabilizaciju rada Garbage Collector-a.
Do potrebnog broja iteracija i parametara za JVM dolazi se uključivanjem opcija:

-XX:+PrintCompilation - prikaz podataka o kompajliranju
-XX:+PrintGC - prikaz informacij o radu Garbage Collectora
-server – rad JVM u serverskom modu

Nakon nekoliko probnih testova pokazalo se da je 10000 iteracija treninga kroz 30 iteracija zagrevanja, i 10 iteracija testiranja daje stabilne rezultate koji ne variraju.

U tabeli 21 data je specifikacija platforme na kojoj je izvršeno testiranje:

CPU	Intel i5 M 450 @ 2.40GHz
Memory	4GB (3.86 GB usable)
OS	Windows 7 Professional 64-bit
JVM	Java v6 update 23 (build 1.6.0_23-b05)

Tabela 21. Specifikacija platforme na kojoj je izvršeno testiranje

U tabel 22 dati su rezultati testiranja performansi implementacija za Neuroph i Encog framework za 10 test iteracija treninga. Svaka test iteracija podrazumeva izvršavanje 10000 iteracija treninga neuronske mreže sa skup za trening od 100 elemenata.

	Neuroph (ms)	**Encog** (ms)
1.	10819	9721
2.	10744	9205
3.	10719	9750
4.	10627	9288
5.	10470	10087
6.	10530	9559
7.	10738	9612
8.	10593	9693
9.	10560	10437
10.	10654	9796
Min	10470	9205
Max	10819	10437
Avg	10645.4	9714.8

Tabela 22. Rezultati testiranja performansi

Dobijeni rezultati pokazuju da najnovija verzija Neuroph framework-a 2.6 svega 10% zaostaje u pogledu performansi u odnosu na Encog framework, verziju 2.5.2. Ovaj rezultat predstavlja značajan napredak jer je na testovima prethodnih verzija Encog bio više nego duplo brži.
Ipak treba naglasiti da su ovi testovi rađeni samo sa jednim jezgrom, kako bi se izvršilo adekvatno poređenje, jer Neuroph ne podržava paralelno izvršavanje na više procesorskih jezgara istovremeno.
U slučaju da se u okviru Encog-a aktivira ta opcija, sigurno bi razlika bila veća u korist Encog-a.
Zbog toga, procesiranje na više jezgara istovremeno je jedan od daljih pravaca razvoja za Neuroph. Međutim, i trenutno prikazan napredak u performansama implementacije predstavlja veliki korak, a posebno je značajno kreiranje podrške za testiranje performansi u okviru Neuroph-a, čime se omogućava sistematsko praćenje i optimizacija performansi tokom daljeg razvoja

8. ZAKLJUČAK

8.1. Osnovni doprinos

Na primeru razvoja dodatnih funkcionalnosti softverskog framework-a za neuronske mreže Neuroph, prikazan je proces planiranja i razvoja softverskog framework-a otvorenog koda iz oblasti inteligentnih sistema. Konkretan doprinos obuhvata:

1. Razvijene su dodatne funkcionalnosti softverskog framework-a za neuronske mreže Neuroph, koje učvršćuju njegovu poziciju vodećeg svetskog framework-a otvorenog koda u toj oblasti, i otvaraju mogućnosti za njegov dalji razvoj.

2. Po uzoru na Neuroph, kreiran je opšti model softverskog framework-a iz oblasti inteligentnih sistema, i definisani su principi projektovanja strukture framework-a i programskog interfejsa (API). Definisani model i principi projektovanja potvrđeni su u praksi, i mogu se primenjivati za projektovanje raznih drugih softverskih framework-a iz oblasti inteligentnih sistema.

3. Kroz razvoj Neuroph framework-a, definisan je ceo proces planiranja, projektovanja, implementacije i evaluacije softverskog framework-a iz oblasti inteligentnih sistema. Dakle pored konkretnih tehničkih preporuka u pogledu projektovanja softvera i implementacije, definisan je ceo proces razvoja softvera otvorenog koda, od nastanka ideje do objavljivanja uspešnog softverskog proizvoda u oblasti inteligentnih sistema.

8.2. Mogućnosti primene

Mogućnosti primene rezultata ovog istraživanja obuhvataju:

1. primenu Neuroph framework-a za probleme za koje se koriste neuronske mreže;
2. korišćenje Neuroph framework-a kao platforme za istraživanja u oblasti neuronskih mreža;
3. korišćenje Neuroph framework-a kao edukativne alatke, u okviru univerzitetskih kurseva na kojima se uči o neuronskim mrežama;
4. primena datih preporuka za razvoj softverskih framework-a, za razvoj framework-a i u drugim oblastima inteligentnih sistema.

