UNIVERZITET CRNE GORE
PRIRODNO-MATEMATIČKI FAKULTET

Goran Šuković

GENERISANJE TOPOLOGIJE NEURONSKE MREŽE PRIMJENOM SISTEMA LINDENMAJERA I GENETSKIH ALGORITAMA

DOKTORSKA DISERTACIJA

Podgorica, 2014. god.

PODACI I INFORMACIJE O DOKTORANTU

Ime i prezime: **Mr Goran Šuković**
Datum i mjesto rođenja: **06.12.1966. godine, Zemun**
Naziv završenog postdiplomskog studijskog programa i godina završetka: **Magistarske studije na Matematičkom fakultetu Univerziteta u Beogradu, smer Računarstvo računarske nauke, 1998.**

INFORMACIJE O DOKTORSKOJ DISERTACIJI
Naziv doktorskih studija: **Doktorski studijski program Računarske nauke**
Naslov teze: **Generisanje topologije neuronske mreže primjenom sistema Lindenmajera i genetskih algoritama**
Fakultet na kojem je disertacija odbranjena: **Prirodno-matematički fakultet, Univerzitet Crne Gore**

UDK, OCJENA I ODBRANA DOKTORSKE DISERTACIJE
Datum prijave doktorske teze: **18.12.2008. godine**
Datum sjednice Senata Univerziteta na kojoj je prihvaćena teza: **27.01.2009. godine**
Komisija za ocjenu podobnosti teze i kandidata:

Dr Dragan Mašulović, vanredni profesor PMF-a u Novom Sadu
Dr Miroslav Vesković, redovni profesor PMF-a u Novom Sadu
Dr Zora Konjović, redovni profesor FTN-a u Novom Sadu
Dr Predrag Stanišić, redovni profesor PMF-a u Podgorici

Mentor:

Dr Milo Tomašević, vanredni profesor ETF-a u Beogradu

Komisija za ocjenu doktorske disertacije:

Dr Milo Tomašević, vanredni profesor ETF-a u Beogradu
Dr Predrag Stanišić, redovni profesor PMF-a u Podgorici
Dr Milenko Mosurović, vanredni profesor PMF-a u Podgorici
Dr Srđan Kadić, docent PMF-a u Podgorici
Dr Savo Tomović, docent PMF-a u Podgorici

Komisija za odbranu doktorske disertacije:

Dr Milo Tomašević, vanredni profesor ETF-a u Beogradu
Dr Predrag Stanišić, redovni profesor PMF-a u Podgorici
Dr Milenko Mosurović, redovni profesor PMF-a u Podgorici
Dr Savo Tomović, docent PMF-a u Podgorici
Dr Aleksandar Popović, docent PMF-a u Podgorici

Datum odbrane: **18.12.2014. godine**

Predgovor

Projektovanje aplikacija koje će samostalno, bez pomoći čovjeka, moći da izvode zaključke i pronalaze obrasce u samim podacima, postaje sve veći izazov u oblasti proizvodnje softvera. Količina podataka koja je dostupna samo unutar jednog sistema je ogromna i postavlja skoro nepremostivu prepreku za tradicionalne arhitekture i infrastrukture. Metodi mašinskog učenja su obećavajuće polje istraživanja za takav tip problema. Vještačke neuronske mreže predstavljaju atraktivnu i popularnu paradigmu mašinskog učenja za dizajn i analizu inteligentnih adaptivnih sistema za razne klase zadataka: od klasifikacije uzoraka, preko aproksimacije funkcija do kontrole i upravljanja robotima. Razlozi za to su višestruki: potencijal za masivna paralelna izračunavanja, robustnost i otpornost na smetnje i šum, elastičnost u odnosu na kvar pojedinih komponenti i sličnost, makar i površna, sa biološkim neuronskim mrežama.

Prvi korak za rješavanje nekog zadatka primjenom neuronske mreže je definisanje topologije ili arhitekture mreže, tj. zadavanje procesorskih jedinica ili neurona i veza među njima. Mreža se, zatim, obučava za rješavanje problema primjenom nekog od algoritama nadgledanog učenja, kao što je „propagacija unazad". Izbor arhitekture mreže utiče ne samo na to koja će funkcija biti implementirana mrežom, već ima i veliki uticaj na sposobnost algoritma obučavanja da „nauči" željenu funkciju. Kako se izbor topologije mreže najčešće obavlja po principu „probe i greške", čitava proces je veoma neefikasan. Jedan od metoda koji poboljšava efikasnot ovog procesa je evolucioni razvoj topologija neuronskih mreža, u kojem se topologija mreže kodira u neki drugi oblik. Evolucione strategije, kao što su genetski algoritmi ili evoluciono programiranje, koriste se za pretraživanje novog prostora kodiranih mreža. Kratak pregled takvih metoda, sa posebnim osvrtom na tip kodiranja poznat kao „indirektno kodiranje", dat je drugom poglavlju ove teze.

Primjenom tzv. „nenadgledanog učenja", neuronske mreže se mogu iskoristiti i za pronalaženje uzoraka u podacima ili za pretprocesiranje nekog skupa podataka. Nenadgledani algoritmi obučavanja mreže mogu se posmatrati kao preslikavanja koja ulazni prostor mapiraju u izlazni prostor, pri čemu svaki od tih prostora ima neka metrička ili topografska svojstva koja je važno očuvati. Pored toga, u podacima često postoje nejasne granice ili nepreciznosti koje možemo modelirati primjenom fazi logike. Treće poglavlje opisuje tri algoritma koja u standarni algoritam Kohonena dodaju mogućnost obučavanja po grupama neurona, očuvanje topoloških i metričkih osobina i elemente fazi logike na svim nivoima algoritma.

Četvrto poglavlje posvećeno je metodu koji je originalno razvijen u toku istraživanja za ovu disretaciju. Predloženi metod predstavlja kombinaciju tri biološke paradigme: modularnih neuronskih mreža, genetskog algoritma i L-sistema.

Topologije mreže se kodira pravilima sistema Lindenmajera (L-sistemima) korišćenjem koncepta indirektnog kodiranja. Genetski algoritam se koristi za pretraživanje prostora stanja kodiranih topologija. Svaka jedinka u populaciji genetskog algoritma predstavlja jednu ili više kodiranih topologija. Iz jedinki se generiše neuronska mreža koja se obučava za rješavanje datog zadatka primjenom algoritma propagacije greške unazad ili nekog drugog metoda obučavanja. Proces obučavanja određuje vrijednost funkcije kvaliteta originalne jedinke i na taj način se omogućava razvoj optimalne arhitekture za dati zadatak. Kako se genetski algoritmi i druge evolucione strategije često koriste za optimizaciju nediferencijabilnih funkcija sa više lokalnih ekstremuma, oni su prirodan izbor i u ovom radu, kada nije jasno kako tačno izgleda prostor kodiranih topologija mreža. Predloženi metod koristi samo stringove za reprezentaciju, pa se na taj način postiže efikasnije pronalaženje odgovarajuće topologije. Formalno je opisan pojam lijevog i desnog konteksta. U odnosu na druge sisteme koji kombinuju L-sisteme i neuronske mreže, i koji kontekst određuju na osnovu generisanog grafa ili njegove matrične reprezentacije, izloženi metod kontekst definiše na osnovu stringa koji kodira pravilo. Mreže generisane na ovaj način imaju modularnu strukturu, čime se povećava skalabilnost i smanjuje se prostor stanja koji pretražuje genetski algoritam.

U cilju verifikacije rezultata istraživanja sprovedenog u ovoj doktorskoj tezi, implementiran je programski paket ***NNGen*** (Neural Net Generator), čija je struktura i funkcionalnost opisana u petom poglavlju. Modularna struktura ovog paketa omogućava njegovu laku nadogradnju i prilagođavanje novim alatima. Originalno je ovaj paket bio zamišljen samo kao implementacija algoritama i metoda opisanih u četvrtom poglavlju ove teze. Razvoj velikog broja programskih alata koji mogu generisati i obučavati neuronske mreže, uslovio je da koncept samog paketa ***NNGen*** bude naknadno promijenjen, pa je njegova osnovna funkcionalnost nadograđena opcijom generisanja koda za jedan takav alat – JOONE paket.

Testiranje predloženog metoda generisanja topologija izvedeno je na više problema različite veličine i opisano je u šestom poglavlju. Dva osnovna cilja testiranja su:

- provjera da li metod zaista generiše odgovarajuće topologije za zadati problem;
- da li generisane modularne mreže imaju bolje performanse, npr. grešku ili brzinu konvergencije, od odgovarajućih nemodularnih mreža.

Problemi na kojima su izvršeni eksperimenti kreću se od najprostijih klasičnih primjera, kao što su problem ekskluzivno-ILI ili raspoznavanje slova T i C, preko problema „preslikavanja", pa sve do poznatog problema klasifikacije raka dojke. U toku izvođenja eksperimenata razvijeno je više heuristika koje omogućavaju dodavanje neurona ili modula u toku procesa obučavanja mreže, u cilju povećanja kvaliteta generisane mreže ili povećanja brzine konvergencije. Ovaj koncept je u

biologiji poznat kao Boldvinov efekat i eksperimenti su pokazali da njegova primjena može uticati na kvalitet generisanih rješenja kao i na brzinu dobijanja samog rješenja.

Najveći dio algoritama i metoda opisanih u ovoj tezi nastao je kao rezultat istraživanja na katedri Automatizacije kompjuterskih sistema (ACBK - Автоматизации Систем Вычислительных Комплексов) Fakuleta numeričke matematike i kibernetike (BMK) Moskovskog državnog univerziteta MGU "M. V. Lomonosov", pod rukovodstvom profesora Lava Nikolajeviča Karaljova, kome dugujem neizmjernu zahvalnost za njegovo strpljenje i uvijek korisne komentare. Implementacija metoda generisanja topologija modularnih neuronskih mreža i razvoj paketa ***NNGen*** obavljen je na Univerzitetu Crne Gore, uz nemjerljiv doprinos profesora Mila Tomaševića.

Izvod teze

Vještačke neuronske mreže predstavljaju model izračunavanja koji se može implementirati softverski ili hardverski i koji pokušava da imitira ponašanje i prilagodljivost nervnih bioloških sistema. One se uobičajeno sastoje od više jedinica izračunavanja ili neurona, koji imaju veći broj ulaza i izlaza. Broj i tip neurona i skup veza između njih definišu topologiju ili arhitekturu neuronske mreže.

Vještačke neuronske mreže se primjenjuju u mnogim realnim problemima, u opsegu od klasifikacije uzoraka do upravljanja robotima ili vozilima. Kreiranje neuronske mreže počinje određivanjem topologije mreže. Izbor topologije, u najvećoj mjeri, utiče na to koja će funkcija biti implementirana mrežom i ima veliki uticaj na sposobnost algoritma obučavanja da „nauči" tu funkciju. Genetski algoritmi i drugi evolucioni metodi mogu se koristiti za automatizovano pronalaženje rješenja za ovaj problem. Evolucioni metodi su klasa stohastičkih metoda traženja zasnovanih na skupu jedinki i inspirisanih principima Darvinovske evolucije.

Nalaženje odgovarajuće topologije mreže je izazovan zadatak. Postoji mnogo metoda koji zajedno koriste evolucione algoritme i neuronske mreže. Većina tih metoda primjenjuje direktno kodiranje, gdje se topologija i/ili težinski koeficijenti mreže direktno kodiraju u hromozome, što utiče na veličinu hromozoma i dovodi do pojave velikog broja hromozoma u populaciji.

Osnovni izazov evolucionih metoda je da razviju fenotipove čija je složenost uporediva sa onom koju imaju biološki sistemi. Iz tog razloga razvijen je čitav niz indirektnih šema kodiranja. Ove šeme kodiranja su evolucioni mehanizmi koji iste „gene" koriste više puta u procesu izgradnje fenotipa. Na taj se način složeni fenotipovi predstavljaju na kompaktan način. Razvoj je prirodan izbor za implementaciju ovih načina kodiranja, jer i sama priroda koristi isti proces.

U procesu evolucione sinteze neuronskih mreža, pored izbora odgovarajućih evolucionih parametara kao što su veličina populacije, vjerovatnoća mutacije i ukrštanja i funkcije kvaliteta, ključni problem je izbor odgovarajućeg načina predstavljanja hromozoma. U ovoj tezi predstavljen je jedan originalno razvijen metod automatizovanog generisanja topologija modularnih neuronskih mreža. Metod kombinuje više bioloških paradigmi. Povećanje hardverske moći utiče i na veću dostupnost neuronskih mreža sa velikim brojem neurona. Veličina tih mreža nije ograničena, pa će biti sve teže kreirati ih i razumjeti njihovo funkcionisanje. Iz tog razloga je važno pronaći metod opisa topologije mreže koji će biti skalabilan. Modularne neuronske mreže pokazuju bolje performanse od odgovarajućih nemodularnih mreža. Ljudski mozak se može posmatrati kao modularna mreža, pa je predloženi metod zasnovan na prirodnim procesima u mozgu. Sistemi Lindemanajera

(L-sistemi) se koriste kao mehanizam kodiranja, da bi modelirali recepte rasta prisutne kod biljaka.

Genetski algoritmi su evolucioni lokalni metod traženja u prostoru mrežnih topologija koji ima veliku dimenziju. Oni predstavljaju jedan od optimizacionih metoda koji se mogu primijeniti čak i kada ne postoji analitičko znanje o problemu, pa su pogodni za automatizovano traženje odgovarajućih topologija mreže. Svaka individua (hromozom) u populaciji predstavlja kodirani opis neke topologije. Cilj je da odredimo optimalnu topologiju za dati problem. Kvalitet mreže se izračunava u procesu obučavanja mreže i prevodi se u funkciju kvaliteta, na osnovu koje genetski algoritam određuje vjerovatnoću selekcije individua za operaciju ukrštanja. Usko grlo ovog procesa je, očigledno, vrijeme obučavanja mreže za svaku topologiju.

Neuronska mreža se obučava jednom varijantom metoda „gradijentnog spusta" – algoritmom prostiranja greške unazad („back propagation"). Uvođenje heuristika za dodavanje modula i neurona u toku procesa obučavanja poboljšava brzinu konvergencije i kvalitet samog rješenja.

Mogućnosti predloženog metoda su istražene kroz više eksperimenata. Rezultati pokazuju da metod zaista generiše modularne mreže i da modularnost u procesu kreiranja mreže može poboljšati njihove krajnje performanse. Generisane mreže za probleme čiji skupovi za obučavanje nisu veliki imaju bolje performanse od odgovarajućih višeslojnih neuronskih mreža. Pored boljih performansi, modularizacija ima efekat i da su šeme kodiranja skalabilnije, što je veoma važno jer se time smanjuje prostor koji pretražuje genetski algoritam.

Abstract

Artificial neural networks are computational models implemented in software or specialized hardware devices that attempt to capture the behavioral and adaptive features of biological neural systems. They are typically composed of several interconnected processing units, or 'neurons' which can have a number of inputs and outputs. The number and type of neurons and the set of possible interconnections between them define the topology or architecture of the neural network.

Artificial neural networks are applied to many real-world problems, ranging from pattern classification to robot and vehicle control. In order to design a neural network for a particular task, the choice of a network topology has to be addressed. The choice of a topology determines to a large extent which functions can be implemented by the network. It also has a strong influence on the ability of the training algorithm to learn the desired function. Genetic algorithms and other evolutionary search methods can provide automatic solutions to this problem. Evolutionary methods are a class of population-based, stochastic search methods inspired by the principles of Darwinian evolution.

Finding suitable neural network for problem at hand is a challenging task. There are numerous methods combining evolutionary algorithms and neural networks. Most of these methods employ direct encoding – the architecture of the network and/or its weights are coded into chromosomes directly, which results in a huge number of chromosomes and increases their dimensions.

A major challenge for evolutionary methods is to evolve phenotypes at the same level of complexity as found in biological systems. In order to meet this challenge, a number of indirect encoding schemes is proposed in literature. Indirect encoding scheme is an evolutionary mechanism where the same genes are used multiple times in the process of building a phenotype. Complex phenotypes can be represented in a compact way with gene reuse. Development is a natural choice for implementing indirect encodings, because nature itself uses this very process.

The evolutionary synthesis of a neural network leads to several design choices. Besides the choice of the setting of appropriate evolutionary parameters e.g., population size, mutation and crossover rates and an appropriate fitness function, the key problem is the choice of suitable genetic representations.

In this thesis, a new method for automatic generation of modular artificial neural network topologies is proposed. A number of biological paradigms is incorporated in the method. As computer hardware becomes more powerful, large artificial neural networks will become feasible. It will become increasingly difficult to design them and understand their internal operation. There will be no limit on the

size of the networks. That is why it is very important to find design methods that are scalable to large network sizes. Modular artificial neural networks have a better performance than their non-modular counterparts. The human brain can also be seen as a modular neural network, and the proposed method is based on the natural process in the brain. Lindenmayer's system (L-system) is used as an indirect encoding scheme in order to model the kind of recipes that nature uses in biological growth of plants.

Genetic algorithms are used as an evolutionary local search method in high-dimensional space of network topologies. They are optimization methods that can operate even if no analytic knowledge about the problem is available. Therefore, it is a technique that is suitable to automatically search for appropriate network topologies. Each individual in a population contains an encoded description of a network topology and the goal is to find the optimal topology for a given task. Final quality of a network is determined after training it on the training data and the performance of the trained network is transformed into a fitness measure. Fitness measure is used by the genetic algorithm to calculate the selection probabilities of individual. The obvious bottleneck of the proposed method is its long training time that is needed to evaluate each network topology.

Neural network is trained using a gradient descent algorithm known as back propagation algorithm. Introducing heuristics for adding modules or nodes during the training process improves the speed of convergence and quality of solution.

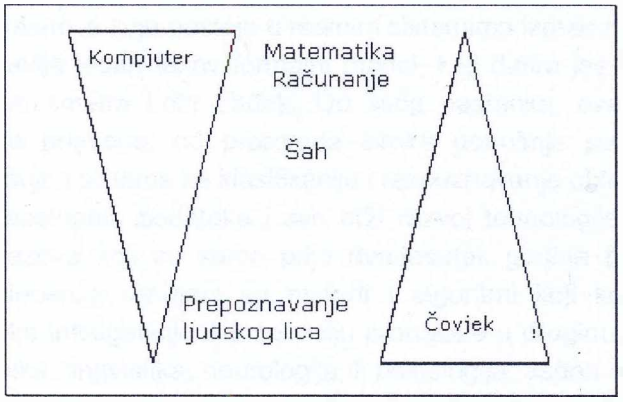
A number of experiments have been done to investigate the possibilities of the proposed method. Results show that the method does find modular networks, and using modularity when designing artificial neural networks can improve their performance. Generated networks on small and medium size problems outperform standard multilayer feed forward networks. Besides the fact that designing modular network topologies specific for a task will result in better performance, modularization also allows for a more scalable coding scheme. In order to minimize the search space of the genetic algorithm, it is very important to keep the size of the encoding as small as possible.

Sadržaj

1. Uvod

1.1 Definisanje problema istraživanja

Metodi vještačke inteligencije pokazali su se izuzetno uspješnim u oblastima koje se tradicionalno smatraju veoma teškim za čovjeka, kao što su razne vrste klasifikacije, aproksimacija funkcija ili igranje šaha. S druge strane, u zadacima tipa vizuelnog raspoznavanja objekata ili prepoznavanja govora, koji su za čovjeka veoma jednostavni, računari su i dalje na relativno niskom nivou. Ovaj efekat je dobio ime "kognitivna inverzija" [EzhShu98], što je ilustrovano na slici 1.1. Može se reći da su metodi vještačke inteligencije, kao što su npr. vještačke neuronske mreže, evolucione strategije, fazi logika ili računarstvo inspirisano biologijom pokušaj da se pronađe „intuicija o problemu". Na kognitivnoj skali sa slike 1.1, ovi metodi su bliže ljudskoj strani i predstavljaju pokušaj pronalaženja rješenja primjenom metoda koji su bliži ljudskom nego računarskom načinu rezonovanja.



Slika 1.1 – Kognitivna inverzija (adaptirano iz [EzhShu98])

Vještačke neuronske mreže već dugo vremena predstavljaju etabliranu računarsku paradigmu i imaju široku primjenu, posebno u oblastima klasifikacije, raspoznavanja oblika, aproksimacije funkcija, nelinearne regresije i kontrole ([Hay04], [EzhShu98]). Neuronske mreže možemo posmatrati kao metod implementacije složenih nelinearnih preslikavanja pomoću elementarnih jedinica, koje su povezane adaptivnim težinskim koeficijentima. Kao i kod bioloških neuronskih mreža, elementarna jedinica je neuron. Vještački neuron predstavlja idealizovani matematički model realnog biološkog neurona.

Tradicionalni pristup razvoja vještačkih neuronskih mreža podrazumijeva da ekspert definiše topologiju mreže koja se zatim obučava nekim algoritmom. Proces

definisanja topologije često se bazira na principu „probe i greške“, pa zato nije efikasan, ni sa stanovišta iskorišćenja vremenskih resursa, ni sa stanovišta produktivne upotrebe ljudskih resursa. Stoga, razvoj tehnika i algoritama koji automatizuju ovaj proces, bilo djelimično ili, u idealnom slučaju, potpuno, predstavlja važnu oblast istraživanja. Postoji veliki broj algoritama za obučavanje vještačke neuronske mreže. Primjenljivost nekog algoritma za obučavanje direktno zavisi of topologije mreže, tj. od načina na koji su povezani pojedine komponente mreže. Zbog toga je veoma važno kreirati okvir koji omogućava automatsko kreiranje topologije vještačke neuronske mreže ili istovremeni razvoj i topologije i koeficijenata mreže.

Neuronske mreže se mogu iskoristiti i za pronalaženje uzoraka u podacima primjenom tzv. „nenadgledanog učenja“ ili za pretprocesiranje nekog skupa podataka. Najčešće takve mreže imaju neku „pravilnu topologiju“, npr. u obliku pravougaonika ili trougla. Algoritmi za obučavanje takvih mreža se stalno razvijaju i dopunjavaju posebnim slučajevima za specifične domene. Jedna od izazova je kako smanjiti složenost tih algoritama i kako u njih uključiti topografsku informaciju o ulaznom prostoru.

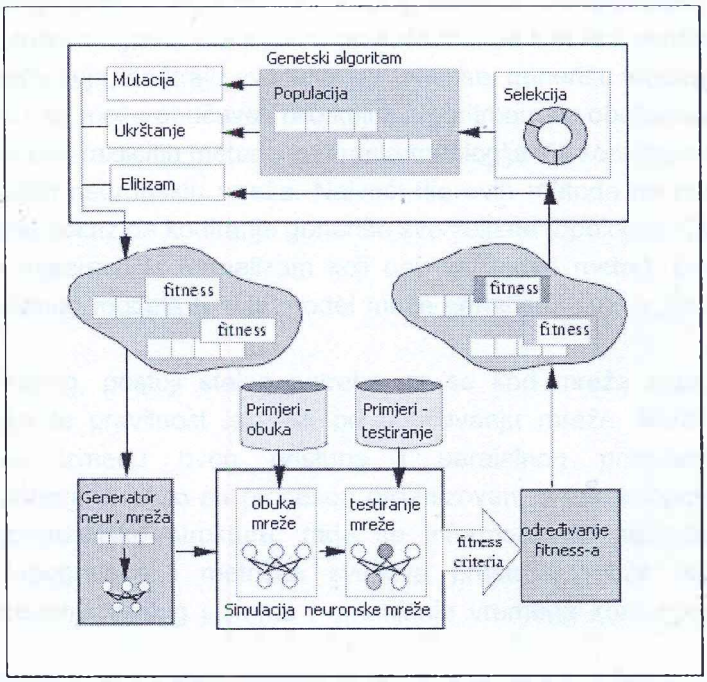
U svakodnevnom rasuđivanju i izvođenju zaključaka, čovjek veoma često donosi odluke na osnovu nepotpunih ili nejasnih činjenica. Primjena računara u mnogim oblastima ljudskog djelovanja dovela je do potrebe da se nepreciznosti, nepotpunosti ili nejasnoće koje postoje u realnim sistemima izraze na formalan način. Fazi logika predstavlja jedan takav formalni model, koji datira još iz 1965. godine i čijim se osnivačem smatra Lofti Zadek. Od svog nastanka, ova teorija je našla značajne praktične primjene, od proizvoda široke potrošnje pa do automatskih sistema za upravljanje i sistema za klasifikaciju i raspoznavanje oblika.

Veliki broj dostupnih podataka i sve brži razvoj tehnologije postavljaju pred tvorce softvera izazove koji su samo prije dvadesetak godina bili nezamislivi. U posljednje dvije decenije razvijeni su metodi i algoritmi koji kombinuju različite paradigme vještačke inteligencije ili inspiraciju pronalaze u drugim naukama kao što su genetika, botanika, lingvistika, neurologija ili psihologija. Jedan od takvih metoda, koji kombinuje vještačke neuronske mreže i evolucione strategije, naziva se evolucioni razvoj neuronskih mreža („embriogeny“ [StaMii03]). Još od sredine osamdesetih godina prošlog vijeka, kada se pažnja naučne javnosti ponovo vratila ka neuronskim mrežama, razvijani su metodi koji u različitim fazama kreiranja i obučavanje neuronske mreže koriste principe evolucije ili fazi logiku.