Unapređena verzija Neuroph framework-a razvijena u okviru ovog istraživanja može se koristiti za rešavanje svih problema za koje su koriste neuronske mreže. Tipični problemi su: klasifikacija, prepoznavanje, klasterizacija, aproksimacija, predviđanje i sl. Zahvaljujući novim algoritmima za učenje i dodatnim funkcionalnostima koji olakšavaju primenu, unapređena verzija Neuroph framework-a, ima značajno bolje performanse nego prethodne verzije, i jednostavnije se koristi.

Zahvaljujući velikom broju funkcionalnost, fleksibilnosti i mogućnostima za proširenje Neuroph se može koristit kao istraživačka platforma za neuronske mreža. Neuroph kao istraživačka platforma omogućava eksperimentisanje sa postojećim i kreiranje novih vrsta neuronskih mreža i algoritama za učenje, kao i poređenje performansi različitih implementacija neuronskih mreža.
Zahvaljujući razumljivoj strukturi i jednostavnosti korišćenja Neuroph je odličan za primenu u obrazovanju, jer jasno demonstrira osnovne principe rada neuronskih mreža. Zahvaljujjući jednostavnim grafičkim alatkama zasnovanim na sistemu *wizard-a*, studenti su u stanju da primenjuju neuronske mreže za konkretne probleme i pre nego što u potpunosti savladaju sve teorijske aspekte, što ih u značajnoj meri motiviše i olakšava im učenje. S druge strane, raspoloživost kompletnog programskog koda omogućava detaljnu analizu algoritama za učenje na način koji je nije moguć sa knjigama i drugim klasičnim materijalima za učenje.
Iskustva iz prakse vezana za projektovanje, implementaciju i vođenje razvoja softverskog framework-a koja su data kao rezultat ovog istraživanja, mogu se primeniti za razvoj uspešnih softverskih framework-a u raznim drugim oblastima inteligentnih sistema (npr. genetski algoritmi, semantički web, sistemi agenata). Pri tom naravno u skladu sa preporukama, treba poći od već postojećih rešenja, sagledati ih u kontekstu preporuka datih u okviru ovog istraživanja i razmotriti mogućnosti za kreiranjem nadogradnje postojećih rešenja. Takođe, bez obzira što možda već postoje brojna i več dobro poznata rešenja za određene oblasti, nova rešenja koja donose novi kvalitet i bolji pristup uvek mogu prevladati, što najbolje pokazuje Neuroph. Takođe treba imati u vidu da je uspešan razvoj i plasman novog softverskog framework-a otvorenog koda, proces za koji je potrebno nekoliko godina.

8.3. Pravci daljeg istraživanja i razvoja

Dalji razvoj Neuroph framework-a obuhvatiće alate koji omogućavaju primenu neuronskih mreža za konkretne probleme, ali i nadogradnju u pogledu mogućnosti za istraživanja u oblasti neuronskih mreža. Aktuelne ideje za dalji razvoj su:

- razvoj podrške za paralelno procesiranje;
- razvoj generičke podrške za ceo proces treniranja i testiranja neuronskih mreža;
- razvoj specijalizovanih alata za određene probleme (npr. wizard-i za klasifikaciju i klasterizaciju);
- vizuelizacija rada i rezultata neuronskih mreža: grafici greške i aktivacije za svaki neuron pojedinačno, grafik promene težina tokom učenja, histogram težina;
- kompletna podrška za sve funkcionalnosti framework-a u okviru razvojnog okruženja Neuroph Studio;
- razvoj vizuelnog alata za rad sa neuronskim mrežama koji obezbeđuje paletu komponenti i omogućava kreiranje neuronskih mreža prevlačenjem komponenti iz palete (po istom principu kao i GUI editori) .

Dalja istraživanja u vezi primene metoda softverskog inženjerstva za razvoj softvera otvorenog koda u oblasti inteligentnih sistema obuhvatiće primenu modela razvoja datog u ovom istraživanju za razvoj drugih framework-a u oblasti inteligentnih sistema.

LITERATURA

[Beck, K., Andres, C., 2004], Extreme Programming Explained: Embrace Change, Addison Wesley, ISBN 978-0321278654

[Goetz, B., 2004], Java theory and practice: Dynamic compilation and performance measurement, IBM developer Works, ONLINE

[Castellano, G., Fanelli, A., M., Pelillo, M., 1997], An Iterative Pruning Algorithm for Feedforward Neural Networks, IEEE Transactions on Neural Networks, vol. 8, no. 3, IEEE Computational Intelligence Society

[Carpenter, G.A., Grossberg, S., 2003], Adaptive Resonance Theory, In Michael A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, Second Edition (pp. 87-90). Cambridge, MA: MIT Press

[Fahlman, S.E., Hinton, G.E. and Sejnowski, T.J. 1983], Massively parallel architectures for A.I.: Netl, Thistle, and Boltzmann machines. Proceedings of the National Conference on Artificial Intelligence, Washington DC.