Evolucione strategije su jedan od nekoliko optimizacionih metoda koji su primjenljivi čak i slučaju kada nam nije dostupno analitičko znanje o zadanom problemu. Paralelno sa rastom računarske moći, raste i praktični potencijal evolucionih strategija. Evoluciona sinteza inteligentnih agenata, softverskih ili hardverskih, postaje sve značajnija oblast istraživanja. Takvi agenti mogu se razvijati za svaki tip problema za koji je moguće definisati odgovarajuću funkciju kvaliteta („fitness function“, [Mic89], [Whi89]). Iz tog razloga su evolucione strategije pogodne

za izbor optimalne topologije neuronske mreže u prostoru svih mogućih topologija. Genetski algoritmi su najpopularnija evoluciona strategija, pa su u ovoj disertaciji upravo oni upotrebljeni kao evoluciona strategija.

Informacije o neuronskim mrežama kodiraju se u hromozome, koji predstavljaju polaznu populaciju genetskog algoritma. Proces evaluacije sastoji se od određivanja kvaliteta kodirane mreže, što se najčešće postiže obučavanjem mreže i transformacijom srednje kvadratne greške ili nekog drugog kriterijuma u odgovarajuću brojnu vrijednost funkcije kvaliteta. Operatori mutacije, ukrštanja (rekombinacije) i inverzije kreiraju nove hromozome koji zamjenjuju sve ili najmanje kvalitetne hromozome iz prethodne populacije. Kompletan proces koji za evolucionu strategiju koristi genetski algoritam prikazan je na slici 1.2.



Slika 1.2 – Evolucioni razvoj neuronskih mreža: opšti okvir (adaptirano iz [Sta04])

Sama ideja je jednostavna, ali postoji više problema prilikom realizacije. Prvi od problema, čijim se rješavanjem najvećim dijelom bavi i ova disertacija, jeste način kodiranja neuronskih mreža da bi se efikasno mogao primijeniti genetski algoritam. Drugo „usko grlo" je samo obučavanje mreže, koje se najčešće obavlja tradicionalnim algoritmima obučavanja, koji se baziraju na metodu gradijentnog spusta ili nekog drugog metoda optimizacije.

1. 2 Ciljevi istraživanja

Cilj istraživanja ove doktorske disertacije jeste definisanje i implementacija jednog metoda generisanja topologija vještačkih neuronskih mreža, koji bi razriješio ili barem ublažio prethodno opisane probleme. Ovaj metod je namijenjen generisanju mreža u kojima se signal prostire unaprijed (tzv. „feed-forward neural networks"), iako se, uz neznatne izmjene, može primjeniti i za generisanje rekurzivnih topologija. Osnovna odlika predloženog metoda jeste da se ne kodira sama topologija, već se kodiraju pravila koja generišu topologiju mreže. Pravila se kodiraju preko GL-sistema koji je zasnovan na sistemima Lindenmajera za morfologiju biljaka. GL-Sistem predstavlja jednu varijantu kontekstnog sistema Lindenmajera, pogodnu za predstavljanje strukture grafa. Genetskim operatorima se simulira evolucija pravila za generisanje mreže koja, na kraju evolucionog procesa, generišu topologiju krajnje mreže. Takva mreža se može obučavati bilo kojim algoritmom za obučavanje.

Postoji veliki broj različitih metoda kodiranja topologija i/ili koeficijenata i drugih parametara vještačkih neuronskih mreža. Najveći dio ovih metoda ne nudi nikakav formalni matematički dokaz da kodiranje generiše sve željene topologije. Cilj ove teze jeste i da ponudi matematički formalizam koji opisuje zadati metod. Dokazana je kompletnost ponuđenog modela tj. dati model može generisati svaku „feed-forward" topologiju.

Pored navednog, postoji stalna potreba da se kod mreža koja posjeduju „pravilnu" topologiju ta pravilnost iskoristi pri obučavanju mreže. Može se uočiti izvjesna analogija između ovog pristupa i paralelnog programiranja na višeprocesorkim sistemima. Ako su procesori organizovani u paralelopiped ili neku drugu pravilnu geometrijsku strukturu, tada se informacija o susjedima nekog procesora, tzv. topografska i metrička svojstva prostora, može iskoristiti za ravnomjerno opterećenje cijelog sistema i smanjenje vremena komunikacije među procesorima.

Često su ulazni podaci mreže nejasni ili neprecizni, pa se i rezultati koje mreža daje moraju uzeti sa rezervom. Ako ulaze takve mreže modeliramo pomoću fazi logike, tada se taj formalizam može prenijeti i u fazu obučavanja mreže i na dobijene rezultate. Uključivanje u postojeće algoritme obučavanja informacije o topografskim i metričkim svojstvima ulaza ili nepreciznostima u ulaznim podacima jedan je od ciljeva ove disertacije.

Radi praktične potvrde dobijenih rezultata, planiran je i razvoj odgovarajućih softverskih komponenti u okviru programskog paketa ***NNGen***, koji je razvijen u toku istraživanja. Više informacija o konceptima na kojima se zasniva navedeni paket može se naći u [Suk02a] i [Suk02b]. Između ostalog, na sadašnjem stepenu razvoja podržano je generisanje topologije „feed-forward" neuronske mreže, obučavanje mreže algoritmom propagacije greške unazad („back-propagation" [Hay04]) i

obučavanje samoorganizujućih karti („self-organizing maps", [Koh01]). Moguće je i generisanje koda u programskom jeziku C i u programskom jeziku Java, kada se mreža kreira i obučava primjenom paketa JOONE (Java Object Oriented Neural Engine) [Joo07].

Dalji razvoj paketa ***NNGen*** usmjeren je na prevođenje cjelokupnog koda u C++ i Javu, kreiranje grafičkog interfejsa i dodavanje modula za generisanje mreža u nekim drugim paketima za rad sa neuronskim mrežama kao što su Encog, FANN (Fast Artificial Neural Network), MATLAB Neural Network Toolbox i Neuroph.

1.3 Rezultati istraživanja

Glavni rezultati istraživanja ove doktorske disertacije su:

- širok pregled relevanatnih koncepata i rješenja;
- kompletna specifikacija GL-sistema za zadavanje topologije feed-forward neuronske mreže. Definisana su pravila konteksta i način kreiranja topologije mreže;
- implementirane programske komponente koje su namijenjene projektantima i koje realizuju pojedinačne segmente predloženog metoda;
- formulacija postupaka za generisanje koda koji predstavlja vještačku neuronsku mrežu. Putem navedenih postupaka moguće je generisati (i) programski kod u jeziku *C* i (ii) programski kod u jeziku Java, koji predstavlja specifikaciju za programski paket JOONE;
- provedene simulacije na četiri skupa i poređenje rezulatata sa sličnim algoritmima;
- algoritam obučavanja samoorganizujuće mreže po disjunktnim klasterima susjednih neurona. Ovaj algoritam za mrežu sa N čvorova i M ulaza ima složenost $O(N \cdot logM)$, za razliku od standardnog algoritma Kohonena čija je složenost $O(N \cdot M)$;
- algoritam klasifikacije koji kombinuje fazi logiku i samoorganizujuće karte. Koncepti fazi logike primjenjuju se na ulaznim podacima, u toku obučavanja mreže i pri tumačenju rezultata;
- algoritam za očuvanje topografskih i metričkih svojstava, koji koristi dvije mjere očuvanja tih svojstava.

1.4 Struktura rada

Pored uvoda ova teza sastoji se od pet poglavlja. U drugom poglavlju prvo je dat pregled osnovnih koncepata korišćenih u ovoj tezi: neuronskih mreža, genetskih

algoritama, sistema Lindenmajera i modularnosti. Napravljen je i osvrt na vladajuće trendove i metode generisanja topologija neuronskih mreža. Naročita pažnja je posvećena onim metodima koji se bave generisanjem topologija „feed-forward" mreža primjenom tzv. indirektnog kodiranja, tj. metoda u kojima se topologija mreže ne kodira direktno. Prikazana su pozitivna iskustva i istaknute glavne razlike u odnosu na pristup koji je predložen u okviru ove doktorske teze.

Treće poglavlje posvećeno je samoorganizujućim mrežama. Prikazana su tri algoritma vezana za ovaj tip neuronskih mreža: algoritam obučavanja po disjunktnim klasterima, algoritam za očuvanje metričkih i topografskih svojstava i algoritam fazi klasifikacije.

Četvrto poglavlje sadrži opis metoda za generisanje topologija „feed-forward" vještačkih neuronskih mreža. Topologija mreže predstavlja se pravilima GL-sistema koja evoluiraju primjenom genetskog algoritma i na taj način zadaju novu topologiju mreže. Kvalitet mreže, tj. njen „fitness", određuje se obučavanjem mreže algoritmom back-propagation. Data je i formalna definicija GL-sistema i konteksta u takvom sistemu, koja se zasniva na stringovima.

Peto poglavlje opisuje implementaciju programskog paketa ***NNGen*** (Neural Net Generator). Dat je detaljan opis glavnih modula paketa i osnovnih funkcije pojedinih modula, kao i primjeri koda za neke faze predloženog metoda.

Šesto poglavlje posvećeno je simulacijama. Opisana je okolina u kojoj su izvođenje simulacije. Programski paket ***NNGen*** testiran je na 4 različita zadatka: ekskluzivno-ILI (XOR problem), TC problem ([Hay04]), problem preslikavanja („mapping problem", [Hoo91]) i klasifikacija raka dojke (Breast Cancer Wisconsin Data Set). Za svaki zadatak je dat opis skupa podataka i prikazane su mreže generisane paketom ***NNGen***. Karakteristike dobijenih mreža upoređene su sa svojstvima mreža dobijenih drugim metodama. Pored toga, date su i heuristike za dodavanje čvora i modula u mrežu u toku obučavanja mreže.

Dodatak A sadrži primjer generisanog koda u jeziku C za poznati problem „ekskluzivno-ILI", dok je dodatak B posvećen generisanoj JOONE specifikaciji istog problema. U dodatku C opisane su neke klase iz paketa ***NNGen*** i dat je kod nekih metoda. Dodatak D predstavlja C++ klase i implementaciju metoda za algoritam Kohonena za obučavanje samoorganizujućih mreža.

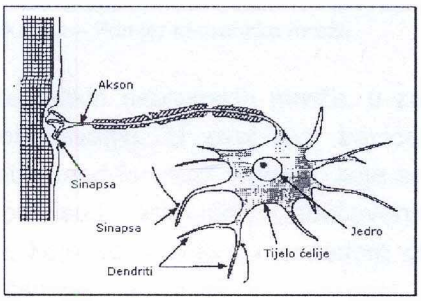
2. Pregled aktuelnog stanja u oblasti istraživanja

U ovom poglavlju dat je pregled aktuelnih koncepata i metoda za generisanje topologija neuronskih mreža. U odjeljku 2.1 dat je pregled ključnih ideja i pristupa vještačke inteligencije koji su upotrebljeni u okviru ove disertacije. Odjeljak 2.2 posvećen je pregledu različitih metoda kodiranja topologije neuronske mreže. U odjeljku 2.3 opisani su metodi evolucionog razvoja neuronskih mreža. Ovo je dijelom zato što trenutno ovi pristupi igraju zapaženu ulogu u disciplini vještačke inteligencije i „mekog računarstva", a dijelom i zato što se ova doktorska disertacija bavi upotrebom evolucionih strategija za generisanje mreže. Ovi metodi, u određenoj mjeri, podržavaju koncepte i pristupe koji imaju sličnosti sa konceptima i pristupima koji su opisani u okviru ove doktorske teze. Prikazana su pozitivna iskustva koja će biti iskorišćena u toku izrade ove doktorske disertacije. Pored toga, istaknute su glavne razlike između pristupa na kojima se ti metodi zasnivaju i pristupa koji je opisan u okviru ove doktorske disertacije.

2.1 Pregled ključnih koncepata

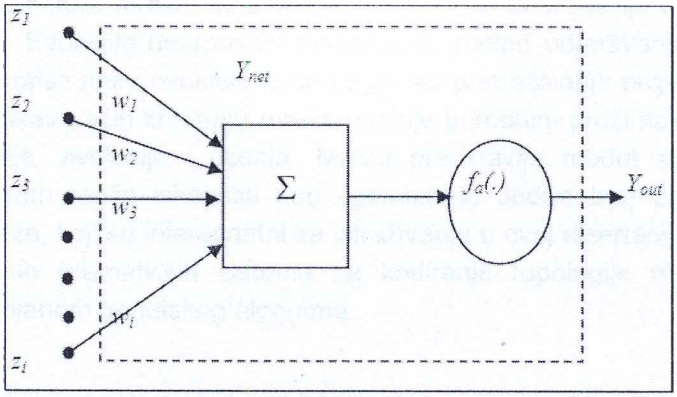
2.1.1 Vještačke neuronske mreže

Vještačke neuronske mreže već dugo vremena predstavljaju etabliranu računarsku paradigmu i imaju široku primjenu, posebno u oblastima klasifikacije, raspoznavanja oblika, aproksimacije funkcija, nelinearne regresije i kontrole ([Hay04], [EzhShu98]). Neuronske mreže možemo posmatrati kao metod implementacije složenih nelinearnih preslikavanja pomoću elementarnih jedinica, koje su povezane adaptivnim težinskim koeficijentima. Kao i kod bioloških neuronskih mreža, elementarna jedinica je neuron (slika 2.1).



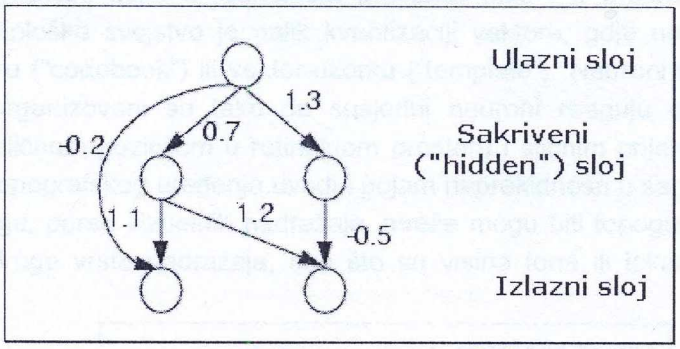
Slika 2.1 – Biološki neuron

Vještački neuron (slika 2.2) predstavlja idealizovani matematički model realnog biološkog neurona, gdje z_1, z_2... predstavaljaju ulaze, w_1, w_2.. težinske koeficijente, Σ označava sabiranje a f_a je aktivaciona funkcija. Formalno, izlaz Y_{out} neurona sa slike 2.2 izračunava se po formuli: $Y_{out} = f_a\left(\sum_{k=1}^{I} w_i \cdot z_i\right)$



Slika 2.2 – Struktura vještačkog neurona

Vještačka neuronska mreža (slika 2.3) je grupa međusobno povezanih vještačkih neurona, koja modelira strukturu i dio funkcionalnosti ljudskog mozga. Performanse vještačkih neuronskih mreža i broj neurona u njima su, još uvijek, daleko od performansi i broja neurona u ljudskom mozgu. Proces obučavanja neuronske mreže zasniva se najčešće na ažuriranju težinskih koeficijenta i podsjeća na prosljeđivanje impulsa u nervnom sistemu [Hay04].



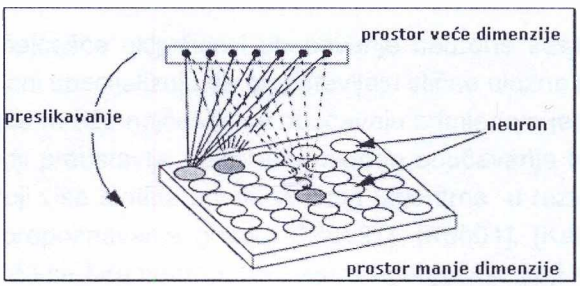
Slika 2.3 – Primjer neuronske mreže

Postoji više modela vještačkih neuronskih mreža, u zavisnosti od tipa veza između neurona i načina obučavanja. U ovoj tezi bavićemo se mrežama sa prostiranjem impulsa unaprijed („feed-forward" mreže), koje se obučavaju metodom nadgledanog učenja („supervised learning", obučavanje sa učiteljem) i samoorganizujućim mrežama, koje se obučavaju metodom nenadgledanog učenja („unsupervised learning", obučavanje bez učitelja).

Prva faza u rješavanju problema primjenom neuronskih mreža jeste određivanje topologije ili arhitekture mreže, tj. načina povezivanja neurona. Teški zadaci zahtijevaju složene mreže sa velikim brojem veza, pa je prostor za pretraživanje tako velike dimenzije, da čak i ako rješenje postoji, teško ga je pronaći. Arhitektura mreže često se određuje metodom probe i greške („trial-and-error") ili na osnovu prethodnog iskustva projektanta, pa je potrebno razviti metode automatskog određivanja arhitekture mreže, koja će poslije procesa obučavanja biti optimalna za zadati problem. Evolucija neuronskih mreža jeste metod određivanja odgovarajuće topologije koji koristi neku evolucionu strategiju za pretraživanje prostora stanja svih topologija. Ovakav način kreiranja mreže nastaje prirodnim prožimanjem dva važna oblika adaptacije: evolucije i učenja. Mreža predstavlja modul koji uči, dok se genetski algoritam može iskoristiti kao optimizator. Jedan broj pristupa evoluciji neuronskih mreža, koji su interesantni za istraživanja u ovoj disertaciji, zasniva se na upotrebi različitih gramatičkih sistema za kodiranje topologije mreže i njihovoj optimizaciji primjenom genetskog algoritma.

2.1.2 Samoorganizujuće mreže

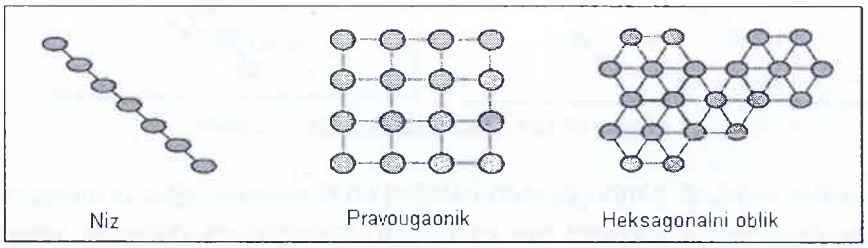
Samoorganizujuće mreže (karte) predstavljaju reprezentaciju podataka koja kombinuje aspekte kvantizacije podataka sa neprekidnošću funkcija. Direktan biološki analog ovim kartama su orijentacione karte u vizuelnom sistemu mnogih vrsta životinja. Pojedinačni neuroni u orijentacionoj mreži imaju receptivna polja koja reaguju samo na određene podskupove nadražaja. Svaki od podksupova nadražaja je strogo lokalizovan, kako u retinalnom prostoru, tako i u prostoru orijentisanih uglova. Ovo biološko svojstvo je nalik kvantizaciji vektora, gdje neuron odgovara kodnom vektoru ("codebook") ili vektor-uzorku ("template"). Neuroni u orijentacionoj karti mozga organizovani su tako da susjedni neuroni reaguju na podskupove nadražaja sa sličnom pozicijom u retinalnom prostoru i sličnim orijentisanim uglom. Ovo svojstvo topografskog uređenja uvodi i pojam neprekidnosti u samoorganizujuće mreže. U mozgu, pored vizuelnih nadražaja, mreže mogu biti topografski uređene u odnosu i na druge vrste nadražaja, kao što su visina tona ili lokalizacija taktilnih nadražaja.



Slika 2.4 – Samoorganizujuća mreža

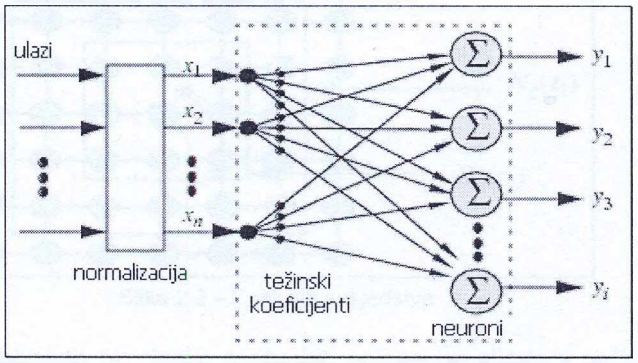
U svim slučajevima, orijentaciona mreža predstavlja površ na korteksu, tj. njena dimenzije je 2, pa se često samoorganizujuće mreže posmatraju kao jedan oblik nelineranog preslikavanja ulaznog prostora veće dimenzije, prostora nadražaja, u izlazni prostor manje dimenzije – orijentacionu mrežu (slika 2.4).

Struktura mreže određuje susjednost neurona i najčešće su neuroni organizovani u obliku paralelopipeda (pravougaonik u slučaju dvodimenzionalnog prostora), mada su mogući i neki drugi oblici (slika 2.5).



Slika 2.5 - Različite strukture samoorganizujućih mreža

Kompetitivno učenje je adaptivni proces u kome neuroni postepeno postaju senzitivni na pojedine podskupove ulaznih nadražaja. U mreži se pojavljuje podjela posla među neuronima, jer se različiti neuroni specijalizuju da predstavljaju različite tipove ulaznih nadražaja. Specijalizacija se odvija kroz nadmetanje neurona: za dati ulaz x, koji je dostupan svim neuronima (slika 2.6), neuron koji najbolje predstavlja x pobjeđuje i dobija mogućnost da se dodatno obučava.

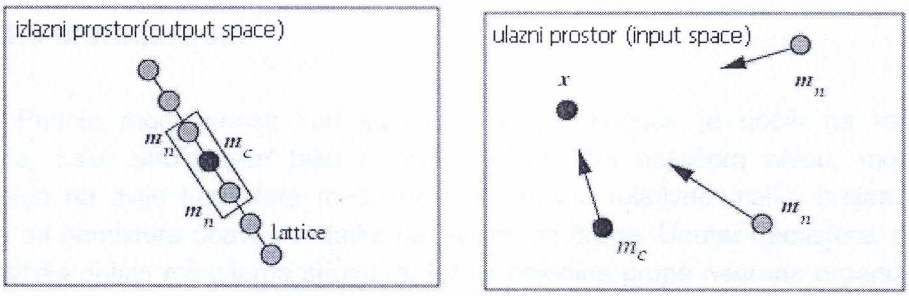


Slika 2.6 – Opšti oblik samoorganizujuće mreže

Ovaj algoritam najčešće uključuje i obučavanje neurona susjednih neuronu-pobjedniku, tako da se oni specijalizuju da predstavljaju slične ulazne podatke.

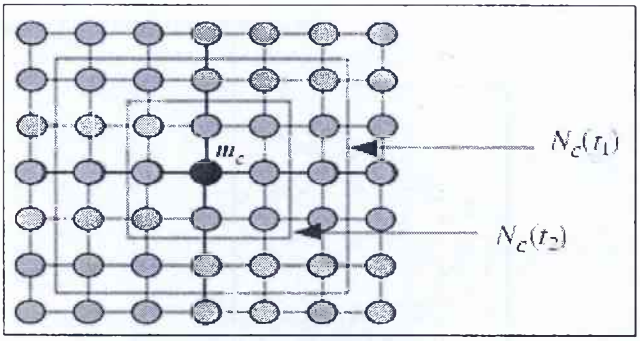
Samoorganizujuće mreže najčešće se obučavaju primjenom jedne od varijanti algoritma Kohonena, koji predstavlja etabliranu metodu obučavanja ovog tipa mreža ([Koh01], [Fri93]). Postoji više stotina primjena ovog algoritma u različitim oblastima kao što su robotika i prepoznavanja govora ([Bau97], [Koh01], [Kas97]). Osnovna ideja ovog algoritma je da se bira neuron, tzv. neuron-pobjednik, koji najbolje reaguje na dati ulazni vektor i da se njegovi težinski koeficijenti i težinski koeficijenti njemu susjednih neurona u mreži promijene. Pojednostavljen postupak prikazan je na slici

2.7, gdje m_c označava vektor težinskih koeficijenata neurona sa najboljim odzivom za ulazni vektor x, a m_n su njemu susjedni neuroni u mreži.



Slika 2.7 – Grafički prikaz algoritma Kohonena

Receptivno polje neurona je na početku rada algoritma dovoljno veliko tako da se promjena težinskih koeficijenata obavlja za sve neurone u mreži, ali se tokom vremena smanjuje. Na početku, receptivna polja neurona se preklapaju, ali se stepen preklapanja vremenom umanjuje i na kraju se dobija podjela koja odgovara ćelijama Voronija [BarBlo99]. Postepeno umanjivanje stepena preklapanja receptivnih polja neurona je neophodno da bi se minimizovala distorzija, koja je važna mjera performansi kod sistema za kvantizaciju vektora.



Slika 2.8 – Funkcija susjedstva

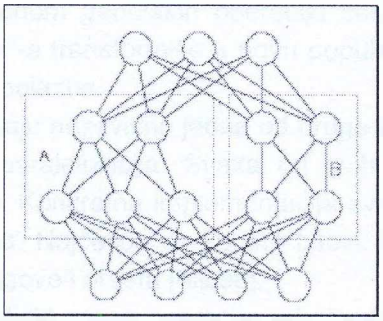
Algoritam Kohonena preciznije može biti opisan na sljedeći način: za slučajno izabrani ulazni vektor $x=x(t)$, u skladu sa raspodjelom $p(x)$, određuje se neuron sa indeksom c takav da je $c = c(x) = \arg\min_i \{d(x, m_i(t))\}$, gdje je t vrijeme i d rastojanje (najčešće Euklidsko rastojanje, ali može se koristiti i neka druga metrika). Broj susjednih neurona određen je funkcijom susjedstva (slika 2.8): ako su r_i i r_j pozicije neurona i i j u mreži, tada je $h_{ij}(t) = h(d(r_i, r_j), t)$ funkcija susjedstva, gdje je d rastojanje (najčešće u mreži). Za vrijeme obučavanja, vektori težinskih koeficijenta neurona-pobjednika i njemu susjednih neurona iterativno se mijenjaju u skladu sa pravilom: $m_i(t+1) = m_i(t) + \varepsilon(t) \cdot h_{ci}(t) \cdot (x(t) - m_i(t))$, gdje je $\varepsilon(t)$ tempo obučavanja i monotono je opadajuća funkcija vremena. Često se funkcija susjedstva i tempo

obučavanja kombinuju u jednu funkciju koja se naziva jezgrom („neighborhood kernel").

2.1.3 Modularnost

Princip modularnosti kod ljudskog mozga moguće je uočiti na različitim nivoima, kako strukturnim tako i funkcionalnim. Na najvišem nivou, mozak je podijeljen na dvije hemisfere međusobno povezane relativno malim brojem veza. Svaka od hemisfera obavlja zadatke nezavisno od druge. Unutar hemisfera, ponovo do izražaja dolazi modularna struktura, jer su pojedine grupe neurona organizovane u slojeve i reaguju na različite spoljne impulse. Na primjer, obrada informacija i reakcija na spoljne vizuelne impulse odvija se istovremeno u više tokova: forma objekta, boja, njegovo kretanje i pozicija obrađuju se paralelno u anatomski razdvojenim dijelovima mozga. Funkcionalna prednost anatomskog razdvajanja različitih funkcija ogleda se u minimizaciji interferencije simultanih procesa i izvršavanju više različitih složenih zadataka istovremeno. Modularne neuronske mreže (slika 2.9) su pokušaj modeliranja ovakvog ponašanja ljudskog mozga. Nemodularne mreže u praksi pokazuju slabije rezultate nego modularne mreže.

Pored boljih performansi, primjena modularnih neuronskih mreže vodi ka povećanju nivoa skalabilnosti, tj. veličina koda potrebnog za specifikaciju mreže ne raste proporcionalno sa rastom same mreže.

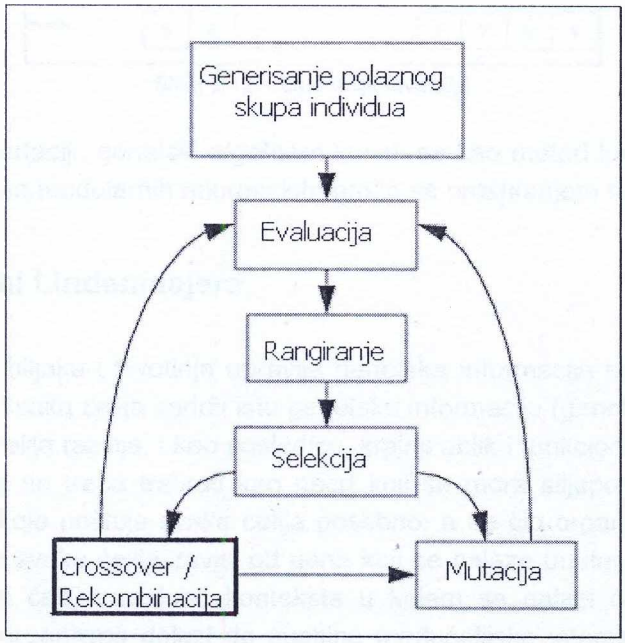


Slika 2.9 – Primjer modularne neuronske mreže

2.1.4 Genetski algoritmi

Darvinovska evolucija, kao važna biološka metafora, modelira se putem genetskog algoritma [Mic98]. Genetski algoritam je metoda pretraživanja zasnovana na principu prirodne selekcije u genetici. Osnovna uloga genetskog algoritma je optimizacija određenog skupa parametara, gdje svaki parametar predstavlja jedno moguće rješenje zadatog problema. Kvalitet predloženog rješenja određuje se

vrijednošću funkcije kvaliteta („fitness“ funkcije). Princip rada genetskog algoritma prikazan je na slici 2.10.

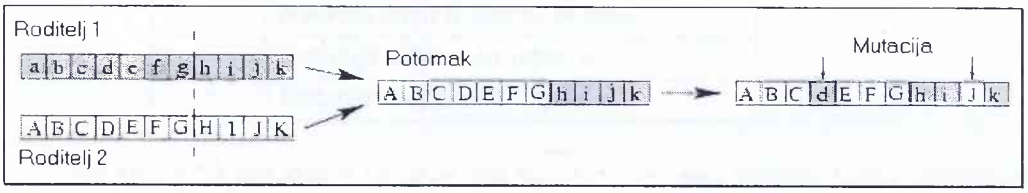


Slika 2.10 – Princip rada genetskog algoritma

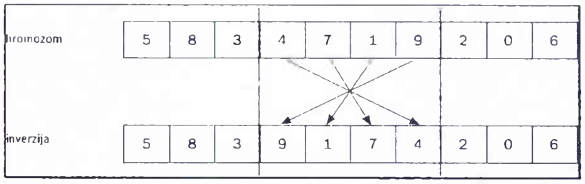
Kodiranjem instanci zadatog problema dobija se polazna populacija ili skup hromozoma, koji se primjenom genetskih operacija selekcije, mutacije, inverzije i rekombinacije ili „crossover“-a transformiše u novu populaciju, koja može da ima isti ili veći broj hromozoma od polazne.

Operacije se izvršavaju nezavisno jedna od druge na različitim hromozomima, pa je moguća efektivna paralelizacija. Svaka od ovih operacija izvršava se sa određenom vjerovatnoćom. Konkretna implementacija svake operacije zavisi od tipa kodiranja instanci problema. Najčešće se koriste bitovi, ali postoje i kodiranja koja koriste realne brojeve, stringove i drveta [Mic98].

Pojednostavljeni grafički prikaz operacija rekommbinacije, mutacije i inverzije prikazan je na slikama 2.11 i 2.12.



Slika 2.11 – Operatori rekombinacije i mutacije



Slika 2.12 – Operacija inverzije

U ovoj disertaciji, genetski algoritam koristi se kao metod lokalnog traženja u prostoru arhitektura modularnih neuronskih mreža sa prostiranjem signala naprijed.

2.1.5 Sistemi Lindenmajera

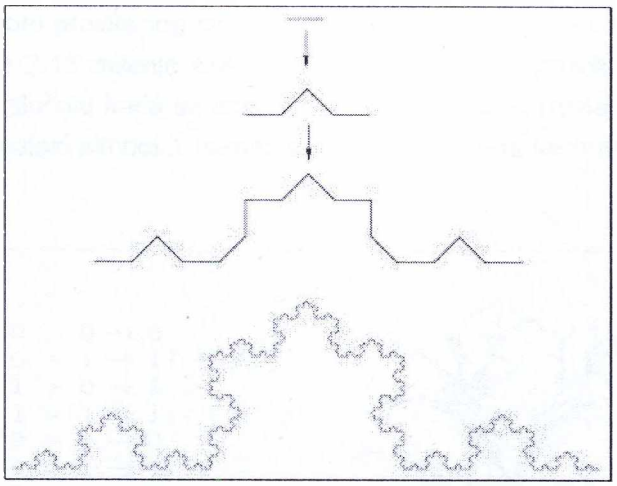
Razvojem biljaka i životinja upravlja genetska informacija sadržana u svakoj ćeliji organizma. Svaka ćelija sadrži istu genetsku informaciju (genotip) koja određuje način na koji se ćelija razvija, i kao posledicu, krajnji oblik i funkcionalnost organizma (fenotip). Genotip ne treba tretirati kao nacrt koji se mora slijepo slijediti, već kao recept ili pravilo koje poštuje svaka ćelija posebno, a ne cio organizam kao cjelina. Oblik i ponašanje svake ćelije zavisi od gena koji se nalaze unutar same ćelije. Koji se geni nalaze u ćeliji zavisi od konteksta u kojem se nalazi ćelija. Na samom početku razvoja organizma dolazi do suptilne međućelijske interakcije koja mijenja skup gena unutar ćelija. Ovaj proces "diferencijacije ćelija" odgovoran je za kreiranje organa [Hyd08]. Matematički model kojim se može modelirati ovakav način razvoja biljaka i životinja su sistemi Lindenmajera ili L-sistemi ([PruLin91], [Bur13]).

Sistemi Lindenmajera (L-sistemi) su prvobitno predstavljali gramatički pristup modeliranju morfogeneze biljaka. Gramatika L-sistema, koja definiše jezik, sastoji se od skupa simbola, skupa pravila (produkcija) i startnog simbola (aksiome). Proces generisanja stringa počinje od startnog simbola (aksiome), zamjenjivanjem svake lijeve strane pravila gramatike odgovarajućom desnom stranom.

Gramatika sa skupom simbola $\{F, +, -\}$, startnim simbolom F i jednim pravilom $F \rightarrow F - F + +F - F$, može generisati poznati fraktal Koha, ako simbole gramatike tretiramo na sljedeći način:

Simbol	Interpretacija
F	Nacrtati liniju u datom pravcu
+	Rotacija ulijevo za ugao φ
−	Rotacija udesno za ugao φ

Na slici 2.13 prikazana su prva dva koraka u razvoju fraktala Koha primjenom date gramatike gdje je ugao rotacije $\varphi = 60^{o}$, a zatim i rezultat poslije 5 koraka.

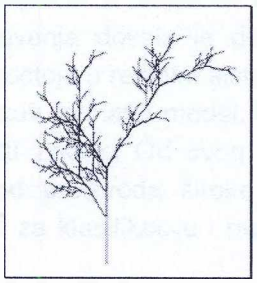


Slika 2.13 – Frakatal Koha generisan primjenom L-Sistema ([PruLin91])

Izražajnost L-sistema povećava se uvođenjem novih struktura podataka i simbola koji predstavljaju nove operacije, kao što su dodavanje steka i operacija sa stekom (*push* i *pop*) ili pomjeranje bez crtanja linije. Na primjer, uvođenje steka dozvoljava da operacije sa stekom simuliraju dijeljenje ćelija kod živih organizama. Novi simboli *f*, [i], najčešće se interpretiraju na način opisan tabelom:

Simbol	Interpretacija
f	Pomjeranje u datom pravcu, bez crtanja linije
[	Dati pravac i poziciju staviti na stek (*Push*)
]	Dati pravac i poziciju ukloniti sa steka (*Pop*)

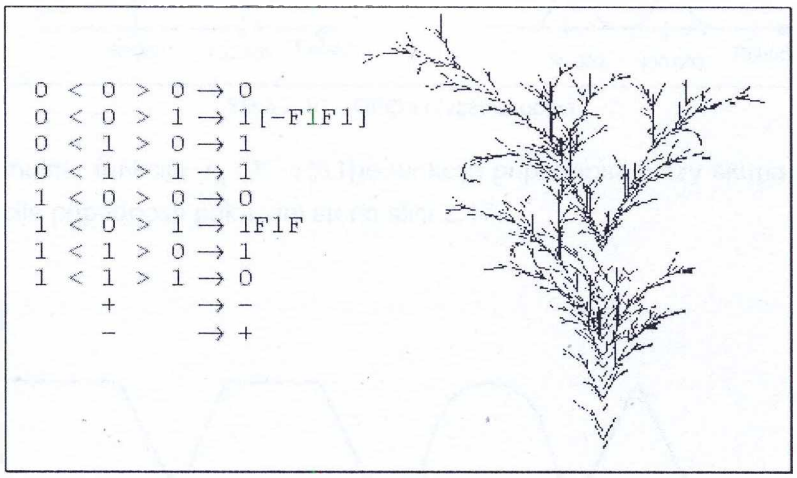
Interpretirajući simbole na način opisan u tabeli, mogu se dobiti fraktali slični onom sa slike 2.14.



Slika 2.14 – L-Sistem sa stekom, ugao rotacije $\varphi = 33^{o}$, 5 iteracija, pravila $X \to F[-[[X]+X]+F[+FX]-X$ i $F \to FF$, aksioma X

Dalje povećanje izražajnosti L-sistema dobija se uvođenjem lijevog i desnog konteksta. Pravilo gramatike se primjenjuje samo ako se desna strane pravila nađe u datom kontekstu. Na slici 2.15 prikazan je jedan kontekstni L-sistem i primjer fraktala

dobijenog primjenom pravila tog sistema. Na primjer, pravilo `0<0>1 → 1[-F1F1]` gramatike sa slike 2.15 mijenja simbol `0` desnom stranom pravila tj. stringom `1[-F1F]`, ali samo u slučaju kada se ispred simbola `0` nalazi `0` (označeno sa `0<`) i ako se iza simbola `0` nalazi simbol `1` (označeno sa `>1`), tj. kada se `0` nađe u „kontekstu" između `0` i `1`.



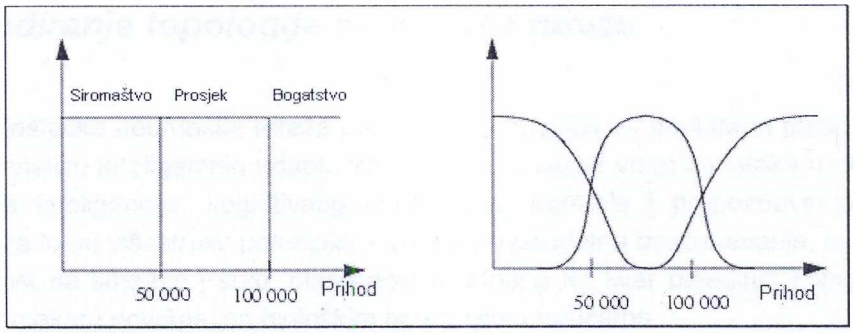
Slika 2.15 – Kontekstni L-sistem i fraktal dobijen poslije 30 iteracija, sa uglom rotacije 16° i aksiomom F1F1F1

Sistemi Lindenmajera su u ovoj disertaciji upotrijebljeni kao metod za kodiranje pravila kreiranja topologije neuronske mreže.

2.1.6 Fazi logika

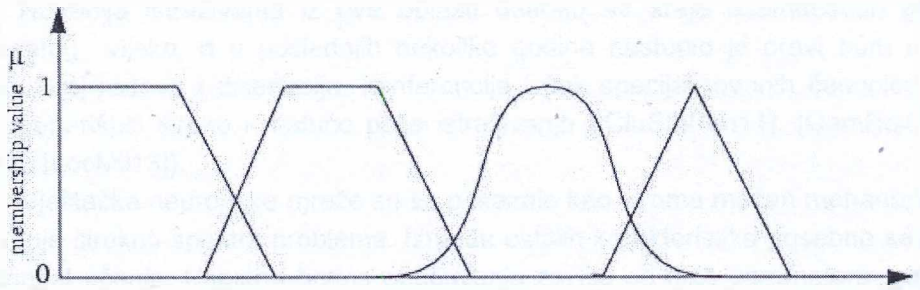
U svakodnevnom rasuđivanju i izvođenju zaključaka, čovjek veoma često donosi odluke na osnovu nepotpunih ili nejasnih činjenica. Primjena računara u mnogim oblastim ljudskog djelovanja dovela je do potrebe da se nepreciznosti, nepotpunosti ili nejasnoće koje postoje u realnim sistemima izraze na formalan način. Fazi logika predstavlja jedan takav formalni model, koji datira još iz 1965. godine i čijim se osnivačem smatra Lofti Zadek. Od svog nastanka, ova teorija je našla značajne praktične primjene, od proizvoda široke potrošnje pa do automatskih sistema za upravljanje i sistema za klasifikaciju i raspoznavanje oblika ([PalBez92], [Oya95]).

U osnovi ove teorije nalaze se pojmovi fazi-skupa i funkcije pripadnosti koja određuje stepene pripadnosti elementa x skupu A ili stepen do kojeg je x uključen u skup A. Na slici 2.16 prikazana je razlika između običnih i fazi skupova.



Slika 2.16 – Obični i fuzzy skupovi

Formalno, funkcija $f_A : X \rightarrow [0,1]$ je funkcija pripadnosti fuzzy skupa A. Različiti tipovi funkcija pripadnosti prikazani su na slici 2.17.

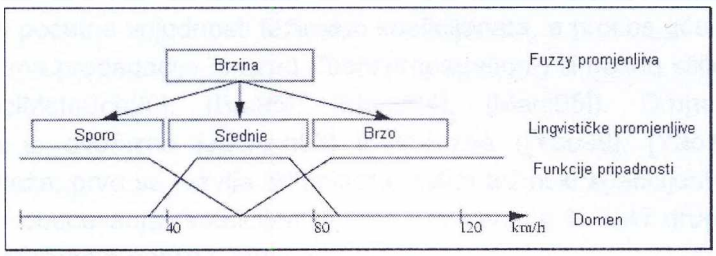


Slika 2.17 - Različiti oblici funkcija pripadnosti

Pojmovi u realnom životu često se daju opisno ili neprecizno, što se u fazi logici reguliše uvođenjem lingvističkih promjenljivih. Pretvaranje opisnih pojmova u brojne vrijednosti obavlja se pomoću funkcije pripadnosti (slika 2.18).

U zadacima klasifikacije, fazi logika se koristila u raznima fazama: za transformaciju ulaznih ili izlaznih podataka ili u samom algoritmu klasifikovanja. U praksi su poznati i mnogobrojni primjeri takozvanih neuro-fazi sistema, koji kombinuju različite tipove obučavanja neuronskih mreža sa fazi teorijom ([BarBlo98], [Fri97], [MitPal94]).

U trećem poglavlju ove teze predloženo je jedno poboljšanje algoritma fazi klasifikacije primjenom samoorganizujućih mreža izloženog u [MitPal94].



Slika 2.18 - Osnovni pojmovi fuzzy teorije

2.2 Kodiranje topologije neuronske mreže

Vještačke neuronske mreže predstavljaju atraktivnu i popularnu paradigmu za dizajn i analizu inteligentnih adaptivnih sistema za razne vrste zadataka u oblastima vještačke inteligencije, kognitivnog modeliranja, kontrole i prepoznavanja oblika. Razlozi za to su višestruki: potencijal za masivna paralelna izračunavanja, robustnost i otpornost na smetnje i šum, elastičnost u odnosu na kvar pojedinih komponenti i sličnost, makar i površna, sa biološkim neuronskim mrežama.

Primjena genetskih algoritama i evolucionog programiranja predstavlja veoma aktivno polje istraživanja u oblasti računarskih nauka. Sama ideja razvoja vještačkih neuronskih mreža primjenom evolucionih strategija, a posebno genetskih algoritama, zasniva se na veoma jasnoj i opšte poznatoj biološkoj metafori - evoluciji ljudskog mozga i drugih bioloških neuronskih mreža.

Najranija istraživanja iz ove oblasti datiraju sa kraja osamdesetih godina dvadesetog vijeka, a u poslednjih nekoliko godina nastupio je pravi bum u ovoj oblasti. Broj radova i disertacija, konferencija i čak specijalizovanih časopisa brzo raste, generišući široko i rastuće polje istraživanja ([CluStaPen11], [CamRoiOli11], [Loc12], [LocMii13]).

Vještačke neuronske mreže su se pokazale kao veoma moćan mehanizam za rješavanje širokog spektra problema. Između ostalih karakteristika posebno se ističe mogućnost učenja. Uspjeh i brzina obučavanja zavise od niza parametara: polazne arhitekture mreže, početnih težinskih koeficijenata, koeficijenta tempa obučavanja, itd. Genetski algoritam se najčešće koristi za pronalaženje optimalnih vrijednosti parametara za dati problem. Generalno gledano, postoje dvije vrste izazova prilikom primjene genetskih algoritama u procesu generisanja neuronskih mreža: kako generisati odgovarajuću topologiju tj. graf koji reprezentuje čvorove-neurone i veze između njih i kako odrediti težinske koeficijente. U manjem broju radova problem izbora koeficijenta tempa obučavanja mreže takođe se ubraja ili u izbor težinskih koeficijenata [Yao99], [Yao10] ili čak se tretira kao dio arhitekture mreže.

Najveći problem koji se pojavljuje u ovoj oblasti istraživanja je način kodiranja vještačkih neuronskih mreža u obliku pogodnom za primijenu genetskog algoritma ili neke druge evolucione strategije. Manji dio predloženih metoda posvećen je pitanju izbora težinskih koeficijenata [WhiStaBog90], dok je najveći dio istraživanja usmjeren na razvoj arhitekture [SchJooWer93]. Mogući su različiti oblici kombinovanja: GA može postaviti početne vrijednosti težinskih koeficijenata, a proces učenja se obavlja putema algoritma propagacije unazad ("backpropagation") ili nekog sličnog algoritma ([Ang94], [BelMcInSch90], [Bra95], [Mand94], [Mani95]). Druga mogućnost kombinovanja je dvofazna [JacReh93] ili trofazna ([Yao99], [Yao10]) evolucija neuronskih mreža: prvo se razvija arhitektura, zatim težinski koeficijenti i na kraju se razvija pravilo obučavanja (koeficijent tempa obučavanja ili neki drugi parametar). Najčešće se ove faze prepliću u jednu cjelinu.

Postoji nekoliko tehnika kodiranja povezanosti mreže. Najprostija od njih je jednobitno kodiranje – postojanje ili nepostojanje veze između neurona [WhiStaBog90]. Nešto složeniji je dvobitni model, koji daje mogućnost kodiranja nepostojanja veze, inhibitorne ("inhibitory") ili pobuđujuće ("excitatory") veze ([JacReh93], [KarDasWhi92]). Težinski koeficijenti mogu biti kodirani kao realni brojevi ([MonDav89], [Mon91]), nizovi (stringovi) bitova ([MilTodHed89], [WhiStaBog90]) ili bit-stringovi koji predstavljaju Grejov kod ("Grey encoding") [Koe94]. U [WhiStaBog90] je predloženo da se broj bitova potrebnih za kodiranje mijenja u toku evolucije, dozvoljavajući finije podešavanje u kasnijim fazama razvoja. Eksperimentalni rezultati pokazuju da povećanje broja bitova potrebnih za kodiranje težinskih koeficijenata poboljšava proces treniranja mreže ako se u tu svrhu koristi genetski algoritam.

Kodiranje topologije (strukture, arhitekture) mreže je mnogo kompleksniji zadatak od kodiranja težinskih koeficijenata. Klasični pristup koji hromozome tretira kao bit-stringove ne predstavlja najpogodniji okvir za složene strukture veza neuronskih mreža, pa se većina istraživača odlučuje za kodiranje koje bolje oslikava samu složenost problema. Načine kodiranja možemo grubo podijeliti u dvije osnovne grupe: direktno i indirektno kodiranje. Metodi direktnog kodiranja mogu se dalje podijeliti u klase u zavisnosti od najmanje jedinice kodiranja, koja može biti neuron, veza između dva neurona, nivo ili putanja u mreži. Indirektni metodi kodiranja koriste činjenicu da se neuronska mreža ne mora predstavljati samo matricom konektivnosti ili grafom, nego i skupom pravila koja generišu mrežu. Kodiranjem su obuhvaćena pravila za generisanje neuronske mreže, a ne sama neuronska mreža.

Opšta podjela metoda direktnog kodiranja ima sljedeći oblik:

- Metodi kodiranja veza između neurona – najveći dio ranih pokušaja pripada ovoj klasi ([Koe94], [Mani93], [Mar92], [MonDav89], [WhiStaBog90]). Najčešće je hromozom string nastao spajanjem težinskih koeficijenata ili matrice konektivnosti, što zahtijeva fiksiranu arhitekturu koja je u nekom smislu "maksimalna" tj. potpunu povezanu arhitekturu ili višeslojnu arhitekturu sa punom povezanošću slojeva.
- Metodi kodiranja neurona (čvorova) – ovaj metod omogućava veću fleksibilnost nego kodiranje veza. Hromozom je string ili drvo informacije o čvorovima [KozRic91] i može u sebi sadržati relativnu poziciju čvora, informaciju o povezanosti unazad ("backward connectivity" [SchJooWer93]), težinske koeficijente, itd. Genetski operatori ukrštanja i mutacije obično su ograničeni na rez između informacija o čvorovima.
- Metodi kodiranja slojeva mreže – primjenom ovih metoda ([HarSma91], [Mand93]) mogu se dobiti mreže sa većim brojem neurona nego kod ostalih metoda. Sam metod kodiranja je veoma složen sistem kodiranja veza između slojeva, što zahtijeva razradu specijalnih genetskih operatora.

- Metodi kodiranja putanja u mreži – ovi metodi tretiraju mrežu kao skup putanja od ulaznih do izlaznih čvorova ([JacReh93], [LukSpe96]). Kao i prethodnom slučaju, potrebni su specijalni genetski operatori.