[Feller, J. 2001], Understanding Open Source Software Development, Addison-Wesley Professional, ISBN 978-0201734966

[Fogel, K., 2005], Producing Open Source Software: How to Run a Successful Free Software Project, O'Reilly Media, ISBN 978-0596007591

[Happel, Bart and Murre, Jacob, 1994], The Design and Evolution of Modular Neural Network Architectures. Neural Networks, 7: 985-1004.

[Hebb, D., O., 2002], Organization of behavior, Psychology Press, ISBN 978-0805843002

[Hecht-Nielsen, R. 1987], Counterpropagation networks, Applied Optics, Vol. 26, Issue 23, pp. 4979-4983, Optical Society of America.

[Hopfield, J., J. 1982], Neural networks and physical systems with emergent collective computational abilities, Proceedings of the National Academy of Sciences of the USA, vol. 79 no. 8 pp. 2554-2558, April 1982.

[Jacobson, I., Booch, G., Rumbaugh, J., 1999], The Unified Software Development Process, Addison-Wesley Professional, ISBN 978-0201571691

[Kosko, B., 1988] Bidirectional Associative Memories, IEEE Transactions on Systems, Man and Cybernetics, vol 18, no. 1, 1988.

[Larman, C., 2003], Agile and Iterative Development: A Manager's Guide, Addison-Wesley Professional, ISBN: 978-0131111554

[Laurent, A., 2004], Understanding Open Source and Free Software Licensing, O'Reilly Media, ISBN 978-0596005818

[McConnell, S., 1996], Rapid Development: Taming Wild Software Schedules, Microsoft Press, ISBN 978-1556159008

[Nguyen, D., Widrow, B. 1990], Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. Proceedings of the International Joint Conference on Neural Networks, 3:21–26, 1990.

[Oja, E., 1989], Neural networks, principal components and subspaces, International Journal of Neural Systems, Vol. 1, pp. 61-68, ISSN: 0129-0657, World Scientific Publishing.

[Riehle. D., 2000] Framework Design: A Role Modeling Approach. Ph.D. Thesis, No. 13509. Zürich,
Switzerland, ETH Zürich, 2000.

[Riedmiller, M., 1994], Rprop - Description and Implementation Details, Technical report.

[Rumelhart, D. E., Hinton, G., E., Williams, R. J., 1986], Learning representations by back-propagating errors, Nature 23 (6088): 533–536. doi:10.1038/323533a0

[Sevarac, Z., 2006], Neuro Fuzzy Reasoner for Student Modeling, Proceedings of Sixth International Conference on Advanced Learning Technologies 2006, 740 - 744, ISBN: 0-7695-2632-2

[Thayer, R., Dorfman, M., Garr, D., 2002] Software Engineering: Volume 1: The Development Process, Wiley-IEEE Computer Society, ISBN 978-0769515557

[Xiaoping, J., 1999] Object-Oriented Software Development Using Java: Principles, Patterns, and Frameworks, Addison Wesley Longman; 1st edition, 1999., ISBN 020135084X

PRILOG 1. Uporedni pregled karakteristika framework-a za neuronske mreže

U prilogu je dat detaljan uporedni pregled karakteristika tri framework-a za neuronske mreže – JOONE, Encog i Neuroph. Za sva tri framework-a tabelarno su prikazane sledeće karakteristike:

1. vrste neuronskih mreža;
2. algoritmi za učenje;
3. funkcije transfera;
4. tehnike generisanja slučajnih brojeva;
5. formati ulaznih podataka;
6. formati izlaznih podataka;
7. dodatne funkcionalosti karakteristike.

Napomena: Sa znakom X su označene funkcionalnosti Neuroph-a koje su razvijene u okviru ovog istraživanja za verziju 2.6, dok su ostale funkcionalnosti postojale i u verziji 2.5.

Vrste neuronskih mreža			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Adaline	+	+	
Perceptron	+	+	+
Multi Layer Perceptron	+	+	+
Hebbian Network	+	+	
Hopfield neural network	+	+	
Bidirectional Associate memory (BAM)	+	+	
Boltzmann machine	X	+	
Counterproragation neural network (CPN)	X	+	
Recurrent-Elman		+	+
Recurrent-Jordan		+	+
Recurrent-SOM		+	+
RBF network	+	+	
Maxnet	+		
Competitive Network	+		
Kohonen SOM	+	+	+
Instar	+		
Outstar	+		
Adaptive resonance theory (ART1)		+	
Time Delay neural network			+
Neuro Fuzzy Perceptron	+		
Support Vector Machine (SVM)		+	
PCA neural network	X		+

Tabela 1. Uporedna tabela podržanih vrsta neuronskih mreža

Algoritmi za učenje			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
LMS	+	+	
PerceptronLearning	+		
Binary delta rule	+		
Smooth Delta Rule	+	+	
Hebbian learning (supervised and unsupervised)	+	+	
Simulated Annealing		+	
Backpropagation	+	+	+
Auto Backpropagation	+	+	
Resilient backpropagation	X	+	+
Manhattan Update Rule Propagation		+	
Levenberg Marquardt (LMA)		+	
Scaled Conjugate Gradient		+	
Competitive learning	+	+	
Hopfield learning	+	+	
Kohonen	+	+	+
Instar learning	+	+	
Outstar learning	+	+	
Instar/Outstar	X	+	
Genetic algorithm training		+	
NEAT	+	+	

Tabela 2. Uporedni pregled podržanih algoritama za učenje neuronskih mreža

Funkcije transfera			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Step	+	+	
Bipolar (Sgn)	+	+	
Sigmoid	+	+	+
TanH	+	+	+
Linear	+	+	+
SoftMax		+	+
Sin wave	X	+	+
Gaussian	+	+	+
Logarithmic	X	+	+
Delay			+
Context function			+
Ramp	+	+	
Trapezoid	+		

Tabela 3. Uporedni pregled podržanih funkcija transfera

Tehnike za generisanje slučajnih brojeva			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Range	+	+	+
Gaussian	X	+	
Fan-In		+	+
Nguyen-Widrow	X	+	
Const		+	
Distort	X	+	
Consistent		+	

Tabela 4. Uporedni pregled podržanih tehnika za generisanje slučajnih brojeva

Podržani formati ulaznih podataka za trening			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Txt	+	+	+
Slika	+	+	+
CSV	+	+	
JDBC	X	+	+
XML		+	
URL	X		+
Yahoo finance input			+
Buffered input		+	
Stream	X		+
Java nizovi	+	+	

Tabela 5. Uporedni pregled podržanih formata podataka za trening (učenje)

Podržani formati izlaza			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Fajl	X		+
Slika			+
Stream	X		+
JDBC	X		+
Java nizovi	+	+	+

Tabela 6. Uporedni pregled formata podržanih izlaznih podataka

Dodatne funkcionalnosti i karakteristike			
	Neuroph 2.6	**Encog 2.5**	**Joone 2.0**
Normalizacija podataka	X	+	
Benchmark	X	+	
Pruning		+	
Modularne neuronske mreže		+	+
Podrška za skript jezik		+	+
Automatsko generisanje Java programskog koda za neuronske mreže		+	
Multithreaded/Višenitno treniranje		+	+
GPU/OpenCl		+	
Platforme	Java/.Net	Java/.Net/Silverlight	Java
Godina objavljivanja prve verzije	2008	2008	2001
Godina objavljivanja poslednje verzije	2012	2011	2007