Prvi pokušaji indirektnih metoda kodiranja pojavljuju se još početkom devedesetih godina dvadesetog vijeka. Pionirski rad iz ove oblasti je [Kitano90], kojem je autor generisao topologiju neuronske mreža primjenom sistema za generisanje grafova. Osnovna ideja je da se kodiraju pravila koja generišu mrežu, a ne sama mreža. Dva najpopularnija metoda su zasnovana na gramatičkim sistemima generisanja fraktala (L-sistemi ili sistemi Lindenmajera [BoeKui01]) i na ćelijskom kodiranju ("cellular encoding" [Gru94]).

Veliki broj radova posvećenih ovoj temi u nekoliko poslednjih godina pokazuje atraktivnost ovog polja istraživanja ([TsiGavGla08], [Yao10], [RocCorNev07], [RadHin13], [MaqKhaAbr07], [LocMii13], [JunReg08], [FloDurMat08]). Osnovno pitanje je da li evolucija težinskih koeficijenata uporedo sa arhitekturom mreže ima prednosti u odnosu na razvoj težinskih koeficijenata nad fiksiranom arhitekturom (bez obzira na koji je način arhitektura dobijena)? Struktura mreže ima uticaja na proces obučavanja, posebno ako se kao algoritam obučavanja koristi propagacija greške unazad ("backpropagation"). Ako se za obučavanje koristi genetski algoritam, tada odgovor ne može biti tako decidan. Jedno rješenje ovog problema može biti inkrementalni razvoj [CluStaPen11], gdje se minimizuje prostor stanja težinskih koeficijenata. Povećanje brzine postiže se minimizacijom strukture tokom čitavog procesa evolucije, a ne na samom njenom kraju, kako je to uobičajeno kod ostalih metoda ove klase ([StaMii02], [StaMii03], [Sta04]). Takođe je moguće koristiti dualnu reprezentaciju grafa koji predstavlja strukturu mreže [PujPol98], ali se postavlja problem prostorne složenosti takve reprezenatcije.

Veličina mreže utiče na veličinu hromozoma, i na taj način i na efikasnost genetskog algoritma ([Whi93], [RocCorNev07]). Kompleksnije metode kodiranja su direktna posledica ove činjenice. Ove metode dovode i do zamjene standardnih genetskih operatora mutacije i ukrštanja operatorima tipa "slučajno dodaj neuron" [SchJooWer93] ili "obriši putanju" [Kitano90].

L-Sistemi se mogu upotrebiti za razvoj vještačkih neuronskih mreža ([AhoKemKos97], [BoeKui01], [Chv02], [CamRoi04], [CamRoiOli11]) na način sličan prikazanom u ovoj disertaciji. Nijedan od ovih metoda ne nudi formalnu definiciju same metode niti dokaz da metoda zaista generiše željene topologije.

Nedostaci nabrojanih metoda kodiranja topologije mreže su:

- nedostatak teorijske osnove – veoma je rijetka praksa da postoje opšti principi koje moraju zadovoljavati metodi kodiranja. Osnovni principi su najčešće kompletnost (svaka mreža može se kodirati) i zatvorenost (samo NN koje imaju smisla mogu se kodirati). Jedini predlog takve vrste dat je kod ćelijskog kodiranja

[Gru94]. Za neke od karakteristika, kao što je tzv. Boldvinov efekat, i dalje ne postoji odgovarajuća teorijska podloga.

- problem permutacija – takođe poznat kao problem strukturno-funkcionalnog preslikavanja („structural-functional mapping") ili problem suprotstavljenih konvencija („competing conventions") [Hay04]. U slučaju kodiranih mreža, ovaj problem se ogleda u tome što dvije slične mreže mogu biti kodirane potpuno različitim kodovima. Primjena operatora ukrštanja može kreirati mrežu koja je potpuno divergentna. Predloženo je više načina rješavanja ovog problema [Han92], [KarDasWhi92], [SriPat91].
- parametri genetskog algoritma i neuronske mreže – postoji veliki broj parametara koje treba podesiti za uspješan rad: veličina populacije, vjerovatnoća mutacije i ukrštanja, način izračunavanja „fitness" funkcije, broj iteracija u obučavanju mreže, tempo obučavanja mreže, aktivaciona funkcija neurona, itd. Da li svaki od ovih parametara zavisi od konkretnog zadatka? Većina ovih parametara određuje se metodom "rule of thumb" i vremenski dugotrajnim eksperimentima. Moguće je i uključivanje nekog od ovih parametara u hromozom, što povećava veličinu hromozoma i složenost genetskog algoritma [CanKam05]. Na primjer, u hromozom se može uključiti broj bita potrebnih za kodiranje težinskih koeficijenata [Mani94] ili koeficijent tempa obučavanja [Mand93].
- nemogućnost poređenja različitih metoda kodiranja – nedostatak teorijskog okvira ne daje mogućnost za upoređivanje različitih metoda kodiranja. Eksperimentalni rezultati ne potvrđuju da se povećanjem složenosti metoda kodiranja postižu bolje performanse ([FloDurMat08], [StaMii03], [CanKam05]).

Modularne mreže su i prirodan izbor u slučaju kada se proces obučavanja neuronskih mreža obavlja na višeprocesorskim sistemima. Druga biološka metafora, genetski algoritmi (kao matematički model procesa Darvinovske evolucije), upotrebljena je kao optimizator u prostoru arhitektura neuronskih mreža. Izbor genetskog algoritma (ili neke druge evolucione strategije) kao metoda optimizacije zasniva se na veoma jasnoj i opšte poznatoj biološkoj metafori - evoluciji ljudskog mozga. Drugi razlog izbora genetskih algoritama leži u njihovoj podesnosti za paralelizaciju.

Izbor evolucije modularnih neuronskih mreža posljedica je jednostavne biološke činjenice da je sam mozak modularna struktura, u kojoj su pojedine grupe neurona obavljaju tačno definisane zadatke [Hyd08]. Pored toga, sam mozak se razvija kao i sve druge ćelije ljudskog organizma, pa je prirodno da se te dvije biološke paradigme kombinuju. Takođe, pojedini ljudski organi, na primjer pluća, imaju fraktalnu strukturu, što opravdava upotrebu sistema Lindenmajera.

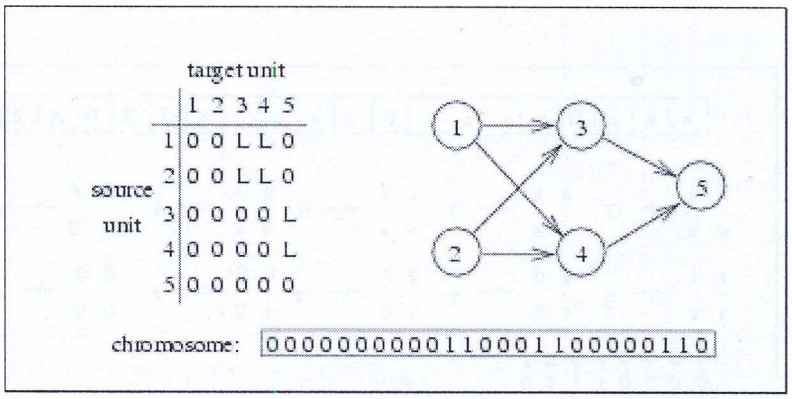
2.3 Evolucioni razvoj topologije neuronske mreže

Poslednjih dvadeset godina razvijeni su mnogi metodi primjene genetskih algoritama i genetskog programiranja u obasti tehnologija neuronskih mreža. Genetski algoritmi, genetsko programiranje i neuronske mreže predstavljaju pokušaj matematičkog modeliranja postojećih bioloških procesa, pa, najčešće, autori tih metoda govore o prirodnom prožimanju tih dvaju bioloških metafora.

Pregledom tih metoda, možemo ustanoviti da postoje tri glavne kategorije primjene genetskih algoritama:

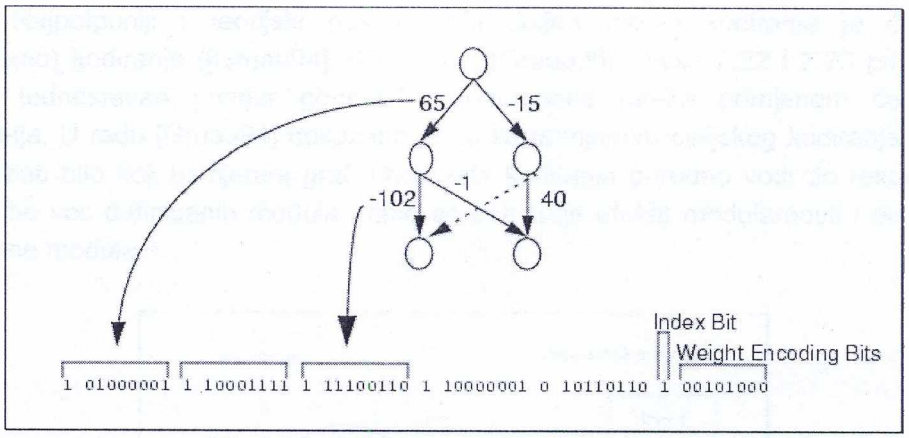
- zadata je arhitektura mreže (tj. veze, broj slojeva i broj neurona u sloju) i pravilo obučavanja mreže, a genetski algoritmi se koriste za određivanje težinskih koeficijenata;
- dato je pravilo obučavanja, a genetski algoritmi se koriste za određivanje arhitekture mreže (konektivnosti neurona);
- data je arhitektura mreže, a genetski algoritmi se koriste za određivanje pravila obučavanja zasnovanog na nekoj funkciji kvaliteta (fitness funkciji).

Najveći dio navedenih metoda vrši direktno kodiranje arhitekture mreže i/ili težinskih koeficijenata, pa su broj hromozoma i njihova dimenzija direktno proporcionalni broju neurona i veza među njima u samoj mreži. Tipičan primjer takvog kodiranja prikazan je na slici 2.19.



Slika 2.19 – Direktno kodiranje mreže [MilTodHed89]

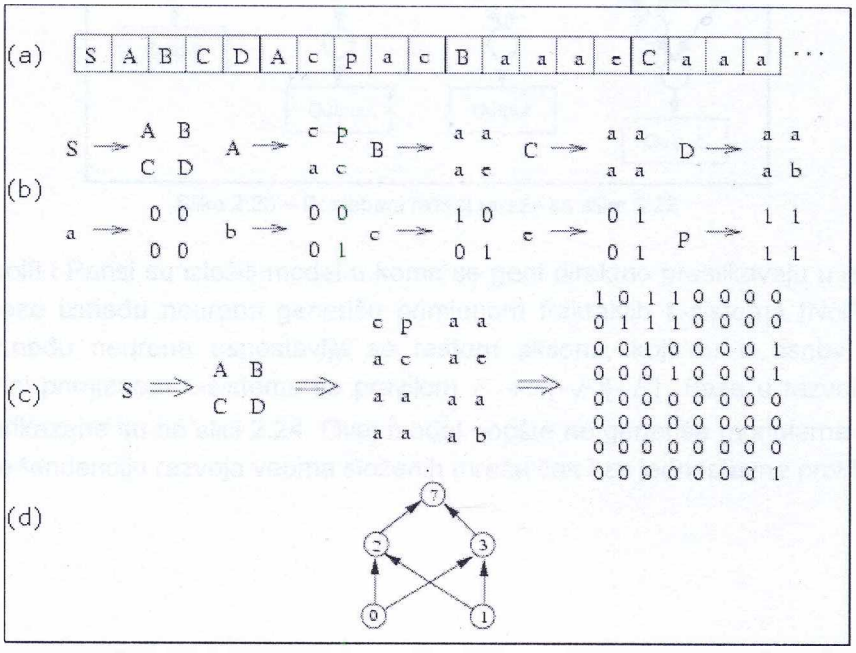
Jedan od prvih, i samim tim najuticajnijih modela, je GENITOR, čiji su osnovni principi kodiranja prikazani na slici 2.20.



Slika 2.20 – GENITOR kodiranje [WhiStaBog90]

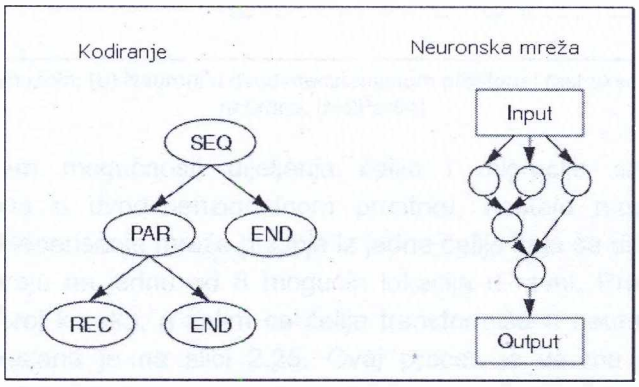
Razmatrajući nedostatke modela sličnih onom sa slike 2.20, veliki broj autora je predložio metode koji ne kodiraju samu mrežu, već niz pravila (recepata) koji generišu ahitekturu mreže i/ili težinske koeficijente. Primjeri takvih metoda su ćelijsko (celularno) kodiranje [Gru94], kodiranje grana grafa [LukSpe96] ili kodiranje pomoću matrica [Kitano90].

Kitano je predložio metod kodiranja zasnovan na fraktalnom predstavljanju mreža pomoću gramatičkih pravila oblika $A \rightarrow H_{2x2}$, gdje je A simbol alfabeta (najčešće malo ili veliko slovo engleske abecede), a H_{2x2} matrica oblika 2x2. Primjer kreiranja matrice konektivnosti koja predstavlja arhitekturu neuronske mreže dat je na slici 2.21.

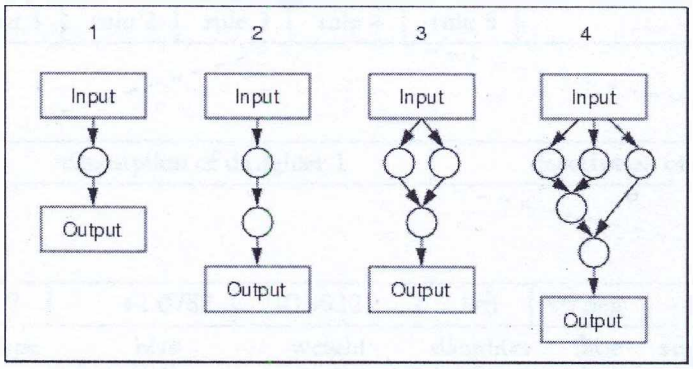


Slika 2.21 – (a) Hromozom; (b) Pravila gramatike; (c) Proces izvođenja; (d) Mreža. [Kitano90]

Najpotpuniji i teorijski najbolje obrazložen model kodiranja je ćelijsko (celularno) kodiranje ([Gruau94], [Gruau95], [Gruau96]). Slike 2.22 i 2.23 prikazuju jedan jednostavan primjer generisanja neuronske mreže primjenom ćelijskog kodiranja. U radu [Gruau94] dokazano je da se primjenom ćelijskog kodiranja može generisati bilo koji usmjereni graf. Ova vrsta kodiranja prirodno vodi do rekurzivne upotrebe već definisanih modula i time se ostvaruje efekat modularnosti i ponovne upotrebe modula.

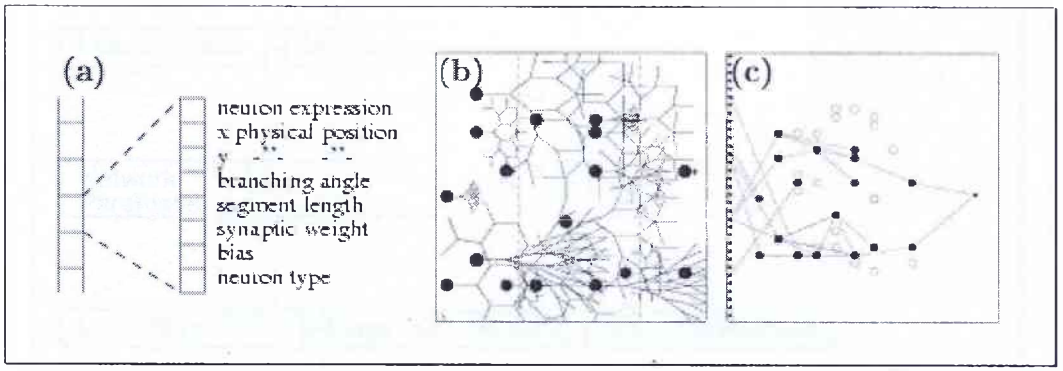


Slika 2.22 – Ćelijski kod i rezultujuća mreža (adaptirano iz [Gruau94])



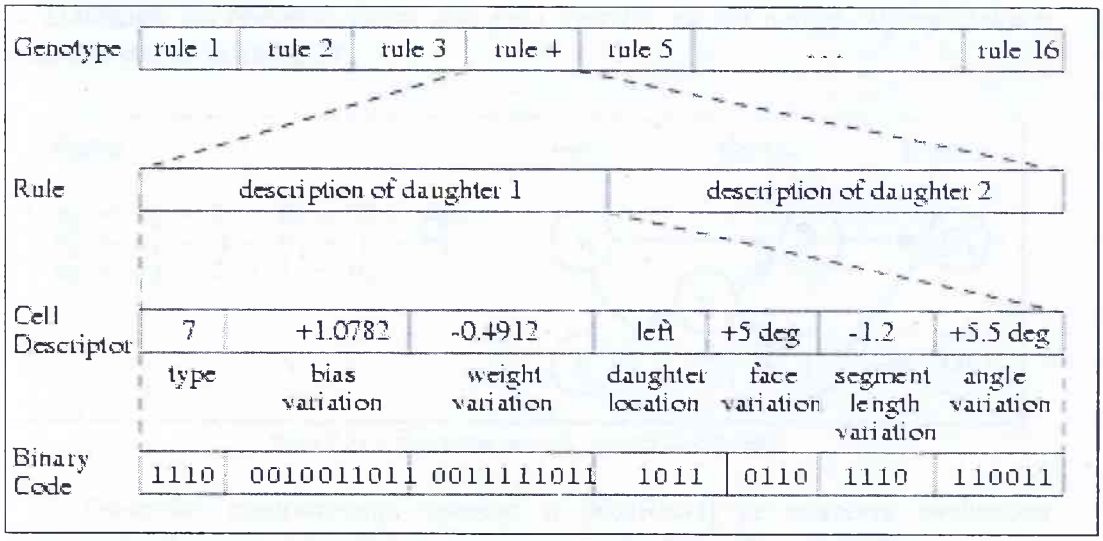
Slika 2.23 – Postepeni razvoj mreže sa slike 2.22

Nolfi i Parisi su izložili model u kome se geni direktno preslikavaju u neurone, ali se veze između neurona generišu primjenom fraktalnih L-sistema [NolPar94] . Veza između neurona uspostavlja se rastom aksona, koji su u osnovi fraktali generisani primjenom L-sistema sa pravilom $F \rightarrow F[-F][+F]$. Faze u razvoju jedne mreže prikazane su na slici 2.24. Ovaj model uopšte ne generiše modularne mreže i pokazuje tendenciju razvoja veoma složenih mreža čak i za jednostavne probleme.



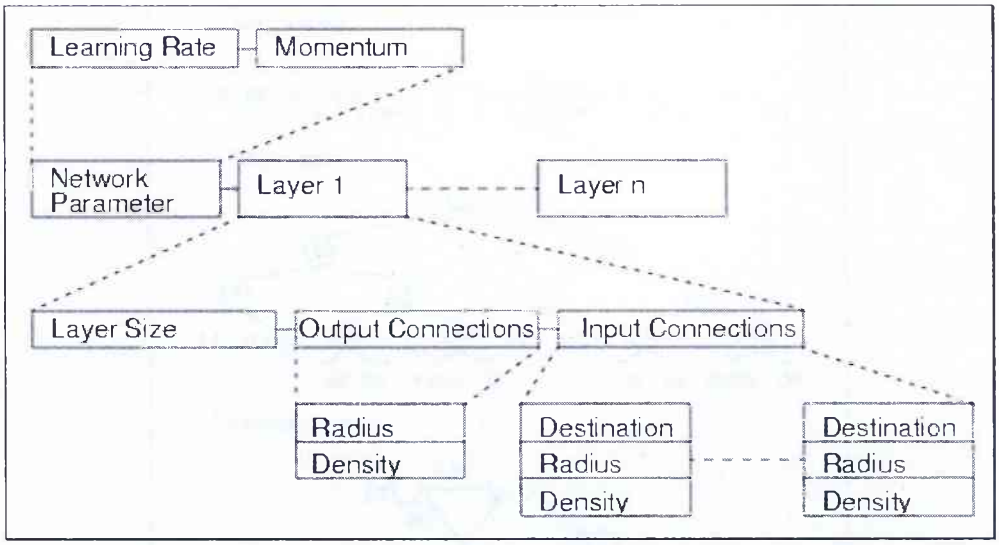
Slika 2.24 – (a) Hromozom; (b) Neuroni u dvodimenzionalnom prostoru i rast aksona; (c) Veze između neurona. [NolPar94]

Dodavanjem mogućnosti dijeljenja ćelija i migracije umjesto direktnog kodiranja neurona u dvodimenzionalnom prostoru, nastaje model predložen u [CanNolPar93]. Generisanje mreže počinje iz jedne ćelije koja se dijeli na dva dijela i koji se pozicioniraju na jednu od 8 mogućih lokacija u ravni. Proces dijeljenja se obavlja fiksiran broj koraka, a zatim se ćelije transformišu u neurone. Organizacija hromozoma prikazana je na slici 2.25. Ovaj proces je veoma dugotrajan zbog složene strukture hromozoma.



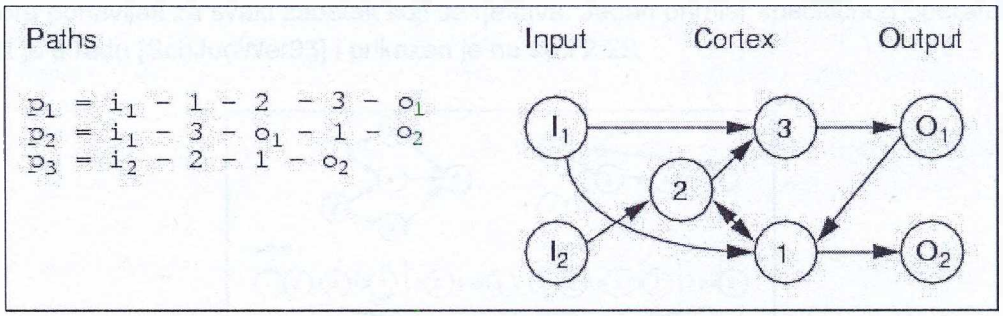
Slika 2.25 – Organizacija hromozoma [CanNolPar93]

Metodi kodiranja slojeva mreže najčešće zahtijevaju čuvanje informacije o veličini sloja i ulaznim i izlaznim vezama sloja. Izlazne veze obezbjeđuju povezanost sloja sa sljedećim susjednim slojem, dok ulazne veze potiču od prethodnih slojeva (označenih sa "destination"). Osnovna struktura jednog modela kodiranja slojeva mreže prikazana je na slici 2.26.



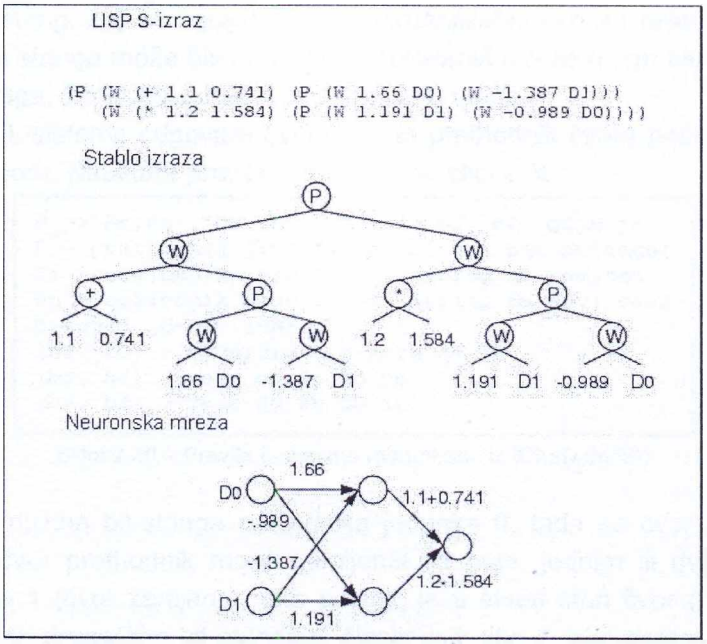
Slika 2.26 – Šema kodiranja slojeva [Mand93].

Kodiranje putanja u mreži predloženo je u radu [JacReh93], gdje se kodiraju sve moguće putanje od ulaznih čvorova mreže do izlaznih čvorova. Ovaj metod kodiranja je pogodan za mreže koje nemaju veliki broj putanja. Dijelovi putanja mogu se preklapati, pa operatori mogu više puta djelovati na isti neuron. Primjer takvog kodiranja dat je na slici 2.27.