Tabela 7. Uporedni pregled dodatnih funkcionalnosti i karakteristika

PRILOG 2. Primeri korišćenja framework-a za neuronske mreže

Primer korišćenja JOONE framework-a

Dat je segment koda kao primer kreiranja neuronske mreže tipa Multi Layer Perceptron za XOR problem pomoću JOONE framework-a.

```
// First, create three Layers
LinearLayer input = new LinearLayer();
SigmoidLayer hidden = new SigmoidLayer();
SigmoidLayer output = new SigmoidLayer();

// sets their dimensions
input.setRows(2);
hidden.setRows(3);
output.setRows(1);

// Now create the two Synapses
/* input -> hidden conn. */
FullSynapse synapse_IH = new FullSynapse();
/* hidden -> output conn. */
FullSynapse synapse_HO = new FullSynapse();
// Connect the input layer whit the hidden layer
input.addOutputSynapse(synapse_IH);
hidden.addInputSynapse(synapse_IH);

// Connect the hidden layer whit the output layer
hidden.addOutputSynapse(synapse_HO);
output.addInputSynapse(synapse_HO);

MemoryInputSynapse  inputStream =  new MemoryInputSynapse();

// The first two columns contain the input values
inputStream.setInputArray(XOR_INPUT);
inputStream.setAdvancedColumnSelector("1,2");

// set the input data
input.addInputSynapse(inputStream);

TeachingSynapse trainer = new TeachingSynapse();

// Setting of the file containing the desired responses
// provided by a FileInputSynapse
MemoryInputSynapse samples = new MemoryInputSynapse();


// The output values are on the third column of the file
samples.setInputArray(XOR_IDEAL);
samples.setAdvancedColumnSelector("1");
```

```
trainer.setDesired(samples);

// Connects the Teacher to the last layer of the net
output.addOutputSynapse(trainer);

// Creates a new NeuralNet
NeuralNet nnet = new NeuralNet();

// All the layers must be inserted in the NeuralNet object
nnet.addLayer(input, NeuralNet.INPUT_LAYER);
nnet.addLayer(hidden, NeuralNet.HIDDEN_LAYER);
Monitor monitor = nnet.getMonitor();
// # of rows (patterns) contained in the input file
monitor.setTrainingPatterns(4);
monitor.setTotCicles(5000);
monitor.setLearningRate(0.7);
monitor.setMomentum(0.3);
monitor.setLearning(true);  // The net must be trained
nnet.go(); //starts the training job
```

Primer korišćenja Encog framework-a

Dat je segment koda kao primer kreiranja neuronske mreže tipa Multi Layer Perceptron za XOR problem pomoću Encog framework-a.

```
double XOR_INPUT[][] = { { 0.0, 0.0 }, { 1.0, 0.0 },
                         { 0.0, 1.0 }, { 1.0, 1.0 } };
double XOR_IDEAL[][] = { { 0.0 }, { 1.0 }, { 1.0 }, { 0.0 } };

BasicNetwork network = new BasicNetwork();
network.addLayer(new BasicLayer(2));
network.addLayer(new BasicLayer(2));
network.addLayer(new BasicLayer(1));
network.getStructure().finalizeStructure();
network.reset();

NeuralDataSet trainingSet =
                  new BasicNeuralDataSet(XOR_INPUT, XOR_IDEAL);
// train the neural network
ResilientPropagation train =
      new ResilientPropagation(network, trainingSet);

int epoch = 1;
      do {
            train.iteration();
            System.out.println("Epoch #" + epoch + " Error:" +
                                    train.getError());
            epoch++;
      } while(train.getError() > 0.01);
```

Primer korišćenja Neuroph framework-a

Dat je segment koda kao primer kreiranja neuronske mreže tipa Multi Layer Perceptron za XOR problem pomoću Neuroph framework-a.

```
// create training set (logical XOR function)
TrainingSet trainingSet = new TrainingSet(2, 1);
trainingSet.addElement(new SupervisedTrainingElement(
                new double[]{0, 0}, new double[]{0}));
trainingSet.addElement(new SupervisedTrainingElement(
                new double[]{0, 1}, new double[]{1}));
trainingSet.addElement(new SupervisedTrainingElement(
                new double[]{1, 0}, new double[]{1}));
trainingSet.addElement(new SupervisedTrainingElement(
                new double[]{1, 1}, new double[]{0}));

// create multi layer perceptron
MultiLayerPerceptron myMlPerceptron = new
       MultiLayerPerceptron(TransferFunctionType.TANH, 2, 3, 1);

// learn the training set
myMlPerceptron.learnInSameThread(trainingSet);
```

PRILOG 3. Detalji implementacije

Listing 3.1. Programski kod funkcije transfera Log

```
public class Log extends TransferFunction {

    @Override
    public double getOutput(double net) {
        return Math.log(net);
    }

   @Override
    public double getDerivative(double net) {
      return (1/net);
    }
}
```

Listing 3.2. Programski kod klase koja implementira Max normalizaciju

```
public class MaxNormalizer implements Normalizer {
   double[] maxVector; // contains max values for all columns

   public void normalize(TrainingSet<? extends TrainingElement>

trainingSet) {
     findMaxVector(trainingSet);
     for (TrainingElement trainingElement :
trainingSet.elements()) {
            double[] input = trainingElement.getInput();
            double[] normalizedInput = normalizeMax(input);
            trainingElement.setInput(normalizedInput);
     }
   }

   private void findMaxVector(TrainingSet<? extends
TrainingElement> trainingSet) {
        int inputSize = trainingSet.getInputSize();
        maxVector = new double[inputSize];
        for (TrainingElement te : trainingSet.elements()) {
            double[] input = te.getInput();
            for (int i = 0; i < inputSize; i++) {
                if (Math.abs(input[i]) > maxVector[i]) {
                    maxVector[i] = Math.abs(input[i]);
                }
            }
        }
    }

    public double[] normalizeMax(double[] vector) {
        double[] normalizedVector = new double[vector.length];
        for(int i = 0; i < vector.length; i++) {
            normalizedVector[i] = vector[i] / maxVector[i];
        }
        return normalizedVector;
    }
}
```

Listing 3.3. Klasa za generisanje slučajnih vrednosti težinskih koeficijenata na zadatom intervalu

```
public class RangeRandomizer extends WeightsRandomizer {
    protected double min;
    protected double max;

    public RangeRandomizer(double min, double max) {
        this.min = min;
        this.max = max;
    }

    @Override
    protected double nextRandomWeight() {
      return min + randomGenerator.nextDouble () * (max - min);
    }
}
```

Listing 3.4. Klasa za promenu postojećih težinskh koeficijenata prema slučajanom faktoru

```
public class DistortRandomizer extends WeightsRandomizer {
    double distortionFactor;

    public DistortRandomizer(double distortionFactor) {
        this.distortionFactor = distortionFactor;
    }

    @Override
    public void randomize(NeuralNetwork neuralNetwork) {
        this.neuralNetwork = neuralNetwork;
        for (Layer layer : neuralNetwork.getLayers())
            for (Neuron neuron : layer.getNeurons())
   for (Connection connection : neuron.getInputConnections()) {
            double weight = connection.getWeight().getValue();
       connection.getWeight().setValue(distortWeight(weight));
      }
    }

    private double distortWeight(double weight) {
        return  weight + (this.distortionFactor -
(randomGenerator.nextDouble() * this.distortionFactor * 2));
    }
}
```

Listing 3.5. Klasa za generisanje slučajnih težinskih koeficijenata *Nguyen-Widrow* metodom

```
public class NguyenWidrowRandomizer extends RangeRandomizer {

    public NguyenWidrowRandomizer(double min, double max) {
        super(min, max);
    }

    @Override
    public void randomize(NeuralNetwork neuralNetwork) {
        super.randomize(neuralNetwork);
        int inputNeuronsCount =
neuralNetwork.getInputNeurons().size();
        int hiddenNeuronsCount = 0;
 for (int i = 1; i < neuralNetwork.getLayersCount() - 1; i++) {
        hiddenNeuronsCount +=
                 neuralNetwork.getLayerAt(i).getNeuronsCount();
        }
        double beta = 0.7 * Math.pow(hiddenNeuronsCount, 1.0 /
                      inputNeuronsCount);
        for (Layer layer : neuralNetwork.getLayers()) {
            double norm = 0.0;
            for (Neuron neuron : layer.getNeurons()) {
   for (Connection connection : neuron.getInputConnections()) {
                 double weight = connection.getWeight().getValue();
                        norm += weight * weight;
                    }
            }
            norm = Math.sqrt(norm);
   for (Neuron neuron : layer.getNeurons()) {
   for (Connection connection : neuron.getInputConnections()) {
                 double weight = connection.getWeight().getValue();
                 weight = beta * weight / norm;
                 connection.getWeight().setValue(weight);
                    }
            }
        }
    }
}
```

Listing 3.6. Ulazni adapter za čitanje ulaza za neuransku mrežu iz ulaznih *stream-ova*

```
public class InputStreamAdapter implements InputAdapter {
    protected BufferedReader bufferedReader;

    public InputStreamAdapter(InputStream inputStream) {
        bufferedReader = new BufferedReader(
                            new InputStreamReader(inputStream));
    }

    public InputStreamAdapter(BufferedReader bufferedReader) {
        this.bufferedReader = bufferedReader;
    }

public double[] readInput() {
        try {
            String inputLine = bufferedReader.readLine();
            if (inputLine != null) {
               double[] inputBuffer =

VectorParser.parseDoubleArray(inputLine);
               return inputBuffer; }
             return null;
        } catch (IOException ex) {
  throw new NeurophInputException("Error reading input from
stream!", ex); }
    }

    public void close() {
      try { bufferedReader.close() }
        catch (IOException ex) {
         throw new NeurophInputException("Error closing
stream!", ex);
    }}
}
```

Listing 3.7. Klasa URLInputAdapter koja implementira ulazni adapter za čitanje sa URL-a

```
public class URLInputAdapter extends InputStreamAdapter {

    public URLInputAdapter (URL url) throws IOException {
        super(new BufferedReader( new InputStreamReader(
url.openStream() )));
    }
    public URLInputAdapter (String url) throws
MalformedURLException, IOException  {
        this(new URL(url));
    }
}
```