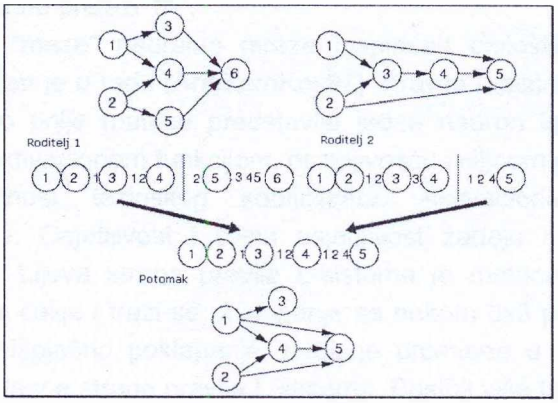
Slika 2.27 – Kodiranje putanja u mreži [JacReh93]

Genetsko programiranje opisano u [KozRic91] je posebna evoluciona strategija bazirana na S-izrazima programskog jezika Lisp. Ova strategija primjenjuje se za generisanje Lisp programa primjenom specijalno definisanih genetskih operatora ukrštanja i mutacije. Genetsko programiranje može se primjeniti za generisanje neuronskih mreža ako se arhitektura i težinski koeficijenti mreže predstave posebnim S-izrazom, kao na slici 2.28.



Slika 2.28 – Generisanje neuronske mreže primjenom genetskog programiranja [KozRic91]

Genetsko programiranje za kodiranje mreže ne koristi bitova ili stringove, pa se moraju se definisati posebni operatori ukrštanja (crossover-a) i mutacije specifični za izabrani način kodiranja, što otežava proces projektovanja mreže. Štaviše, ovo se mora ponavljati za svaki zadatak koji se rješava. Jedan primjer specifičnog operatora dat je u radu [SchJooWer93] i prikazan je na slici 2.29.



Slika 2.29 – Operator ukrštanja [SchJooWer93].

Razvijeno je i više modela koji koriste sisteme Lindenamajera (ili skraćeno L-sisteme) za generisanje arhitekture neuronske mreže.

Channon i Damper su u [ChaDam98] istraživali mogućnosti razvoja novog ponašanja vještačkih bića (*Artificial Life Creatures*) kroz evoluciono razvijanje neuronske mreže čija je arhitektura kodirana primjenom L-sistema. Svaki čvor ima

pridruženi bit-string, koji se inicijalizuje pri konstruisanju čvora i mijenja kroz razvoj mreže. Dužina stringa može biti proizvoljna, a čvorovi mreže mogu biti ulazni, izlazni ili nijedno od toga, što zavisi od aksioma i razvoja mreže.

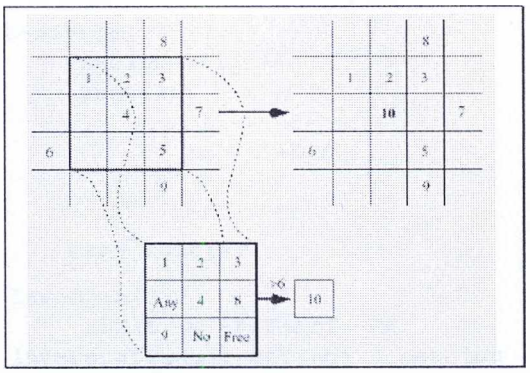
Pravilo L-sistema odgovara čvoru ako je prethodnik čvora početak bit-stringa pridruženog čvoru. Struktura pravila opisana je na slici 2.30:

```
P -> Sr,Sn : b1, b2, b3, b4, b5, b6  gdje je:
P - prethodnik (inicijalni bitovi bit-stringa)
Sr - naslednik tipa 1: bit-string za zamjenu
Sn - naslednik tipa 2: bit-string za novi cvor
bitovi: (0=DA, 1=NE)
(b1, b2) - reverzni tip veza za Sn
(b3, b4) - veze od Sr do Sn
(b5, b6) - veze od Sn do Sr
```

Slika 2.30 – Pravila L-sistema (adaptirano iz [ChaDam98])

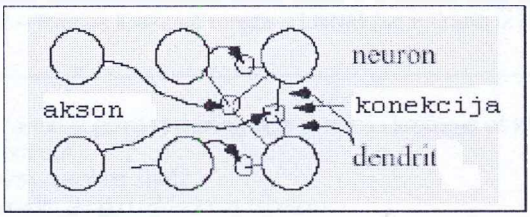
Ako je dužina bit-stringa naslednika jednaka 0, tada se čvor ne kreira, što znači da se čvor prethodnik može zamijeniti sa nula, jednim ili dva nasljednika. Nasljednik tipa 1 (čvor zamjene), ako postoji, je u stvari stari čvor (prethodnik) sa istim vezama ali drugačijim bit-stringom. Nasljednik tipa 2, ako postoji, je novi čvor, koji nasljeđuje kopiju svih ulaza starog čvora, osim ako već ne postoji veza od starog čvora (b3 ili b4) i kopiju svih izlaza starog čvora, osim ako već ne postoji veza ka starom čvoru (b5 ili b6). Novi ulazni čvorovi se kreiraju samo od ulaznih čvorova mreže, dok se novi izlazni čvorovi kreiraju samo od izlaznih čvorova. Poklapanje zasnovano na bit-stringovima osigurava da dodavanje ili uklanjanje čvorova neće pokvariti veze koje su već uspostavljene razvojem mreže. Predloženi model nije skalabilan i pokazuje dobre rezultate samo za mreže male ili srednje veličine, u kojima broj konekcija ne prelazi 10^4.

Metod rasta "mase" neuralne mreže inspirisan biološkim rastom i vođen L-sistemima predložen je u radu [AhoKemKos97]. Pravila L-sistema primjenjuju se u matrici ćelija. Svako polje matrice predstavlja jedan neuron koji se opisuje sa 4 atrubuta: težinom, aktivacionom funkcijom, osjetljivošću i ciljnom osjetljivošću. Težina je početna vrijednost težinskog koeficijenta. Aktivaciona funkcija zadaje funkcionalnost ćelije. Osjetljivost i ciljna osjetljivost zadaju način uspostavljanja međućelijskih veza. Lijeva strana pravila L-sistema je matrica 3x3, koja biološki predstavlja susjedne ćelije i traži se poklapanje sa nekom 3x3 podmatricom matrice ćelija (slika 2.31). Uspješno poklapanje uzrokuje promjene u matrici ćelija. Vrsta promjene zavisi od desne strane pravila L-sistema. Postoji više tipova poklapanja, jer se mogu koristiti specijalni karakteri (*wildcards*). Različite načine razvoja ćelija u zavisnosti od okoline modeliraju 4 vrste pravila: And, Or, GreaterThan i LesserThen. Ako dođe do bar jednog poklapanja, promjenljiva koja označava starost pravila se uvećava, tako da pravilo poslije određenog broja koraka "umire", to jest prestaje da se primjenjuje.



Slika 2.31 – Poklapanje u L-sistemu [AhoKemKas97]

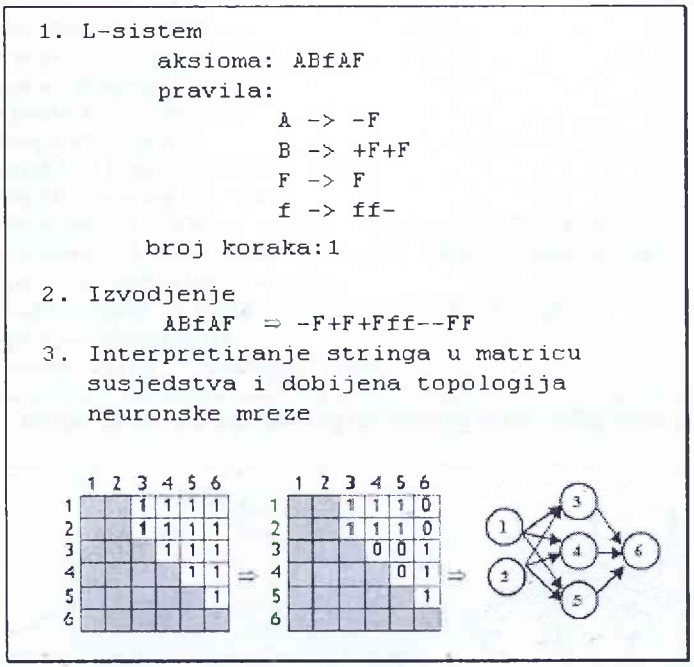
Poslije rasta neurona potrebno je uspostaviti veze između neurona. Svaki neuron prvo izgrađuje jedan akson sa lijeve strane i jedan ili više dendrita sa desne strane. Kada dođe do susreta aksona i dendrita, kreira se konekcija između 2 neurona (slika 2.32). Ovakav način kreiranja veza između neurona garantuje kreiranje feed-forward mreža.



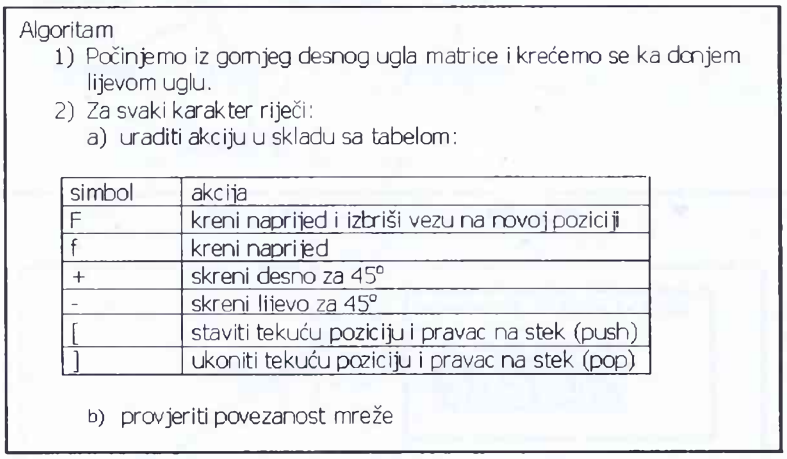
Slika 2.32 – Kreiranje konekcije (adaptirano iz [AhoKAemKas97])

LNNM (L-system Neural Network Modeller) je sistem za pronalaženje najefikasnije arhitekture feedforward mreže primjenom evolucionih algoritama i L-sistema. L-sistem se gradi nad dvije azbuke X_1={A, B, F, f} i X_2={ A, B, F, f ,[,], +, -}, aksioma i 4 pravila. Aksioma je riječ nad azbukom X_1 dužine ne veće od 4. Lijeva strana pravila je riječ nad azbukom X_1, a desna strana pravila je riječ nad azbukom X_2. Poslije unaprijed definisanog broja koraka izvođenja, generisana riječ se prevodi u matricu povezanosti.

Evoluciono izračunavanje predstavljeno je primjenom genetskog algoritma, a jedini genetski operator koji je implementiran je mutacija. Zbog specifičnog oblika kodiranja, uvedeni su posebni oblici mutacije, kao što su izbacivanje, umetanje ili mijenjanje jednog simbola u aksiomi ili u pravilu i promjena broja koraka izvođenja ili broja neurona u mreži. Proces kreiranja mreže u LNNM sistemu dat je na slici 2.33, a algoritam prevođenja stringa u matricu prikazan je na slici 2.34.



Slika 2.33 – Proces kreiranje mreže u LNNM (adaptirano iz [Chval02])



Slika 2.34 – Algoritam prevođenja stringa u matricu (adaptirano iz [Chval02])

Metod LNNM koristi istu kombinaciju bioloških metafora koja je opisana i u ovoj disertaciji. Kodiranje mreže se obavlja preko matrice, što značajno usporava cio proces. Nisu jasni kriterijumi koji određuju kada se mijenjaju parametri mreže ili kada se npr. dodaje neuron.

U radu [Huss98] predloženo je da se arhitektura mreže generiše primjenom atributivnih gramatika, gdje se kao atributi zadaju svojstva mreže. Primjer jednostavne atributivne gramatike dat je na slici 2.35, dok su drvo parsiranja i rezultujuća arhitektura neuronske mreže prikazani na slikama 2.36 i 2.37:

Neterminalni simboli: (sintet. atrib.) {naslijeđ. atrib.}

<Network> : (all_nodes, all_connections) { }

<Layer> : (nodes) { }

Terminalni simboli: (sintet. atrib.)

<perceptron> : (spec)

Pravila gramatike i izračunavanje atributa:

<Network> → <Layer>$_1$ <Layer>$_2$

<Network>.all_nodes := <Layer>$_1$.nodes ∪ <Layer>$_2$.nodes

<Network>.all_connections := <Layer>$_1$.nodes × <Layer>$_2$.nodes

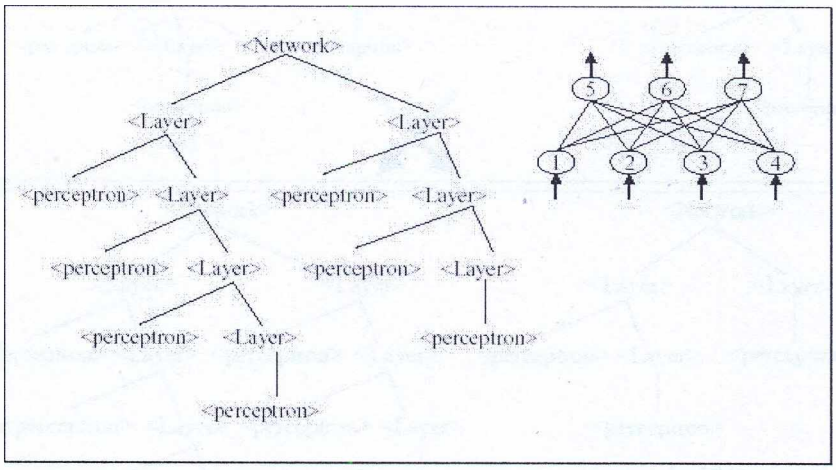
<Layer>$_1$ → <perceptron> <Layer>$_2$

<Layer>$_1$.nodes := <Layer>$_2$.nodes ∪ <percepron>.spec

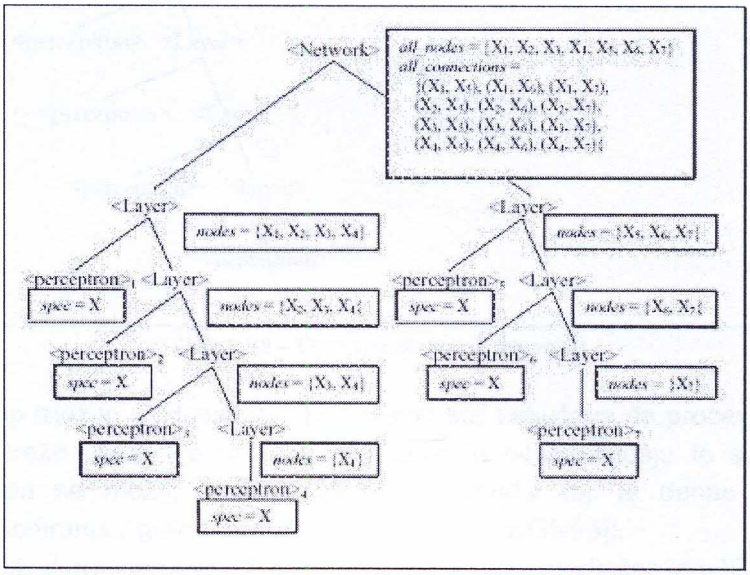
<Layer> → <perceptron>

<Layer>.nodes := <percepron>.spec

Slika 2.35 - Primjer atributivne gramatike za generisanje mreže (adaptirano iz [Huss98])

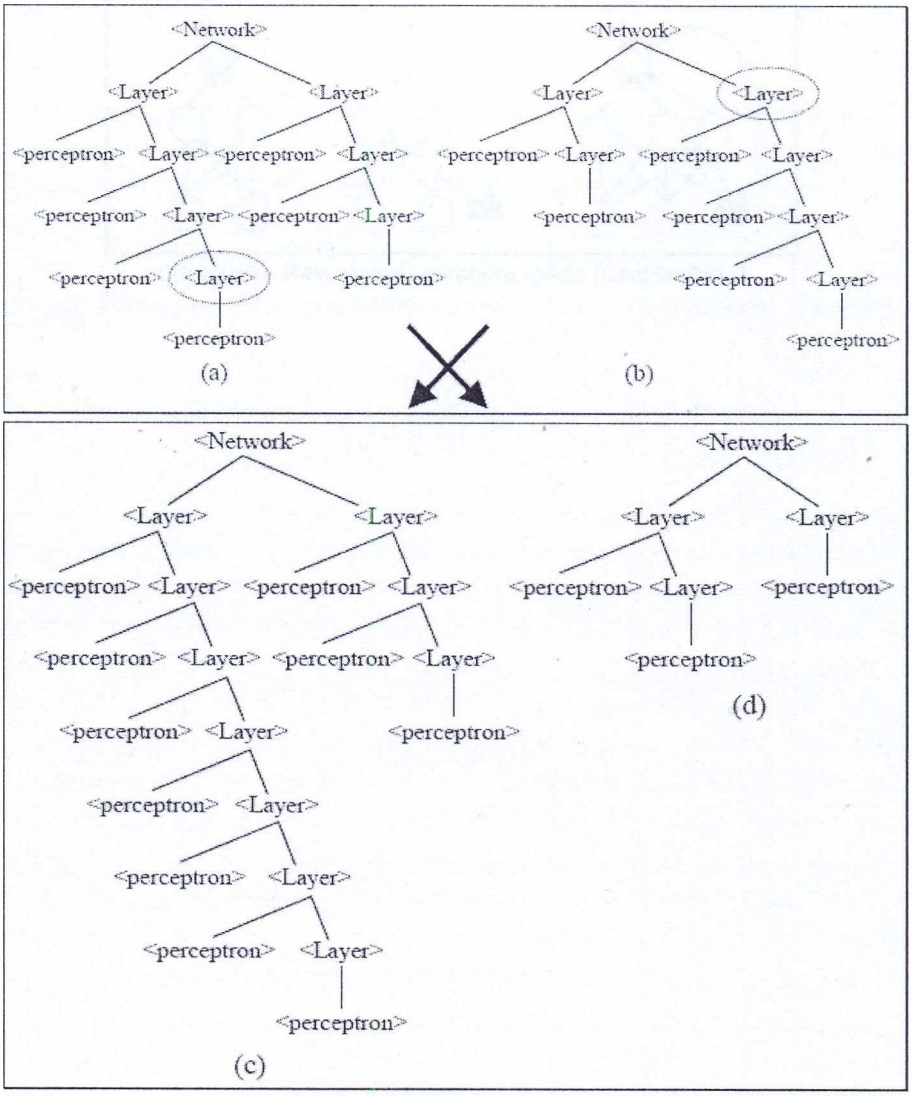


Slika 2.36 - Drvo parsiranja [Huss98]



Slika 2.37 – Drvo parsiranja [Huss98]

Genetski operatori se definišu na drvetu parsiranja, na sličan način kao kod genetskog programiranja. Na primjer, na slici 2.38 prikazan je operator ukrštanja (crossover), gdje se ukrštanjem mreža definisanih drvetima (a) i (b) dobijaju 2 nova drveta (c) i (d).

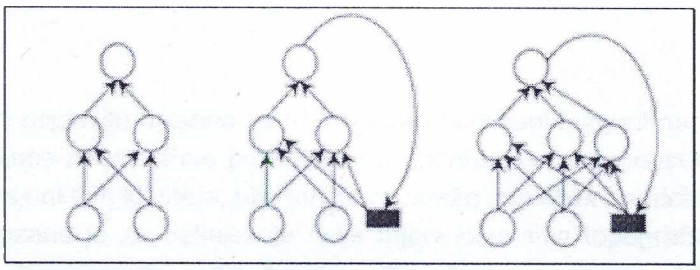


Slika 2.38 – Operator ukrštanja [Huss98]

Pristup izložen u [Huss98] je prvi metod koji pokušava da proces generisanja topologije mreže postavi u neki poznati okvir, u ovom slučaju to su atributivne gramatike, pa se može smatrati pretečom metoda koji je danas poznat kao gramatičko kodiranje ("grammatical encoding", [TsiGavGla08]).

Metod, skoro identičan predloženom u ovoj tezi, predložen je u [CamRoiOli11] za generisanje rekurzivnih neuronskih mreža. Implementacija se, za razliku od ove

teze, i dalje zasniva na matricama, što utiče na dugotrajnost procesa nalaženja odgovarajuće topologije. Generisane su mreže za problem parnosti („parity problem") i prepoznavanja malih regularnih jezika, a prvi put je metod iskorišćen i za predviđanje vremenskih serija. Primjer generisanja rekurentne neuronske mreže prikazan je na slici 2.39.



Slika 2.39 – Rekurentne neuronske mreže [CamRoiOli11]

3. Predlog poboljšanja algoritama za obučavanje samoorganizujućih mreža

U ovom poglavlju opisana su tri moguća poboljšanja algoritma za obučavanje samoorganizujuće mreže. Prvo poboljšanje se odnosi na mogućnost obučavanja na nivou grupe disjunktnih klastera, ako struktura mreže ima oblik paralelopipeda. Drugo poboljšanje vezano je za definisanje dvije mjere očuvanja topografskih i metričkih karakteristika preslikavanja koje definiše algoritam Kohonena. Treći predloženi algoritam kombinuje algoritam Kohonena sa fazi logikom za kreiranje klasifikatora.

3.1 Obučavanje mreže po disjunktnim klasterima

U ovom dijelu rada predložen je metod obučavanja samoorganizujuće mreže modificiranom varijantom algoritma Kohonena. Modifikacija se odnosi na distribuciju neurona po disjunktnim klasterima i obučavanju na nivou klastera. Kod tradicionalnih metoda obučavanja ovih mreža algoritmom Kohonena, obučavanje se vrši na nivou neurona, pa je vremenska složenost takvih algoritama reda O(*mN*), gdje je *m* broj vektora u skupu za obučavanje mreže i *N* broj neurona u mreži. Primjenom obučavanja na nivou klastera vremenska složenost smanjuje se do O(*mlogN*).

Razmatra se neuronska mreža za probleme klasifikacije ili predobrade podataka. Primjenjujemo metoda obučavanja koji kombinuje tradicionalni metod obučavanja neurona u samoorganizujućim mrežama i algoritam za određivanje uzajamnog dejstva N tijela u prostoru (tzv. N-Body algoritam [Aar03]). Pretpostavimo da su neuroni organizovani u obliku d-dimenzionog paralelopipeda i da svakom neuronu (čvoru) $a(i_1,\ldots,i_d)$ odgovara težinski vektor $w(i_1,\ldots,i_d)$, $1 \le i_k \le N_k, k = 1,\cdots,d$.

Podijelimo paralelopiped na disjunktne klastere (grupe) veličine $c_1 \times c_2 \times \cdots \times c_d$ svaki. Indeksi neurona koji pripadaju klasteru $C(j_1,\ldots,j_d)$, $1 \le j_k \le \frac{N_k}{c_k}, k = 1,\cdots,d$, imaju sljedeći oblik:

$$\begin{matrix} (i_1(j_1,1),\cdots,i_{d-1}(j_{d-1},1),i_d(j_d,1))\ldots(i_1(j_1,1),\cdots,i_{d-1}(j_{d-1},1),i_d(j_d,c_d)) \\ \vdots \qquad\qquad \ddots \\ (i_1(j_1,c_1),\cdots,i_{d-1}(j_{d-1},c_{d-1}),i_d(j_d,1))\ldots(i_1(j_1,c_1),\cdots,i_{d-1}(j_{d-1},c_{d-1}),i_d(j_d,c_d)) \end{matrix} \quad (1)$$

Klasteru $C(j_1,\ldots,j_d)$ pripadaju neuroni $a(i_1,\ldots,i_d)$, za koje važi:

$$\forall m \in \{1,\cdots,d\}\ i_m = (j_m - 1)\cdot c_m + 1,\cdots, j_m \cdot c_m .$$

Primjer 1. U slučaju dvodimenzionalne mreže, klasteri imaju sljedeći oblik:

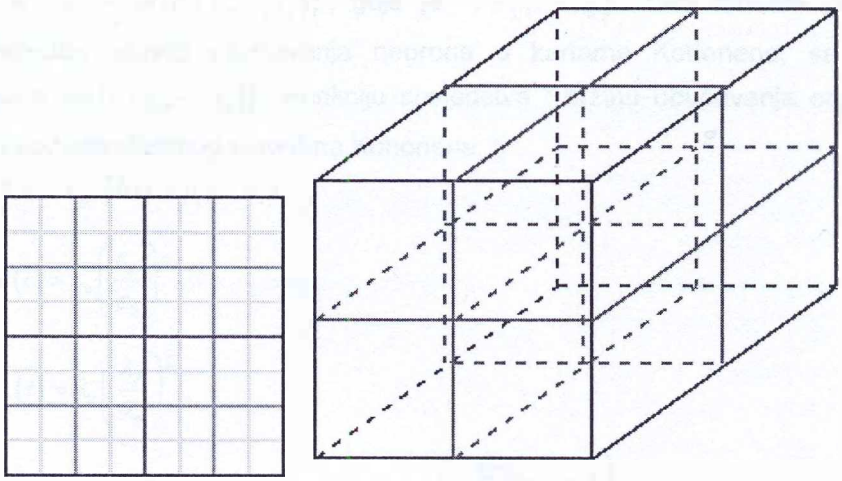
C(1,1)	C(1,2)	...	C(1,c_2)
C(2,1)	C(2,2)	...	C(2,c_2)
⋮	⋮	⋱	⋮
C(c_1,1)	C(c_1,2)	...	C(c_1, c_2)

Svakom od klastera $C(i, j)$ pripada $c_1 \times c_2$ neurona. Vezu koja postoji između susjednih klastera možemo opisati sljedećim jednakostima:

$$\forall k \in \{1,\cdots,c_1\}: (i(j_1,c_1), i(j_2,k)+1) = (i(j_1,c_1), i(j_2+1,k))$$

$$\forall k \in \{1,\cdots,c_2\}: (i(j_1,c_1)+1, i(j_2,k)) = (i(j_1+1,c_1), i(j_2,k)) .$$

Pretpostavimo da je $N_1 = 8, N_2 = 9, c_1 = 2, c_2 = 3$, tada klasteru C(2,3) pripadaju neuroni a(3,7), a(3,8), a(3,9), a(4,7), a(4,8), a(4,9). Na slici 3.1 prikazan je način podjele mreže na klastere u prostorima dimenzije 2 i 3.