Listing 3.8. Klasa *JDBCInputAdapter* koja implementira ulazni adapter za čitanje ulaza za neuronsku mrežu iz baze podataka

```
public class JDBCInputAdapter implements InputAdapter {
    Connection connection;
    ResultSet resultSet;
    int inputSize;

    public JDBCInputAdapter (Connection connection, String sql)
{
        this.connection = connection;
        try {
            Statement stmt = connection.createStatement();
            resultSet = stmt.executeQuery(sql);
            ResultSetMetaData rsmd = resultSet.getMetaData();
            inputSize = rsmd.getColumnCount();
        } catch (SQLException ex) {
            throw new NeurophInputException("Error executing
query at JdbcInputAdapter", ex);
        }
    }

    public double[] readInput() {
        try {
            while (resultSet.next()) {
                double[] inputBuffer = new double[inputSize];
                for (int i = 1; i <= inputSize; i++) {
                   inputBuffer[i - 1] = resultSet.getDouble(i);
                }
                return inputBuffer;
            }
        } catch (SQLException ex) {
            throw new NeurophInputException(
                "Error reading input value from the result
set!", ex);
        }
        return null;
    }

    public void close() {
        try {
            resultSet.close();
        } catch (SQLException ex) {
            throw new NeurophInputException(
                           "Error closing database
connection!", ex);
        }
    }
}
```

Listing 3.9. Izlazni adapter za čitanje iz izlaznih *stream-ova*

```
public class OutputStreamAdapter implements OutputAdapter {
    protected BufferedWriter bufferedWriter;

    public OutputStreamAdapter(OutputStream outputStream) {
        bufferedWriter = new BufferedWriter(
                                new
OutputStreamWriter(outputStream));
    }

   public OutputStreamAdapter(BufferedWriter bufferedWriter) {
        this.bufferedWriter = bufferedWriter;
    }

    public void writeOutput(double[] output) {
        try {
            String outputLine = "";
            for (int i = 0; i < output.length; i++) {
                outputLine += output[i] + " ";
            }
            outputLine += "\r\n";
            bufferedWriter.write(outputLine);
        } catch (IOException ex) {
            throw new NeurophOutputException(
                                "Error writing output to
stream!", ex);
        }
    }

    public void close() {
        try {
            bufferedWriter.close();
        } catch (IOException ex) {
            throw new NeurophOutputException(
                                   "Error closing output
stream!", ex);
        }
    }
}
```

Listing 3.10. Klasa *URLOutputAdapter* koja implementira izlazni adapter za upisivanje na URL

```
public class URLOutputAdapter extends OutputStreamAdapter {

   public URLOutputAdapter(URL url) throws IOException {
        super(new BufferedWriter(new
OutputStreamWriter(url.openConnection().getOutputStream())));
    }

    public URLOutputAdapter(String url) throws
MalformedURLException, IOException {
        this(new URL(url));
    }
}
```

Listing 3.11. Klasa *JDBCOutputAdapter* koja implementira izlazni adapter za upis izlaza neuronske mreže u bazu podataka

```
public class JDBCOutputAdapter implements OutputAdapter {
    Connection connection;
    String tableName;

    public JDBCOutputAdapter(Connection connection, String
tableName){
        this.connection = connection;
        this.tableName = tableName;
    }

    public void writeOutput(double[] output) {
        try {
            String sql = "INSERT " + tableName + " VALUES(";
            for (int i = 0; i < output.length; i++) {
                sql += output[i];
                if (i < (output.length - 1)) {
                    sql = ", ";
                }
            }
            sql += ")";
            Statement stmt = connection.createStatement();
            stmt.executeUpdate(sql);
            stmt.close();
        } catch (SQLException ex) {
Logger.getLogger(JDBCOutputAdapter.class.getName()).log(Level.SE
VERE, null, ex);
        }
    }
    public void close() {    }
}
```

Listing 3.12. Programski kod klase Stopwatch koja predstavlja štopericu za merenje vremena

```
public class Stopwatch {

    private long startTime = -1;
    private long stopTime = -1;
    private boolean running = false;

    public Stopwatch() {
    }

    public void start() {
        startTime = System.currentTimeMillis();
        running = true;
    }

    public void stop() {
        stopTime = System.currentTimeMillis();
        running = false;
    }

    public long getElapsedTime() {
        if (startTime == -1) {
            return 0;
        }
        if (running) {
            return System.currentTimeMillis() - startTime;
        } else {
            return stopTime - startTime;
        }
    }

    public void reset() {
        startTime = -1;
        stopTime = -1;
        running = false;
    }
}
```