Slika 3.1 – Podjela na klastere u slučaju k=2 i k=3

Svakom klasteru $C(j_1,\ldots,j_d)$ pridružimo težinu klastera, tj. aritmetičku sredinu težinskih vektora svih neurona koji pripadaju tom klasteru:

$$w(C(j_1,\ldots,j_d)) = \frac{1}{c_1 \cdot c_2 \cdot \ldots \cdot c_d} \sum_{k_1=1}^{c_1} \sum_{k_2=1}^{c_2} \ldots \sum_{k_d=1}^{c_d} w(i_1(j_1,k_1), i_2(j_2,k_2), \cdots, i_d(j_d,k_d)) . \quad (3)$$

Takođe, svakom klasteru $C(j_1,\ldots,j_d)$ pridružimo položaj klastera, tj. vektor aritmetičkih sredina indeksa neurona koji pripadaju tom klasteru:

$$in\left(C(j_1,\ldots,j_d)\right)=\frac{1}{c_1\cdot c_2\cdot\ldots\cdot c_d}\sum_{k_1=1}^{c_1}\sum_{k_2=1}^{c_2}\ldots\sum_{k_d=1}^{c_d}\left(i_1(j_1,k_1),i_2(j_2,k_2),\cdots,i_d(j_d,k_d)\right). \quad (4)$$

Algoritam:

1. Inicijalizacija skupa čvorova $C=\{a_1,a_2,...,a_N\}$, $N=N_1\cdot\ldots\cdot N_d$ i težinskih vektora $w_c\in R^n$ izabranih slučajno. Inicijalizacija skupa veza E između čvorova (neurona), E – pravougli paralelopiped dimenzije $N=N_1\cdot\ldots\cdot N_d$. Za svaki klaster $C(j_1,\ldots,j_d)$ neka je $\Delta w\left(C(j_1,\ldots,j_d)\right)=0$, gdje je $\Delta w\left(C(j_1,\ldots,j_d)\right)$ promjena težine klastera u slučajevima kada on nije pobjednik.
2. Iz skupa primjera za obučavanje slučajno se bira jedan vektor x.
3. Određuje se klaster pobjednika $C(j_1,\ldots,j_d)$, tj. indeksi $j_1,\ldots,j_d$ takvi da: $j_1,\ldots,j_d=\arg\min\|x-w(C(j_1,\ldots,j_d))\|$.
4. Promjena težinskih vektora svih neurona koji pripadaju klasteru pobjedniku i svih neurona koji pripadaju susjednim klasterima, u skladu sa pravilom $w_i=w_i-\Delta w_i+\Delta w\left(C\left(j_1,\cdots,j_d\right)\right)$, gdje je $i=\left(i_1,\cdots,i_d\right)$. Ova formula predstavlja standardno pravilo obučavanja neurona u kartama Kohonena, sa dodatim izrazom $\Delta w\left(C\left(j_1,\cdots,j_d\right)\right)$. Funkciju susjedstva i brzinu obučavanja određujemo kao i kod standardnog algoritma Kohonena, tj.:

   $$\Delta w_i=\varepsilon(t)h(r,s)(x-w_i),$$

   $$\varepsilon(t)=\varepsilon_0\left(\frac{\varepsilon_f}{\varepsilon_0}\right)^{\frac{t}{m}},$$

   $$\lambda(t)=\lambda_0\left(\frac{\lambda_f}{\lambda_0}\right)^{\frac{t}{m}},$$

   $$h(r,s)=\exp\left(-\frac{\|r-s\|}{\lambda(t)}\right)\ \text{ili}\ h(r,s)=\exp\left(-\frac{\sum_{k=1}^{d}|p_k-q_k|}{\lambda(t)}\right),$$

   gdje je $r=a\left(p_1,\cdots,p_d\right),s=a\left(q_1,\cdots,q_d\right)$.
5. Težinski vektori neurona u ostalim klasterima $C\left(k_1,\cdots,k_d\right)$ se ne mijenjaju. Mijenja se samo težina samog klastera u skladu sa pravilom:

   $$y=\varepsilon(t)h\left(i,in\left(C\left(k_1,\cdots,k_d\right)\right)\right)\left(x-w\left(C(k_1,\cdots,k_d)\right)\right)$$

$$w\left(C\left(k_1,\cdots,k_d\right)\right) = w\left(C\left(k_1,\cdots,k_d\right)\right) - y.$$

6. Ako su obrađeni svi vektori iz skupa primjera za obučavanje, algoritam završava rad. U suprotnom slučaju, prelazimo na korak 2.

Vremenska složenost algoritama izračunava se primjenom rasuđivanja sličnog izračunavanju složenosti algoritma Barnes-Hut-a [Aar03]. Uzajamno dejstvo *n* tijela jedno na drugo iz algoritma Barnes-Hut-a, zamjenjujemo dejstvom *m* ulaznih vektora na *N* neurona mreže. Ne umanjujući opštost, pretpostavimo da je mreža organizovana u obliku *k*-dimenzione kocke, tj. da sadrži d^k neurona. Ako je $d = a \cdot b$, $a > 1$, tada se mreža sastoji od a^k klastera veličine b^k. Maksimalan broj neurona za koje se mijenjaju težinski koeficijenti nije veći od $(2k+1)\cdot b^k$, u odnosu na $N = d^k = a^k \cdot b^k$ u originalnom algoritmu Kohonena. Smatrajući da je veličina klastera konstantna, dobijamo da je $k = \log_d N$. U svakom koraku, broj neurona za koje se mijenjaju težinski koeficijenti se smanjuje, pa je ukupan broj operacija za jedan ulazni vektor *x* reda $O(\log N)$. Vremenska složenost algoritma je $O(m \cdot \log N)$. Primjena algoritma ima smisla u slučajevima kada mreža sadrži veliki broj neurona. U tom slučaju, sve klastere možemo posmatrati kao određenu vrstu indeksa za polaznu mrežu. Ako je i broj klastera veoma veliki, moguće je i njih grupisati u nove klastere i napraviti hijerarhiju.

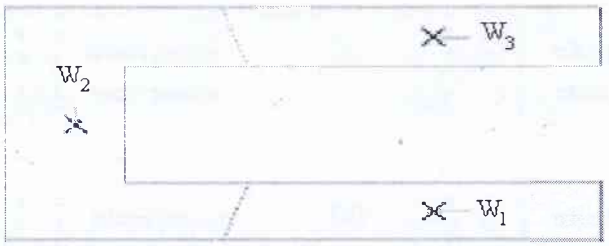
3.2 Očuvanje metričkih i topografskih karakteristika

Algoritam Kohonena definiše preslikavanje Ω skupa ulaznih podataka (nadražaja, stimula) u skup izlaza (čvorova mreže, neurona, šablona, kodnih riječi) [BauHerVil97]. Jednostavnosti radi, označavaćemo skup ulaza sa V i skup izlaza sa A. Tada preslikavanje Ω^{-1} definiše ćelije Voronija u ulaznom prostoru. Uvešćemo i pojam restriktivnog inverznog preslikavanja $\hat{\Omega}^{-1}$, koje svakom neuronu r pridružuje samo jedan "tipičan" vektor w_r ("codebook", prototip, referentni vektor, pointer, optimalni nadražaj ili centar receptivnog polja). Ovo preslikavanje može se tretirati kao kvantizacija vektora ($\hat{\Omega}^{-1}(\Omega(v))$ ili kao klasifikacija ($\Omega(v)$). Funkcija susjedstva u algoritmu Kohonena na indirektan način utiče na topografske karakteristike, dok samo pravilo obučavanja pokušava da minimizuje distorziju. Ova dva svojstva algoritma su neraskidivo povezana, što ima implikacije na optimizaciju topografskih karakteristika preslikavanja. Kako u opštem slučaju nije moguće dokazati da algoritam Kohonena konvergira ka topologiji koja čuva stacionarna stanja ([Koh01], [Hay04]), jedina mogućnost je nadgledanje stepena očuvanja topografskih

karakteristika u toku procesa obučavanja, što omogućava primjenu različitih topografskih mjera.

Centralno pitanje u primjeni samoorganizujućih mreža jeste izbor odgovarajuće topologije izlaznog prostora. Ako se efektivna dimenzija ulaznih podataka ne poklapa sa dimenzijom mreže, tada dolazi do konflikta i ne može se postići savršeno topografsko preslikavanje. Tipičan primjer je preslikavanje kvadrata na jednodimenzioni niz neurona [BauHerVil97].

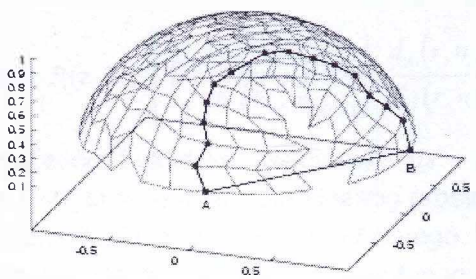
Metričke i topografske karakteristike izlaznog prostora obično su relativno jednostavne i mogu se na prirodan način izmjeriti. Prirodna struktura ulaznih podatka u realnim situacijama nije tako jasna, pa se pribjegava pojednostavljenju. Jedan način pojednostavljenja je zamjena skupa ulaznih vektora skupom referentnih vektora, što ponekad dovodi do narušavanja susjedstva u ulaznom prostoru (slika 3.2).



Slika 3.2 – Regioni 1 i 2 su susjedni, ali je w_3 najbliži vektor vektoru w_1

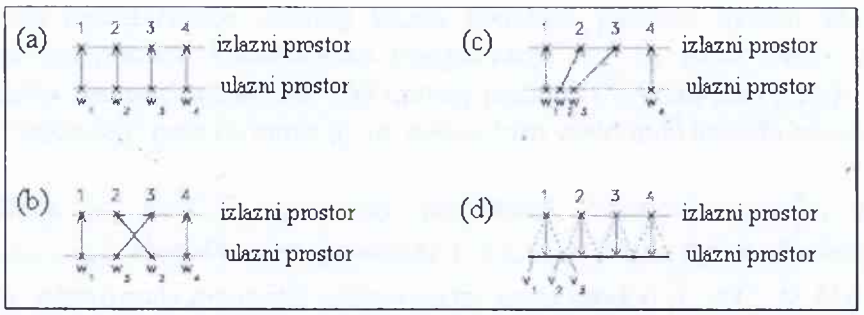
Način da se izbjegne narušavanje susjedstva u ulaznom prostoru i da se zaobiđe korišćenje rastojanja referentnih vektora predložen je u [Kos97], gdje se formira graf čiji su čvorovi ćelije Voronija, a grana u grafu postoji ako dvije ćelije imaju zajedničku granicu. Ova procedura zahtijeva da postoji dovoljno veliki broj ulaznih podataka, da bi se mogle predstaviti složene relacije između referentnih vektora (kriterijum gustine, [Kos97]).

Problem koji nije vezan za broj neurona jeste izbor odgovarajuće metrike ulaznog prostora. Tipičan primjer je predstavljanje dvodimenzionalne površi u trodimenzionalnom ulaznom prostoru na dvodimenzionalnu mrežu neurona (slika 3.3), gdje se rastojanje može mjeriti ili po površi ili u samom ulaznom prostoru.



Slika 3.3 – 2D površ u 3D prostoru projektuje se na 2D mrežu

Različite definicije topografije neuronskih mreža kombinuju metrička i topografska svojstva. Čak i u slučajevima kada nisu narušena topografska svojstva, moguće je da dođe do narušavanja metričkih osobina, što je ilustrovano u primjerima sa slike 3.4. U primjeru 3.4(a), imamo savršeno topografsko preslikavanje. U primjeru 3.4(b) postoji narušavanje topografije, jer je par najbližih vektora (w_1,w_3) preslikan u par neurona (1,3) koji nisu susjedni, ali i narušavanje metričkih svojstava jer $d_V(w_1,w_2) > d_V(w_1,w_3)$ i $d_A(1,2) < d_A(1,3)$, d_V i d_A su rastojanja u ulaznom i izlaznom prostoru respektivno. Topografska svojstva nisu narušena u primjeru (c), ali su narušena metrička svojstva, jer $d_V(w_3,w_4) > d_V(w_3,w_1)$ i $d_A(3,4) < d_A(3,1)$. U primjeru (d) ulazni prostor nije diskretan. Nema narušavanja topografskih svojstava ali postoji narušavanje metričkih svojstava jer $d_V(v_1,w_3) > d_V(w_2,w_3)$ i $d_A(1,1) < d_A(1,2)$.



Slika 3.4 – Metrička i topografska svojstva

Postoji više načina određivanja nivoa očuvanja topografskih karakteristika ulaznog prostora ([BauHerVil97], [KasLag96], [Kos97]). Dvije mjere, topografski proizvod i topografska funkcija, koje ćemo definisati ovdje imaju zajedničku osobinu: osnovna ideja nije upoređivanje apsolutnih rastojanja u ulaznom i izlaznom prostoru, već se zahtijeva neka vrsta uređenja "susjeda".

Kod topografskog proizvoda, za svaki neuron u izlaznom prostoru (koji je određen svojim indeksom ***r***), određuju se nizovi n_1 i n_2, gdje $n_1(r,k)$ k-ti susjed neurona r u ulaznom prostoru, a $n_2(r,k)$ k-ti susjed neurona r u izlaznom prostoru. Topografski proizvod definiše se formulom $P = \frac{1}{N(N-1)} \sum_r \sum_{k=1}^{N-1} \log\left(P_1(r,k)\right)$, gdje je izraz $P_1(r,k)$ dat formulom $P_1(r,k) = \left(\prod_{i=1}^{k} \frac{d_V\left(w_r, w_{n_2(r,k)}\right)}{d_V\left(w_r, w_{n_1(r,k)}\right)} \cdot \frac{d_A\left(r, n_2(r,k)\right)}{d_A\left(r, n_1(r,k)\right)} \right)$, d_V - rastojanje u ulaznom prostoru (najčešće Euklidsko rastojanje), d_A - rastojanje u izlaznom prostoru (mreži). Ako je P<0, tada je dimenzija ulaznog prostora suviše mala; ako je P>0, tada je tada je dimenzija ulaznog prostora veća nego što je potrebno; ako je $P \approx 0$, tada imamo približno poklapanje ulazne i izlazne topografije.

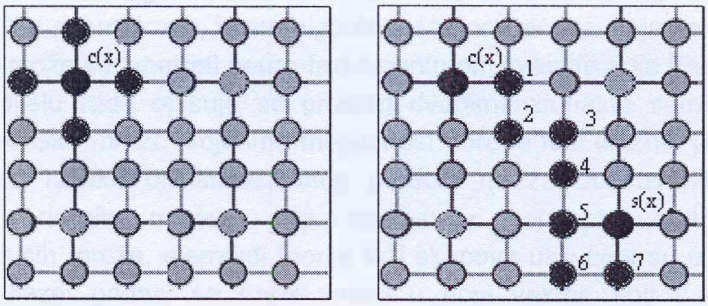
Za definisanje topografske funkcije potrebno je da konstruišemo graf D_g čiji su čvorovi ćelije Voronija, a grane povezuju dva čvora ako njihove odgovarajuće ćelije Voronija imaju zajedničku ivicu. Rastojanja između referentnih vektora zamjenjuju se rastojanjima u grafu. Za svaki neuron r i svaki prirodan broj k iz skupa $\{1,...,N-1\}$ izračunavamo vrijednosti

$$f_r(k) = \#\{r' \mid d_{\max}(r,r') > k, d^{D_g}(r,r') = 1\} \quad f_r(k) = \#\{r' \mid d(r,r') = 1, d^{D_g}(r,r') > k\},$$

gdje je $d_{\max}(r,r') = \max_{1 \le k \le d} |r_k - r'_k|$, d^{D_g} označava rastojanje u grafu, #{.} je broj elemenata skupa. Topografska funkcija definiše sa na sljedeći način: $\Phi(k) = \frac{1}{N} \sum_{r \in A} f_r(k)$, za $k \neq 0$ i $\Phi(0) = \Phi(1) + \Phi(-1)$. Topografska funkcija identički je jednaka nuli ako oba preslikavanja Ω i Ω^{-1} ne narušavaju susjedstvo.

Metod koji je predložen u ovom radu kombinuje distorziju (kao mjeru preciznosti reprezentacije ulaznog skupa podataka pomoću mreže) sa mjerom očuvanja topografskih karakteristika preslikavanja Ω. Za svaki ulazni vektor x određujemo neurona-pobjednika $c(x)$ i prvog pratioca ("second-best") $s(x)$ i tražimo dužinu "najkraćeg" puta po mreži (tj. po referentnim vektorima) između neurona $c(x)$ i $s(x)$.

Neka je $G=(C,E)$ graf koji predstavlja neuronsku mrežu, gdje je $C = \{a_1, a_2, ..., a_N\}$ skup čvorova (neurona) i $E \subseteq C \times C$ skup grana. Svakom čvoru mreže a_i pridružen je referentni vektor (vektor koeficijenata) w_i, $i=1,...N$. Ako su a i b dva neurona u mreži, tada sa $p(a,b)$ označavamo put u grafu koji spaja neurone a i b, tj. niz međusobno različitih neurona (r_1, r_2, ..., r_k) takvih da je $k>1$, $r_1=a$, r_2=b i za svako $i \in \{1,...,k-1\}$ $(r_i,r_{i+1}) \in$ E (neuroni r_i i r_{i+1} susjedni u mreži tj. postoji grana koja ih spaja). Skup svih puteva između a i b označavamo sa $L(a,b)$.



Slika 3.5 – Susjedi neurona i najkraći put u mreži

Neka je p_i = (r_1,r_2, ..., r_{ki}) $\in$ $L(a,b)$, tada je dužina puta p_i jednaka $d^G(p_i) = \sum_{l=1}^{k} d^V(w_i, w_{i+1})$. Za svaki ulazni vektor izračunavamo neurona-pobjednika $c(x) = \arg\min_i \{d(x, w_i(t))\}$ i sljedećeg najboljeg neurona $s(x) = \arg\min_{i, i \neq c(x)} \{d(x, w_i(t))\}$, a

zatim vektoru x pridružujemo i odgovarajuće rastojanje $d^C(x) = d(x, w_{c(x)}) + \min\{d^G(p_i) | p_i \in L(c(x), s(x))\}$. Funkcija kvaliteta mreže tada se izračunava po formuli $Q = \frac{1}{m}\sum_{i=1}^{m} d^C(x_i)$, gdje je $\{x_i | i = 1, ..., m\}$ skup ulaznih vektora.

3.3 Fazi-klasifikacija primjenom samorganizujućih mreža

Samoorganizujuće mreže obučavane algoritmom Kohonena najčešće se upotrebljavaju za klasterizaciju podataka, uz druge metode kao što su rastući neuro-gas (Growing Neural Gas, [Fri93], [CanCha07]), rastući višedimenzioni intervali (Hypercubical SOM [Fri97-1], [DuZhaBao06]), LVQ (Learning Vector Quantization, [Fri93-1], [SchHamBie09]) i ART (Adaptive Resonance Theory) [CarGro03]. Rješavanje zadatka klasifikacije obično se obavlja primjenom feed-forward neuronskih mreža u kombinaciji sa učenjem pomoću učitelja, tzv. "supervised learning", iako se karte Kohonena mogu upotrijebiti i za ovaj tip zadatka ([Koh01], [RitKoh89], [MitPal94], [PalBez92], [KaiKan07]).

Često je priroda ljudskog rezonovanja i zaključivanja nejasna (fuzzy), tj. postoji određeni stepen neodređenosti ili neopredijeljenosti. Sami podaci su nerijetko ili nedovoljno jasni ili su granice među njima neprecizne. U slučajevima kada treba izvršiti klasifikaciju podataka koji nisu dovoljno precizni ili kada granice između klasa nisu precizno definisane, moguće je iskoristiti teoriju fazi skupova u različitim fazama: za transformaciju ulaznih podatka, u procesu obučavanja ili za transformaciju izlaznih podataka. Takva sinteza dovela je do velikog broja takozvanih neuro-fazi algoritama klasterizacije (FKCN – Fuzzy Kohonen Clustering Networks, FLVQ – Fuzzy Learning Vector Quantization, Fuzzy ART, FOSART – Fuzzy Self-Organizing ART, itd. [Fri97-1], [Fri97-2]). Šire posmatrano, i sam algoritam samoorganozovanja karti Kohonena i rastućeg gasa može se smatrati neuro-fazi algoritmom kvantifikacije [BarBlo99].

U ovom dijelu rada opisuje se predlog dvodimenzionalne samoorganizujuće vještačke neuronske mreža, koja ima mogućnost obrade fazi ulaznih podataka i fazi klasifikacije. Za razliku od standardnog modela mreža obučavanih algoritmom Kohonena ili drugih modela koji kombinuju koncepte fazi skupova i samoorganizujućih mreža, elementi teorije fazi skupova uključeni su u svim nivoima generisanja. Ulazni podaci se transformišu u novi vektor, koji u sebi sadrži i informaciju kontekstualnog tipa. Izlazne podatke je moguće tumačiti na bar dva načina: u čistom obliku i u fazi obliku. Predloženi algoritam pripada klasi takozvanih „batch" algoritama.

Mreža prolazi kroz dvije faze: fazu samoorganizovanja i fazu testiranja. Ulazni skup vektora se transformiše tako da se novi ulazni vektor sastoji od lingvističkog i kontekstnog dijela. Svaka ulazna karakteristika (koordinata vektora) opisuje se sa nekoliko lingvističkih kategorija (promjenljivih). Takođe se uzima u obzir i informacija

o pripadnosti ulaznog vektora nekoj od konačnog broja klasa. U prvoj fazi, skup transformisanih ulaznih podataka se koristi za samoorganizovanje mreže, promjenom njenih težinskih koeficijenata. Takođe se vrši i konačna kalibracija izlaznog prostora putem pridruživanja labela neuronima, u saglasnosti sa ulaznim skupom podataka. U fazi testiranja, koristi se dio ulaznog vektora koji sadrži informacije o kontekstualnoj pripadnosti ulaznog vektora nekoj od konačnog broja klasa. U poređenju sa standardnim algoritmima, u kojim ulazni vektor može pripadati samo jednoj klasi i u kojima se informacija o pripadnosti klasi ne koristi kao ulazni podatak, predloženi algoritam efikasnije modelira slučajeve kada se podaci u ulaznom prostoru preklapaju ili su nejasno definisane granice između klasa.

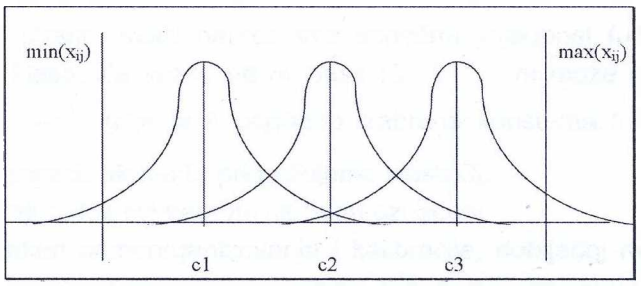
Algoritam:

1. Neka je $X = \{x_1, \cdots, x_p\}$ skup ulaznih vektora, gdje je $x_i = (x_{i1}, \cdots, x_{in}) \in R^n, i = 1, \cdots, p$ i neka je dato m klasa C_1, ..., C_m. Neka su $M_i = (M_{i1}, \cdots, M_{in}) \in R^n, i = 1, \cdots, m$ vektori matematičkog očekivanja i $D_i = (d_{i1}, \cdots, d_{in}) \in R^n, i = 1, \cdots, m$ vektori disperzije klasa C_1,..., C_m u odnosu na ulazni skup X. Inicijalizuje se skup čvorova $A = \{a_{11}, \cdots, a_{1N_2}, \cdots, a_{N_1N_2}\}$, sa težinskim vektorima $m_{ij} \in R''$ izabranim slučajno. Inicijalizacije se skup veza C između čvorova (neurona), C – pravougaonik dimenzija $N = N_1 \cdot N_2$. Funkcija susjedstva određuje se sljedećom formulom: $N_r(a_{pq}) = \{a_{ij} \mid \max(|i-p|, |j-q|) = r\}$. Lateralne veze među neuronima date su formulom: $w(a_{ij}, a_{pq}) = \begin{cases} b, a_{pq} \in N_1 \\ \frac{b}{2}, a_{pq} \in N_2 \\ 0, \ a_{pq} \in N_r, r \geq 3 \end{cases}$. Izlazna funkcija (odziv) neurona a_{ij} data je formulom $\eta_{ij}(t) = \sigma\left(m_i^T(t) \cdot x\{t\} + \sum_{a_{pq} \in N_r} w(a_{ij}, a_{pq}) \cdot \eta_{ij}(t - \delta t) \right)$

2. Funkcija pripadnosti vektora x_i klasi C_k izračunava se po formuli $\mu_k(x_i) = \frac{1}{1 + \left(\frac{d(x_i, C_K)}{fd} \right)^{fe}}$, gdje je $d(x_i, C_j) = \sqrt{\sum_{k=1}^{n} \left(\frac{x_{ik} - m_{ik}}{d_{ik}} \right)^2}$, *i=1,...,p*. Ako je $d_{ik}=0$ (u slučaju kada su sve vrijednosti koordinate jednake, disperzija je 0), uzimamo da je $d_{ik}=1$. U cilju povećavanja kontrasta između klasa, može se koristiti i funkcija $v(x_i) = \begin{cases} 2(\mu_k(x_i))^2, 0 \leq \mu_k(x_i) \leq 0.5 \\ 1 - 2(1 - \mu_k(x_i))^2, 0.5 \leq \mu_k(x_i) \leq 1 \end{cases}$, kako je predloženo u [PalBez92]

3. Neka *i*-toj (*i=1,...,n*) koordinati vektora *x* odgovara k_i lingvističkih kategorija *A(i,j)*, *j=1,...,*k_i. Svakoj od navedenih lingvističkih promjenljivih možemo pridružiti funkciju

pripadnosti f_{ij}, $f_{ij}(x,c,a) = \begin{cases} 2\left(1-\frac{\|x-c\|}{a}\right)^2, \frac{a}{2} \le \|x-c\| \le a \\ 1-2\left(\frac{\|x-c\|}{a}\right)^2, 0 \le \|x-c\| \le \frac{a}{2} \\ 0, \quad \|x-c\| > a \end{cases}$, j=1,...,$k_i$. Grafik funkcije $f_{ij}(x,c,a)$ u R^2 prikazan je na slici 3.6.



Slika 3.6 - Funkcija f_{ij} sa tri lingvističke kategorije

4. Slučajno se bira proizvoljni vektor $x_i \in X$ i transformiše se u novi vektor $x_i^{new} \in R^s$, gdje je:

   $s = \sum_{i=1}^{n} k_i + m, x_i^{new} = \alpha\left(x_i^1, 0\right) + \beta\left(0, x_i^2\right)$

   $x_i^1 = \left(f_{11}(x_{i1}, c_1, a), \cdots, f_{1k_1}(x_{i1}, c_1, a), f_{21}(x_{i2}, c_2, a), \cdots, f_{2k_2}(x_{i2}, c_2, a), \cdots, f_{nk_n}(x_{in}, c_n, a)\right)$,

   $x_i^2 = \left(\mu_1(x_i), \cdots, \mu_{m1}(x_i)\right)$,

   α i β parametri koji kontrolišu učešće fazi i kontekstualne informacije u ulaznom vektoru. Uzimamo da je $\alpha=1$ i $0<\beta<0.5$ proizvoljno (značaj kontekstualne informacije je umanjen u odnosu na lingvistički dio informacije u vektoru, ali i ona učestvuje u procesu samoorganizovanja).
5. Određuje se neuron-pobjednik $a_{i_1 j_1}$, tj. indeksi i_1 i j_1 takvi da je: $i_1, j_1 = \arg\min \left\| x_i - m_{i_1 j_1}(t) \right\|$.
6. Mijenjaju se težinski vektori svih neurona u skladu sa pravilom $m_{ij}(t+1) = \begin{cases} m_{ij}(t) + h(a_{ij}, a_{i_1 j_1})\left(x(t) - m_{ij}(t)\right), a_{ij} \in N_r(a_{i_1 j_1}) \\ m_{ij}(t), \end{cases}$.
7. Ako nisu obrađeni svi vektori ulaznog skupa X, preći na korak 4, inače preći na korak 8.
8. Kalibracija. Iz transformisanog vektora uzima se u obzir samo kontekstualna informacija, tj. vrijednosti parametara su $\alpha=0$ i $\beta=1$. Izlazni prostor moguće je

podijeliti na particije na dva načina. Prvi način pridružuje svakom neuronu jednu klasu C_k od mogućih *m* klasa, za koju dati neuron daje maksimalni odziv ("hard partitioning"). Tada se u formuli za x_i^2 uzima da je $\mu_j(x_i) = \begin{cases} 1, q = k \\ 0, q \neq k \end{cases}$, *k=1,...,m.*

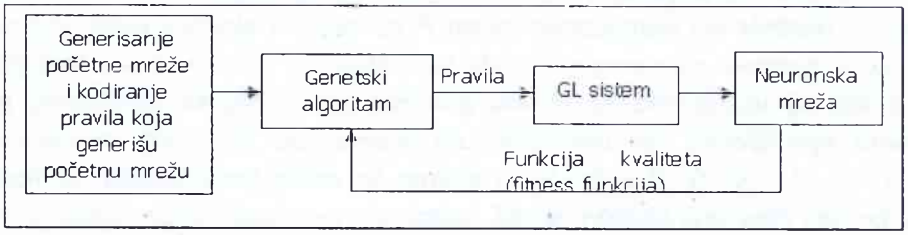
Drugi način svakom neuronu pridružuje konačnu vrijednost funkcije pripadnosti klasi C_k. Neka je a_{pq} neuron koji generiše maksimalan odziv η^k za klasu C_k. Definišemo pripadnost izlazne vijednosti neurona a_{ij} u odnosu na klasu C_k kao $\mu_k(\eta_{ij}) = \frac{\eta_{ij}}{\eta^k}$, $i = 1,...,N_1; j = 1,...,N_2; k = 1,..,m$. $0 \leq \mu_k(\eta_{ij}) \leq 1$ i $\mu_k(\eta_{ij}) = 1$ ako je *i=p*, *j=q*. S druge strane, svaki neuron ima konačnu vrijednost funkcije pripadnosti svakoj od *m* klasa. Za svaku od *m* klasa C_k, *k=1,...,m* može se definisati skup $L_k = \{a_{ij} | \mu_k(\eta_{ij}) > \delta\}$, gdje je δ pogodno izabrana konstanta, $0 < \delta \leq 1$, i svakom neuronu koji pripada skupu L_k pridružujemo klasu C_k.

9. Ponavljati korak 8 dok svi neuroni ne budu označeni.
10. Testiranje: Nakon samoorganizovanja i kalibracije, dobijenoj mreži se kao ulaz predstavljaju vektori iz skupa za testiranje, koji je neki dio skupa X. Ulazni vektori sastoje se samo od lingvističke informacije, tj. parametri su α=1 i β=0.

4. Generisanje topologije neuronske mreže

U ovom poglavlju definisan je jedan metod za generisanje topologije modularne neuronske mreže primjenom sistema Lindenmajera (L-sistema) za generisanje fraktala i prikazana je upotreba genetskog algoritma kao svojevrsnog optimizatora dobijenih topologija.

Proces generisanja modularne neuronske mreže prikazan je na slici 4.1.



Slika 4.1 – Proces generisanja modularne mreže

Proces ilustrovan na slici 4.1 detaljnije se može opisati na sljedeći način:

1. Određujemo početnu mrežu i skup pravila koji generišu izabranu početnu mrežu. Početnu arhitekturu mreže je uvijek moguće izabrati na jednostavan način – biramo dvoslojnu mrežu koja sadrži d neurona u ulaznom sloju i jedan neuron u izlaznom sloju, gdje je d dimenzija ulaznog prostora. Ako već postoji mreža M koja rješava postavljeni zadatak, a žele se poboljšati njene karakteristike, kao početnu arhitekturu možemo izabrati baš arhitekturu mreže M. U svakom slučaju, određujemo skup pravila koja generišu izabranu početnu arhitekturu (u krajnjem slučaju, uvijek je moguće odrediti bar jedno pravilo), a zatim kodiramo data pravila u obliku niza simbola. Dobijeni nizovi simbola će predstavljati skup hromozoma. Ovaj korak nije obavezan i u simulacijama je upotrebljen samo u problemu preslikavanja [Hoo91].
2. Genetski algoritam generiše nizove binarnih brojeva, koji su hromozomi individua. Kreiranje novih individua ostvaruje se primjenom operatora kombinacije, čija je vjerovatnoća primjenjivanja direktno proporcionalna funciji kvaliteta individua izabranih za kombinaciju. Vrijednost funkcije kvaliteta individue određuje se u koraku 4.
3. Hromozome dobijene u koraku 2, dekodiramo u gramatička pravila L-sistema. Primjenjujući dobijena pravila izvođenja, generišemo string koji je kodirani zapis arhitekture mreže.
4. Iz dobijenog stringa generišemo mrežu, koju obučavamo za rješavanje određenog zadatka, primjenom algoritma obratnog rasprostiranja greške ili

nekog drugog algoritma obučavanja. Funkcija kvaliteta hromozoma može se odrediti u zavisnosti od srednje kvadratne greške mreže ili nekog drugog kriterijuma.

4.1 GL-sistemi

L-sistem je trojka $G=(\Sigma_1, M, H)$, gdje je Σ_1 alfabet, M skup gramatičkih pravila, H aksioma ili početni simbol gramatike. Gramatičko pravilo u kontekstno-zavisnim L-sistemima ima oblik $L<P>R \rightarrow S$, gdje je L lijevi, a R desni kontekst simbola P. U običnim L-sistemima, značenje pravila $L<P>R \rightarrow S$ je sljedeće: ako se L nalazi neposredno isred simbola P i ako se R nalazi neposredno iza simbola P, tada P zamjenjujemo simbolom S. Za razliku od običnih gramatičkih sistema u kojima se pravila primjenjuju serijski, u L-sistemima pravila se primjenjuju paralelno, tj. u svakom koraku izvođenja istovremeno se zamjenjuju sva pojavljivanja simbola P simbolom S, primjenjujući jedno od pravila $L<P>R \rightarrow S$, $P>R \rightarrow S$, $L<P \rightarrow S$ i $P \rightarrow S$ (ukoliko ona sva postoje u gramatici). Ako je moguće primjeniti više od jednog pravila, bira se pravilo sa najdužim kontekstom. Ako postoji više pravila sa jednakom dužinom, bira se prvo pravilo u skupu pravila. String se interpretira slijeva udesno.

Primjer 1. Kontekstno-zavisan L-sistem

Alfabet: $A=\{X,Y,Z\}$

Aksioma: X

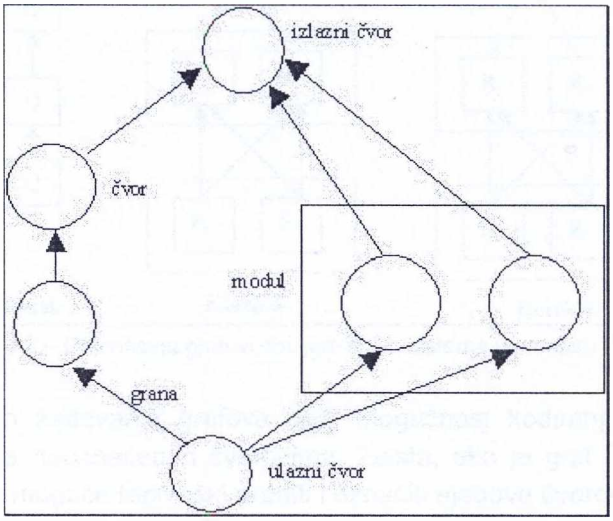
Pravila: $\begin{cases} 1.\ X \rightarrow XYZ \\ 2.\ X<Y \rightarrow X \\ 3.\ Y<Z \rightarrow YZY \\ 4.\ Y<Z>X \rightarrow Y \end{cases}$

Neka izraz $\alpha \Rightarrow \beta$ označava, da podstring α stringa iz koraka $i-1$ zamjenjujemo stringom β u koraku i. Stringove zamjenjujemo slijeva udesno. Proces izvođenja po koracima će imati sljedeći izgled:

Korak 1. X	(aksioma)
Korak 2. XYZ	($X \Rightarrow XYZ$ - pravilo 1)
Korak 3. XYZXYZY	($X \Rightarrow XYZ$ - pr. 1, $Y \Rightarrow X$ pr. 2 , $Z \Rightarrow YZY$ - pr. 3)
Korak 4. XYZXYXYZXYZYY	($X \Rightarrow XYZ$ - pr.1, $Y \Rightarrow X$ pr. 2, $Z \Rightarrow Y$ - pr. 4, $X \Rightarrow XYZ$ - pr.1, $Y \Rightarrow X$ pr. 2, $Z \Rightarrow YZY$ - pr. 3, $Y \Rightarrow Y$ - prepiše se posljednji simbol Y iz koraka 3)

GL-sistem je specijalni slučaj takozvanih 2L-sistema, tj. L-sistema u kojima postoji najmanje jedno gramatičko pravilo sa dvostranim kontekstom. Rezultat primjene GL-sistema je orijentisani graf. Alfabet Σ_1 GL-sistem čine tri grupe simbola: Σ – neki podskup skupa svih slova engleske abecede, C – skup cifara 0-9 (ili neki njegov podskup) i specijalni simboli – parovi zagrada '[' i ']', '(' i ')' i simbol ',' (zapeta). Na taj način, $\Sigma_1 = \Sigma \cup C \cup \{'[',']','(',')',','\}$.

Topologiju neuronske mreže opredeljuje graf dobijen primjenom gramatičkih pravila GL-sistema. Čvorovi grafa označeni su slovima engleske abecede. Slova koja se nalaze između zagrada '[' i ']' obrazuju modul. Svaki broj x iz liste brojeva koji se nalaze unutar simbola '(' i ')', a neposredno iza slova (čvora) ili podstringa (modula) zatvorenog unutar zagrada '[' i ']' (takav čvor ili modul nazvaćemo početnim) interpretiramo kao "napraviti skok od x čvorova ili modula nadesno, ako je to moguće, i povezati odgovarajući čvor ili modul granom sa početnim čvorom ili modulom". Ako dopustimo da alfabet sadrži i simbol '-' (minus), i ako izraz '-x' interpretiramo kao "napraviti skok od x čvorova ili modula ulijevo, ako je to moguće, i povezati odgovarajući čvor ili modul granom sa početnim čvorom ili modulom", tada imamo mogućnost kreiranja rekurzivnih neuronskih mreža. Ako se u listi brojeva unutar simbola '(' i ')' nalazi samo jedan broj, tada ne pišemo simbole '(' i ')'.



Slika 4.2 – Graf koji opisuje topologiju mreže

Za razliku od 2L-sistema, gdje se u svakom koraku može zamijeniti samo jedan simbol, u GL-sistemima se može zamijeniti cijeli podstring. Kontekst u GL-sistemima sa određuje tek nakon interpretacije cijelog stringa (kao npr. u primjeru 2). Izlazni čvorovi prvog od dva susjednih modula povezani su sa ulaznim čvorovima drugog modula. Ulazni čvorovi modula ne primaju ulaz ostalih čvorova unutar tog modula, a izlazni čvorovi ne predaju svoj izlaz drugim čvorovima unutar modula.

Primjer 2. Kontekstno-zavisan GL-sistem

Alfabet: $A = \{P,Q,R,S\} \cup \{1,2\} \cup \{'[','],',','\}$

Aksioma: P

Pravila:
$$\begin{cases} 1.\ P \to QQQ \\ 2.\ Q > Q \to [R,S] \\ 3.\ Q \to R \\ 4.\ R < S \to R \\ 5.\ S > S \to R1 \end{cases}$$

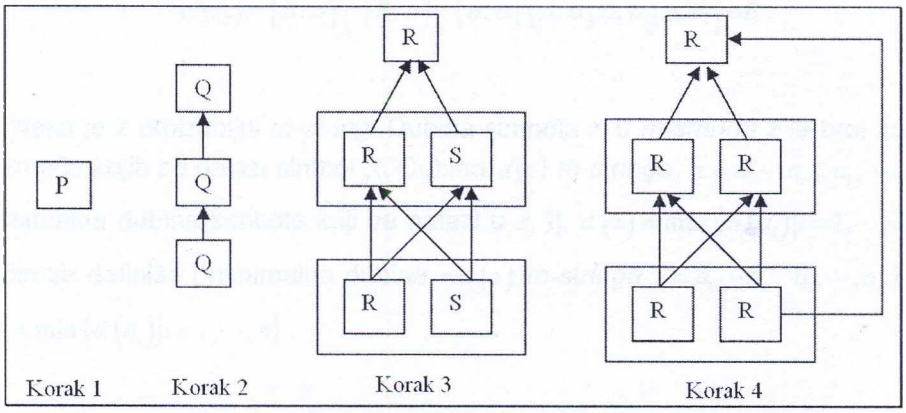
Korak 1. P (aksioma)

Korak 2. QQQ (pravilo 1.)

Korak 3. $[R,S][R,S]R$ (pravilo 2., pravilo 2., pravilo 3.)

Korak 4. $[R,R1][R,R]R$ (pravilo 5., pravilo 4.)

U koraku 4, prvo slovo S zamijenjeno je stringom R1, jer je ono povezano sa drugim slovom S u stringu $[R,S][R,S]R$. Takođe, drugo slovo S zamijenjeno je slovom R, jer postoji grana od prvog slova R do drugog slova S u stringu $[R,S][R,S]R$. Grafički prikaz koraka prikazan je na slici 4.3.



Slika 4.3 – Orijentisani grafovi dobijeni iz GL-sistema u primjeru 2

Ovakav način zadavanja grafova daje mogućnost kodiranja svih mogućih acikličnih grafova sa neoznačenim čvorovima. Zaista, ako je graf G sa n čvorova acikličan, tada ga je moguće topološki urediti i označiti njegove čvorove na primjer sa $A_1, \ldots, A_n$. Topološko uređenje nam garantuje da ako postoji grana (A_i, A_j), tada je $i<j$. Za svaku granu (A_i, A_j), $i<j$, u string $A_1 \cdots A_n$ umećemo između simbola A_i i A_j oznaku da postoji grana koja spaja čvor A_i sa A_j, na sljedeći način: ako između simbola A_i i A_j već postoji lista oblika $(k_1,\ldots,k_s)$, $s>0$, tada na kraj liste dopisujemo broj $j-i$; ako lista ne postoji , tada je kreiramo tj. umećemo $(j-i)$. Tako dobijeni string reprezentuje dati graf G i može se uzeti kao aksioma.

4.1.1 Formalna definicija GL-sistema

U cilju formalnog definisanja pojmova lijevog i desnog konteksta u GL-sistemu, uvodi se pojam *m-stringa*. *m-stringom* nazivamo string (riječ) $x \in \Sigma_1^*$ takvu da:

- broj otvorenih i zatvorenih zagrada '[' i ']' je jednak
- u svakom prefiksu riječi x, broj otvorenih zagrada '[' nije manji od broja zatvorenih zagrada ']'
- svaki simbol zapeta mora se nalaziti između bar jednog para zagrada '[' i ']'.

Za *m-string x*, *cl*(x) označava string koji se dobija iz stringa *x* uklanjanjem svih brojeva, zagrada '(' i ')' i zapeta koje su se nalazile unutar tih zagrada.

Primjer 3. $s_1 = [C,C1][C,C]C$, $cl(s_1) = [C,C][C,C]C$,

$s_2 = [B,B]\big[\,[B,B]\big[\,[B,B](1,2)[A,B]B1,B\big]B1,B\big]BB$,

$cl(s_2) = [B,B]\big[\,[B,B]\big[\,[B,B][A,B]B,B\big]B,B\big]BB$.

Neka je *z* proizvoljni *m-string*. Dubina simbola *X* u *m-stringu z* je broj zagrada '[' i ']' između kojih se nalazi simbol *X*. Dubina $d(z)$ *m-stringa* $z = a_1 \cdots a_n$, $a_1, \cdots, a_n \in \Sigma$ je maksimalna dubina simbola koji se nalazi u *z*, tj. $d(z) = \max\{d(a_i) | i = 1, \cdots, n\}$. Na isti način se definiše i minimalna dubina $md(z)$ *m-stringa* $z = a_1 \cdots a_n$, $a_1, \cdots, a_n \in \Sigma$, tj. $md(z) = \min\{d(a_i) | i = 1, \cdots, n\}$.

Primjer 4. Dubine svih simbola u stringovima s_1 i s_2 iz primjera 3 prikazane su ispod samog simbola. Na osnovu toga su određene dubina i minimalna dubina samih stringova.

$s_1 = [C,C1][C,C]C$ (dubine: 0 1 1 1 1 0 0 1 1 1 0 0)

$s_2 = [B,B][[B,B][[B,B]12[A,B]B1,B]B1,B]BB$ (dubine: 0 1 1 1 0 0 1 2 2 2 1 1 2 3 3 3 2 2 2 3 3 3 2 2 2 2 1 1 1 1 1 0 0 0)

$md(s_1) = md(s_2) = 0$ $d(s_1) = 1$, $d(s_2) = 3$.

Neka je string $s_3 = [[B,B]12[A,B]B1,B]$ (dubine: 0 1 2 2 2 1 1 1 2 2 2 1 1 1 1 0) (podstring *m-stringa* s_2), tada je $d(s_3) = 2$ i $md(s_3) = 1$.

Neka je $y = a_j \cdots a_k$ podstring *m-stringa z*. Nosač $cr^d(y)$ nivoa *d*, $0 \le d \le md(y)$, podstringa *y* je najduži m-podstring $t = a_i \cdots a_j \cdots a_k \cdots a_m$ *m-stringa z* koji zadovoljava uslove:

1. $\forall l \in \{i, \cdots, m\}\ d(a_l) \ge d$ u m-stringu *z*;
2. ako za neki indeks $l \in \{i, \cdots, m\}\ a_l = ','$, tada $d(a_l) > d$ u m-stringu *z*.

Minimalni nosač $mcr^d(y)$ nivoa *d*, $0 \le d \le md(y)$, podstringa *y* je najkraći m-podstring $t = a_i \cdots a_j \cdots a_k \cdots a_m$ *m-stringa z* koji zadovoljava uslove:

1. $\forall l \in \{i, \cdots, m\}\ d(a_l) \ge d$ u m-stringu *z*;
2. ako za neki indeks $l \in \{i, \cdots, m\}\ a_l = ','$, tada $d(a_l) > d$ u m-stringu *z*;
3. $\exists l \in \{i, \cdots, m\}\ d(a_l) = d$ u m-stringu *z*.

Primjer 5. Odredićemo nosače i minimalne nosače za simbol *B* označen strelicom u *m-stringu* s_2 (šesto slovo B sa lijeve strane) i za string s_4, gdje je $s_4 = [_2 A, B _2]$ podstring stringa s_3. Kako je $md(B) = d(B) = 3$ i $md(s_4) = 2$, odredićemo $cr^d(B), d = 0,1,2,3$ i $cr^d(s_3), d = 0,1,2$.

$$s_2 = [_0 B, B _1][_1[_1 B, B _2][_2[_2 B, \overset{\downarrow}{B} _3]_3 1_2 2_2 \overbrace{[_2 A, B _2]}^{s_4} B 1_2, B _2]_2 B 1_1, B _1]_1 B B _0$$

$$cr^3(B) = mcr^3(B) = B,$$

$$cr^2(B) = [_2 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, \quad mcr^2(B) = [_2 B, \overset{\downarrow}{B} _2]$$

$$cr^1(B) = [_1 B, B _1][_1[_1 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, B _1]_1 B 1_1,$$

$$mcr^1(B) = [_1[_1 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, B _1]$$

$$cr^0(B) = s_2, \quad mcr^0(B) = [_0[_0 B, B _1][_1[_1 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, B _1]_1 B 1_1, B _0]$$

$$cr^2(s_4) = [_2 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, \quad mcr^2(s_4) = s_4,$$

$$cr^1(s_4) = [_1 B, B _1][_1[_1 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, B _1]_1 B 1_1,$$

$$mcr^1(s_4) = [_1[_1 B, B _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, B _1],$$

$$cr^0(s_4) = s_2, \quad mcr^0(s_4) = [_0[_0 B, B _1][_1[_1 B, \overset{\downarrow}{B} _2]_2 1_2 2_2 [_2 A, B _2]_2 B 1_2, B _1]_1 B 1_1, B _0].$$

Pretpostavimo da za neki m-podstring *t* *m-stringa* *z* važi $cl\left(cr^{d}(t)\right)=y_1\cdots y_n$, gdje $\forall i\in\{1,\cdots,n\}$, y_i simbol alfabeta ili je oblika $y_i=[\underset{d}{t_1},\cdots,\underset{d}{t_k}]$. Tada broj *n* nazivamo dužinom nosača $cr^{d}(t)$ i označavamo sa $len^{d}(t)$. Ako podstring *t* pripada y_i, tada broj *i* nazivamo pozicijom podstringa *t* u nosaču $cr^{d}(t)$ i označavamo je sa $pos^{d}(X)=i$.

Simbol *X* dubine *d* je ulazni simbol *m-stringa* *z* dubine *d*(*z*) ako su ispunjeni sljedeći uslovi:

1. $d(X)\le d(z)$
2. $\forall m\in\{d(z),\cdots,d(X)\}$, ako je $cl\left(cr^{m}(X)\right)=y_1^{m}\cdots y_n^{m}$, tada X pripada stringu y_1^{m}.