Listing 3.13. Klasa *BenchmarkTask* koja predstavlja apstraktnu osnovnu klasu za testove performansi

```
abstract public class BenchmarkTask {
    private String name;
    private int warmupIterations=1;
    private int testIterations=1;

    public BenchmarkTask(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public int getTestIterations() {
        return testIterations;
    }

    public void setTestIterations(int testIterations) {
        this.testIterations = testIterations;
    }

    public int getWarmupIterations() {
        return warmupIterations;
    }

    public void setWarmupIterations(int warmupIterations) {
        this.warmupIterations = warmupIterations;
    }

    abstract public void prepareTest();
    abstract public void runTest();
}
```

Listing 3.14. Klasa *Benchmark* sadrži osnovnu logiku toka izvršavanja testova

```
public class Benchmark {
    ArrayList<BenchmarkTask> tasks;

    public Benchmark() {
        tasks = new ArrayList<BenchmarkTask>();
    }

    public void addTask(BenchmarkTask task) {
        tasks.add(task);
    }

    public static void runTask(BenchmarkTask task) {
        System.out.println("Preparing task " + task.getName());
        task.prepareTest();

        System.out.println("Warming up " + task.getName());
        for (int i = 0; i < task.getWarmupIterations(); i++) {
            task.runTest();
        }

        System.out.println("Runing " + task.getName());

        Stopwatch timer = new Stopwatch();
        BenchmarkTaskResults        results        =        new
BenchmarkTaskResults(task.getTestIterations());

        for (int i = 0; i < task.getTestIterations(); i++) {
            timer.reset();
            timer.start();
            task.runTest();
            timer.stop();
            results.addElapsedTime(timer.getElapsedTime());
            System.gc();
        }

        results.calculateStatistics();
        System.out.println(task.getName() + " results");
        System.out.println(results); // could be sent to file
    }

    public void run() {
        for(int i=0; i < tasks.size(); i++)
            runTask(tasks.get(i));
    }
}
```

BIOGRAFIJA AUTORA

Zoran Ševarac rođen je 18.01.1977. godine u Beogradu, gde je završio osnovnu i srednju školu. Studije na Fakultetu organizacionih nauka započeo je školske 1997/98. godine. Diplomirao je 2004. godine, na smeru za Informacione sisteme, odbranom diplomskog rada iz oblasti veštačke inteligencije na temu „Aplikacioni okvir za razvoj neuronskih mreža".

Školske 2004/05 godine upisuje magistarske studije na Fakultetu organizacionih nauka, na smeru za Informacione sisteme. Magistarsku tezu pod naslovom „Hibridni inteligentni agent" odbranio je 2009. godine. Mentor za diplomski rad i na magistarskim studijama bio je prof. dr. Vladan Devedžić.

Od 2005. godine radi na FON-u kao saradnik na istraživačkim projektima u Laboratoriji za veštačku inteligenciju, od 2008. kao saradnik u nastavi, a od 2009. kao asistent na katedri za softversko inženjerstvo.

Kao saradnik u nastavi i asistent, radio je na predmetima Principi programiranja i Inteligentni sistemi. Na predmetu Principi programiranja držao je laboratorijske vežbe iz programskog jezika Java, a na predmetu Inteligentni sistemi iz neuronskih mreža i inteligentnih agenata.

Najznačajniji rezultat dosadašnjeg rada predstavlja softverski *framework* za razvoj neuronskih mreža *Neuroph*, koji je nastao kao rezultat višegodišnjeg istraživanja, , i trenutno je vodeći softver otvorenog koda u oblasti neuronskih mreža u svetu.

Kao istaknuti član zajednice otvorenog koda, i promoter NetBeans projekta 2011. postao je član upravnog odbora i glavnog tima NetBeans projekta http://www.netbeans.org (NetBeans Dream Team). NetBeans projekat je jedan od vodećih razvojnih okruženja za Java-u i razvija se u okviru kompanije Oracle.

Autor je više stručnih radova na domaćim i stranim konferencijama i u časopisima.

Прилог 1.

Изјава о ауторству

Потписани-а Зоран Шеварац

број уписа 485/2009

Изјављујем

да је докторска дисертација под насловом

Софтверско инжењерство интелигентних система

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 28.03.2012.

Зоран Шеварац

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Зоран Шеварац

Број уписа 485/2009

Студијски програм _____

Наслов рада Софтверско инжењерство интелигентних система

Ментор проф. др Владан Девеџић

Потписани Зоран Шеварац

изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 28.03.2012.

Зоран Шеварац

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

СОФТВЕРСКО ИНЖЕЊЕРСТВО ИНТЕЛИГЕНТНИХ СИСТЕМА

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 23.01.2014.

Зоран Ж. Дарић