Skup svih ulaznih simbola *m-stringa* *z* označićemo sa *In*(*z*).

Simbol *X* dubine *d* je izlazni simbol *m-stringa* *z* dubine *d*(*z*) ako su ispunjeni sljedeći uslovi:

1. $d(X)\le d(z)$
2. $\forall m\in\{d(z),\cdots,d(X)\}$, ako je $cl\left(cr^{m}(X)\right)=y_1^{m}\cdots y_n^{m}$, tada X pripada stringu y_n^{m}.

Skup svih ulaznih simbola *m-stringa* *z* označićemo sa *Out*(*z*).

Primjer 6. Skupovi ulaznih i izlaznih simbola

$In\left(cr^{3}(B)\right)=In(B)=\{B\},\ Out\left(cr^{3}(B)\right)=Out(B)=\{B\}$

$In\left([\overset{\downarrow}{B},\overset{\downarrow}{B}]1\,2\,[A,B]B1\right)=\left\{\overset{\downarrow}{B},\overset{\downarrow}{B}\right\},\ Out\left([B,B]1\,2\,[A,B]\overset{*}{B}1\right)=\left\{\overset{*}{B}\right\}$

$In\left([\overset{\downarrow}{B},\overset{\downarrow}{B}]\right)=Out\left([\overset{\downarrow}{B},\overset{\downarrow}{B}]\right)=\left\{\overset{\downarrow}{B},\overset{\downarrow}{B}\right\}$

$In\left([[B,B][[B,B]1\,2\,[A,B]B1,B]B1,B]\right)=\{B,B\}$ - prvo i drugo slovo B sa lijeve strane

$Out\left([[B,B][[B,B]1\,2\,[A,B]B1,B]B1,B]\right)=\{B,B\}$- prvo i drugo slovo B sa desne strane.

Skup vezujućih brojeva m-podstringa *t* m-stringa *z*, u oznaci *cif*(*t*), određujemo na sljedeći način:

- ako postoji podstring $t(c_1,\cdots,c_n)$, tada svi brojevi $c_i, i=1,\cdots,n$ pripadaju skupu *cif*(*t*);
- ako za prirodan broj d važi $cl\left(mcr^d(t)\right)=y_1\cdots y_n$, $t\in Out\left(mcr^d(t)\right)$, i ako postoji podstring oblika $y_1\cdots y_n(c_1,\cdots,c_m)$, tada svi brojevi $c_i, i=1,\cdots,n$ pripadaju skupu *cif*(*t*).

Skup svih desnih konteksta *RC*(*t*) m-podstringa *t* određujemo na sljedeći način: za svaki broj $j\in cif(t)\cup\{0\}$, neka je $p(j)=\max\left\{d \middle| i+j+1\le len^d(t), i=pos^d(t)\right\}$ i $cl\left(cr^{p(j)}(t)\right)=y_1\cdots y_n$; tada y_{i+j+1} i $In\left(y_{i+j+1}\right)$ pripadaju skupu *RC*(*t*). Ako takav broj *p* ne postoji, tada broj *j* isključujemo iz skupa *cif*(*t*). Skup svih lijevih konteksta *LC*(*t*) m-podstringa *t* određujemo pomoću skupa desnih konteksta, jer je $LC(t)=\left\{s \middle| t\in RC(s)\right\}$.

Gramatičko pravilo $L<P>R\to S$ GL-sistema je pravilno ako su ispunjeni sljedeći uslovi:

- P je m-string
- S je m-string ili prazan string (string dužine nula)
- L (ako postoji) pripada skupu LC(t)
- R (ako postoji) pripada skupu RC(t)

Skupove LC(t) i RC(t) određujemo samo za stringove *t* za koje u gramatici postoji najmanje jedno pravilo oblika $L<t>R\to S$, ili $L<t\to S$ ili $t>R\to S$.

Sve m-stringove alfabeta Σ_1 možemo generisati primjenom kontekstno-slobodne gramatike $G=(\Sigma_1,NS,<Graph>,PP)$, gdje je: $\Sigma_1=\Sigma\cup C\cup\{'[','],'(',')',',',\}$ alfabet, Σ skup slova, $letter\in\Sigma$, $digit\in C$, *NS* skup neterminalnih simbola gramatike sa elementima $<Modul>$, $<ListOfModuls>$, $<Graph>$, $<GraphRest>$, $<ListOfDigits>$, $<ListOfNumbers>$ i $<Number>$, početni simbol gramatike $<Graph>$, a *PP* skup gramatičkih pravila:

$<Graph>\to<Modul><ListOfDigits><GraphRest>$

$<Modul>\to letter|[<ListOfModules>]$

$<ListOfModules>\to<Graph>|<Graph>,<ListOfModules>$

$<GraphRest>\to\varepsilon|<Modul><ListOfDigits><GraphRest>$

$<ListOfDigits>\to\varepsilon|<Number>|(<ListOfNumbers>)$

$<ListOfNumbers>\to<Number>|<ListOfNumbers>$

$<Number>\to digit\,|\,digit<Number>$

String dužine nula označen je simbolom ε.

4.2 Genetski algoritam

Kodiranje simbola alfabeta realizuje se pomoću tablice prevoda. Jednom alfabetu možemo pridružiti više tablica prevoda.

Proces interpretacije hromozoma ostvaruje se u tri koraka:

1. Svaka grupa od k bitova se prevodi odgovarajući simbol tablice prevoda. U primjeru 7, k = 6; u opštem slučaju k predstavlja broj bitova potrebnih za kodiranje svih simbola gramatike.
2. Određuje se najkraći podstring takav da su prvi i poslednji simbol '&' i između njih se nalaze još tačno tri simbola '&'.
3. Svaki podstring iz koraka 2 predstavlja jedno gramatičko pravilo. Iz dobijenog skupa pravila eliminišemo sva ona koja ne predstavljaju pravilna pravila. Tako dobijeni skup pravila predstavlja gramatička pravila GL-sistema za jednu neuronsku mrežu.

Primjer 7. Jedna moguća tablica prevoda za alfabet $\Sigma_1 = \Sigma \cup C \cup \{'[','],',','}$, $\Sigma = \{A, B, C, D, E, F, G, H\}$, $C = \{1, 2, 3, 4, 5, 6\}$. Specijalni simbol '&' ima dvostruku namjenu: određuje početak i kraj gramatičkog pravila i određuje lijevi i desni kontekst.

	00	01	10	11	
00	4	,	D	]	00
	4	[	D	]	01
	&	[	3	3	10
	&	[	3	6	11
01	&	2	E	,	00
	&	2	E	]	01
	&	2	F	]	10
	&	2	F	]	11
10	,	,	G	[	00
	,	,	G	[	01
	3	A	H	]	10
	5	A	H	]	11
11	1	B	&	C	00
	1	B	&	C	01
	1	B	[	C	10
	1	B	[	C	11

Tabela 4.1 – Tablica prevoda

Primjer 8. Interpretacija hromozoma u skladu sa kodiranjem datim tablicom prevoda iz primjera 7.

$$\underbrace{000011}_{\&}\underbrace{111000}_{\&}\underbrace{100110}_{A}\underbrace{111001}_{\&}\underbrace{110101}_{B}\underbrace{110111}_{B}\underbrace{000010}_{\&}\underbrace{001000}_{D}\underbrace{000011}_{\&}$$

Kao rezultat dobijamo string &&A&BB&D&, koji je kodirana verzija gramatičkog pravila $A > BB \rightarrow D$.

Ako počnemo interpretaciju hromozoma od drugog bita, tada se dobija string [1]1H][&, koji nije *m-string* jer broj simbola ']' u podstringu [1]1H] je veći od broja simbola '[' u istom podstringu.

4.3 Definisanje polazne arhitekture mreže

Ako postoji mreža koja daje neko rješenje traženog zadatka, tada za polaznu arhitekturu možemo izabrati baš arhitekturu te mreže. Proizvoljno izaberemo alfabet, svakom čvoru mreže pridružimo neki simbol alfabeta i izaberemo jedan string X koji generiše tu mrežu. Skup gramatičkih pravila se sastoji od samo jednog pravila $A \to X$, gdje $A \in \Sigma$ odabiramo kao aksiomu.

Za polaznu arhitekturu uvijek se može izabrati i dvoslojna mreža koja u prvom sloju ima *d* neurona, gdje je *d* dimenzija prostora ulaza, a u drugom sloju ima samo jedan neuron. U slučaju ako unaprijed znamo i broj izlaza *m*, tj. kada je dimenzija izlaznog prostora m, tada za polaznu arhitekturu možemo izabrati dvoslojnu mrežu sa *d* neurona u prvom i *m* neurona u drugom sloju.

5. Programski paket NNGen

U ovom poglavlju opisane su osnovne karakteristike programskog paketa NNGen (Neural Network Generator). NNGen je alat sa komandne linije koji je napisan u programskim jezicima C i C++. Ovaj paket se sastoji od više modula:

- ***GenAlg***, modul za genetski algoritam
- ***BackProp***, modul koji implementira algortam propagacije unazad („back-propagation") za obučavanje neuronske mreže
- ***StringToRules***, modul za transformaciju jedinki populacije u pravila gramatike
- ***LSystem***, modul koji simulira proces izvođenja u gramatici i kao izlaz daje graf koji predstavlja arhitekturu tražene mreže. Ovaj modul trenutno ima tri podmodula. Prvi podmodul je ***BasicNet***, koji kao izlaz daju graf u obliku pogodnom za direktnu upotrebu u modulu ***BackProp***. Drugi podmodul je ***CNet***, koji generiše kod u jeziku C za kreiranje neuronske mreže. Treći podmodul je ***JooneNet***, koji generiše kod za kreiranje neuronske mreže primjenom softverskog paketa JOONE [Joo07]
- ***SOM***, modul koji je namijenjen za obučavanje i klasterizaciju samoorganizujućih karti. Ovaj modul implementira algoritam Kohonena za obučavanje mreže.

Svi modulu su napisani u programskom jeziku C, osim modula ***BackProp*** i ***SOM*** koji su napisani u programskom jeziku C++. Modul ***BackProp*** predstavlja prerađenu i dopunjenu verziju koda iz [BoeKuiHap93]. Prva verzija softvera napisana je primjenom alata Microsoft Visual C++ 6.0. Trenutna verzija softvera se dorađuje primjenom alata CodeBlocks. Svi programi su testirani na operativnom sistemu Windows XP SP3. Kod je napisan tako da se može lako prebaciti na bilo koju platformu koja podržava i C i C++.

5.1 Modul GenAlg

Ovaj modul predstavlja implementaciju genetskog algoritma i sadrži funkcije koje implementiraju genetske operatore opisane u [Whi89] i [Mitc98]. Glavne strukture u programu su `Member` i `Population` koje su date sa:

```
typedef struct{
  unsigned  *genPos; /* niz pozicija gena u hromozomu */
  float fitness;
  Byte **genValue; /* niz hromozoma */
} Member;
```

```
typedef struct{
  unsigned popSize, /* velicina populacije */
           nrgenes, /*  broj gena u hromozomu */
           genSize; /* velicina gena u populaciji */
  Member *member;   /* hromozomi */
} Population;
```

Struktura `Member` opisuje jedan hromozom u populaciji, dok struktura `Population` opisuje čitavu populaciju. Čitava populacija može biti snimljena u fajl. Zaglavlje fajla opisano je sljedećom strukturom:

```
typedef struct{
  unsigned long generation;  /* velicina populacije */
  unsigned popSize, nrGenes, genSize;
} Fileheader;
```

Poslije zaglavlja, u fajl se upisuju podaci o svim hromozomima populacije: kvalitet (fitness) hromozoma, niz pozicija i sam hromozom. Inicijalizacija populacije se realizuje primjenom funkcija `DefinePopulation` i `DefineMember`, dok se memorija oslobađa pozivom funkcija `FreePopulation` i `FreeMember`:

```
Population *DefinePopulation(unsigned popSize,
                             unsigned nrGenes,
                             unsigned genSize,
                             BOOL init);
Population *DefineMember(unsigned nrGenes, unsigned genSize);
void FreePopulation(Population *p);
void FreeMember(Member *p, unsigned genSize);
```

Funkcije koje upisuju cjelokupnu populaciju u fajl i čitaju populaciju iz fajla su:

```
int SavePopulation(Population *p, FILE *fp,
                   unsigned long gen);
int SavePopName(Population *p, char *fname,
                unsigned long gen);
int LoadPopulation(Population *p, FILE *fp,
                   unsigned long gen);
int LoadPopName(Population *p, char *fname,
                unsigned long gen);
```

Moguće je upisati u fajl ili učitati iz fajla i pojedinačni hromozom, primjenom sljedećih funkcija, gdje argument `nrMember` predstavlja redni broj hromozoma koji će biti upisan u fajl ili pročitan iz fajla:

```
int SaveMember(Member *p, unsigned nrMember, FILE *fp,
               unsigned nrGenes, unsigned genSize);

int LoadMember(Member *p, unsigned nrMember, FILE *fp,
               unsigned nrGenes, unsigned genSize);
```

Operacija selekcije implementirana je u funkcijama:

```
unsigned Select(Population *p);
unsigned Rank(Population *p, float pressure);
```

Funkcija `Select` vrši izbor hromozoma po principu točka za rulet ("roulette wheel", [Mitc98]), dok funkcija `Rank` vrši selekciju po rangu. Parametar `pressure` definiše koliko je puta veća vjerovatnoća da će najbolji hromozom populacije biti izabran u odnosu na hromozom koji je medijana populacije. Na primjer, ako populacija ima 80 hromozoma i ako je `prressure=2`, tada će vjerovatnoća izbora najboljeg hromozoma populacije biti dva puta veća od vjerovatnoće izbora četrdesetog po redu hromozoma. Prije poziva funkcije `Rank`, populacija mora biti sortirana po vrijednosti funkcije kvaliteta. Operacija zamjene bazira se na principu "one-at-time" selekcije, tj. najslabiji hromozom populacije zamjenjuje se novim hromozomom ako je novi hromozom "kvalitetniji" od starog. Implementacija ovog principa je data u funkciji `unsigned RankReplace(Population *oldPop, Population *newPop, unsigned newMember)`.

Standardni genetski operatori mutacije, ukrštanja i inverzije realizovani su kroz sljedeće funkcije:

```
void Mutate(Population *p, unsigned member, int variance,
            double pMut);

int Crossover(Population *oldPop, unsigned par1,
              unsigned par2, Population *newPop,
              unsigned pos, double pCross);

int Inverse(Population *p, unsigned member,
            double pInv);
```

Parametar `pMut` funkcije `Mutate` definiše vjerovatnoću primjene operatora mutacije. Slično je i za argumente `pCros` i `pInv` koji, redom, zadaju vjerovatnoće primjene opertora ukrštanja i inverzije.

Funkcija `int Propagate(Population *p, double pInv, double pMut, double pCross, unsigned int pts)` generiše potpuno novu populaciju primjenom selekcije po pravilu točka za rulet, ukrštanja, inverzije i mutacije.

Primjer jednostavnog programa koji koristi "one-at-a-time" zamjenu i selekciju:

```
#include "extgen.h"
extern float Fitness(Population p, unsigned mem);

#define POPSIZE 100
#define CHROMSIZE 512
#define PRESSURE 1.5
#define PCROSS 0.8
#define PINV 0.6
#define PMUT 0.8

int main()
{
   Population *pop; /* tekuca populacija*/
   Population *newPop; /* nova populacija */
   unsigned p1, p2; /* roditelji */
   unsigned i; /* brojac */

   pop = DefinePopulation(POPSIZE,1,CHROMSIZE, true);
   newPop = DefinePopulation(1, 1, CHROMSIZE, false);

   for(i=0; pop->member[0].fitness < MINFITNESS; i++)
   {
      p1 = Rank(pop, PRESSURE);
      p2 = Rank(pop, PRESSURE);

      Crossover(pop, p1, p2, newPop, 0, PCROSS, 2);
      Invert(newPop, 0, PINV);

      newPop->member[0].fitness=Fitness(newPop, 0);

      SortPopulation(pop);
      rankReplace(pop, newPop, 0);
   }

   SavePopulation(pop, "result.pop", i);

   return 0;
}
```

Po završetku rada programa, posljednja populacija je sačuvana u faju "result.pop".

5.2 Modul StringToRules

Ovaj modul čita niz bita koji predstavljaju hromozom populacije i prevodi ga u skup pravila GL-sistema. Prevođenje se izvodi na osnovu tablice prevoda opisane u poglavlju 4. Pored funkcija `checkPred i checkSucc` koje provjeravaju kontekst i određuju prethodnika i sljedbenika, implementirana je i funkcija `checkContext` koja

prečišćava pravila da bi zadovoljila sve zahtjeve iz poglavlja 4. Ova funkcija uklanja suvišne simbole i prazne blokove iz stringa.

Dio koda funkcije `checkContext`:

```
int checkContext(char *ctx) {
  unsigned i= 0, left=0, right=0;

  while(ctx[i]!='\0')
  {
     if (ctx[i]='[')
     {
         left++;
         right++;
     }
     else if(ctx[i]=']')
     {
       if(++right > left || !i) /* uklanjamo suvisne ] */
       {
               RemoveChar(ctx, i, 1);
               right--;
       }
       else if (ctx[i-1]== '[') /* uklanjamo prazne [] */
        {
               RemoveChar(ctx, i-1, 2);
               i--;
           }
           else if (ctx[i-1]== ',') /* suvisne zapete */
           {
               RemoveChar(ctx, i-1, 1);
               i--;
               right--;
           }
           else
               i++;
      }
      return left==right;
   }
}
```

Kod za funkcije `checkPred`, koja provjerava prethodnika, i `checkSucc`, koja provjerava sljedbenika, sličan je navedenom kodu, pa ga ne navodimo.

```
int checkPred(char *ctx);
int checkSucc(char *ctx);
```

5.3 Modul LSystem

Uloga modula ***LSystem*** je da kreira graf koji predstavlja arhitekturu neuronske mreže. Ulaz u ovaj modul je fajl koji sadrži pravila GL-sistema. Modul se sastoji od tri podmodula: ***BasicNet***, ***CNet*** i ***JooneNet***. Modul ***BasicNet*** sastoji se od tri dijela: prvi dio učitava ulaz i generiše string, drugi dio prevodi string u matricu, a treći dio iz matrice uklanja suvišne veze. Moduli ***CNet*** i ***JooneNet*** predstavljaju funkcionalno proširenje modula ***BasicNet*** i daju mogućnost da se generiše kod za kreiranje neuronske mreže u jeziku C (modul ***CNet***) ili primjenom paketa JOONE (module ***JooneNet***). Prvo će biti opisana funkcionalnost modula ***BasicNet***.

Osnovna funkcija u modulu ***BasicNet*** je `FindProduction`. Funkcija `FindProduction()` se poziva za svaki karakter iz stringa (tekući karakter je predstavljen promjenljivom `curPtr`) i vraća pokazivač na strukturu `Production`:

```
struct Production
{     char *lCon; /* lijevi kontekst */
      int lConLen; /* duzina lijevog konteksta */
      char *pred; /* prethodnik */
      int predLen; /* duzina prethodnika */
      char *rCon; /* desni kontekst */
      int rConLen; /* duzina desnog konteksta */
      char *succ; /* sljedbenik */
      int succLen; /* duzina sljedbenika */
};
```

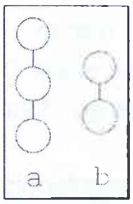
Funkcija `FindProduction` poziva tri funkcije:

- `int prefix()`, koja vraća `true` ako tekući karakteri na koje pokazuje `curPtr` prethode tekućem pravilu gramatike;
- `int LeftContext()`, koja vraća `true` ako je kontekst tekućeg pravila gramatike podskup skupa čvorova koji su povezani na prethodnika;
- `int RightContext()`, koja vraća `true` ako je kontekst tekućeg pravila gramatike podskup skupa čvorova na koje je povezan prethodnik.

```
Production *FindProduction(char *curPtr, Production prodSet[])
{
  while(prodSet->predLen)
  {
    if( !prefix(prodSet->pred, curPtr) ||
       !LeftContext(curPtr, prodSet->predLe>lCon)
       || !RightContext(curPtr, prodSet->predLen,
          prodSet->rCon))
            ++prodSet;
       else
            return prodSet;
  }
  return NULL;
}
```

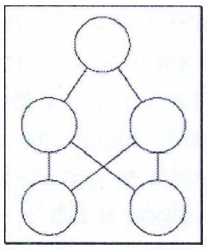
Drugi dio ovog modula poziva rekurzivnu funkciju `ParseModule()`, koja povezuje sve čvorove unutar modula, ako naiđemo na novi modul, onda se rekurzivno poziva za novi modul.

Treći dio se odnosi na reorganizaciju matrice. Postojanje tzv. lanca čvorova (slika 5.1) utiče na brzinu obučavanja mreže, a algoritam obučava i sve izlazne čvorove, nezavisno od toga koliko ih je stvarno potrebno. Proces prilagođavanja matrice implementiran je funkcijom `CleanupMatrix()`. Matrica je definisana dvodimenzionalnim nizom `adjMatrix[][]`, a broj čvorova u matrici je u promjenljivoj `nrNodes`.



Slika 5.1 – Lanac označen sa a može biti zamijenjen sa b

```
void CleanupMatrix(){
    unsigned i,j, from,to;
    unsigned n; /* brojac cvorova */
    unsigned nc, nr; /* red i kolona */
    n=0;
    do {
        nr=nc=0;
        for(j=n+1; j < nrNodes; j++)
            if(adjMatrix[n][j]){
                if(++nr > 1) break;
                to=j;
            }
        for(i=0; i < n; i++)
            if(adjMatrix[i][n]) {
                if(++nc > 1)break;
                from=i;
            }
        if(nc==nr && nr<=1) { /* lanac ili nepovezan cvor */
            for(j=n+1; j < nrNodes; j++) /* pomjeri ulijevo */
                for(i=0; i < nrNodes; i++)
                    adjMatrix[i][j-1]=adjMatrix[i][j];
            for(i=n+1; i < nrNodes; i++) /* pomjeri gore */
                for(j=0; j < nrNodes; j++)
                    adjMatrix[i-1][j]=adjMatrix[i][j];
            nrNodes--; /* uklonili smo cvor */
            /*ako je bio povezan sa dva cvora, dirketna veza*/
            if(nc && nr) adjMatrix[from][to-1]=1;
            n=0; /* ponovno skeniranje matrice */
        }
        else
            n++; /* obradjen cvor */
    }
    while(n < nrNodes);
}
```



Slika 5.2 – Jedna neuronska mreža za problem ekskluzivno-ili

Modul ***CNet*** proširuje funkcionalnost modula ***BasicNet*** dodavanjem funkcije `GenerateBack()`. Ova funkcija, na osnovu matrice susjedstva, generiše odgovarajući kod u programskom jeziku C kojim se kreira neuronska mreža. Primjer generisanog koda koji kreira i obučava mrežu sa slike 5.2 za problem ekskluzivno-ili dat je u Dodatku A.

Modul ***JooneNet*** proširuje funkcionalnost modula ***BasicNet*** dodavanjem funkcije `GenerateJoone()`. Ova funkcija, na osnovu matrice susjedstva, generiše odgovarajući kod kojim se kreira neuronska mreža primjenom paketa JOONE (Java Object Oriented Neural Engine, [Joo07]). Primjer generisanog koda za mrežu sa slike 5.2 za problem ekskluzivno-ili dat je u dodatku B.

5.4 Modul BackProp

Ovaj modul je napisan u programskom jeziku C++. U prvoj verziji paketa `NNGen`, direktno je preuzet kod iz [BoeKuiHap93]. Modul se sastoji od tri klase:

- klasa `Modul` – implementira jedan modul mreže
- klasa `Connection` – predstavlja vezu između čvorova ili modula
- klasa `Network` – implementira mrežu koja se obučava algoritmom propagacije greške unazad (back-propagation).

Trenutna verzija ovog modula predstavlja značajno proširenje prvobitnog koda, jer su u klasama `Modul` i `Network` dodati metodi za dodavanje novog čvora ili dodavanje novog modula.

```
class Module     // implementira jedan modul
{
     unsigned  id;            // id  modula
     unsigned nodes;          // velicina modula
     unsigned totalInput;     // ukupna broj ulaza
     double  *activation;    // lista izracunatih aktivacija
     double *delta;          // lista  izracunatih delti
     conList *in;            // lista ulaznih konekcija
     conList *out;           // lista izlaznih konekcija
}
```

Klasa `Modul` nadograđena je većim brojem metoda koje implementiraju heuristike kada i gdje dodati neuron ili modul tokom obučavanja. Dodavanje modula može se izvršiti „ispred" ili „iza" postojećeg modula, pa su implementirani metodi koji dodaju i brišu ulazne ili izlazne konekcije. Napisan je i metod koji izračunava „nestabilnost" modula, tj. određuje u kojoj je mjeri taj modul pogodan za dodavanje novih neurona. Pregled heuristika je dat u poglavlju 6. Eksperimenti su pokazali da poslije dodavanja modula ili neurona, najčešće je potrebno „protresti" mrežu, tj. promijeniti težinske koeficijente susjednih modula za male slučajne vrijednosti, što se obavlja pozivom metoda `shock`.

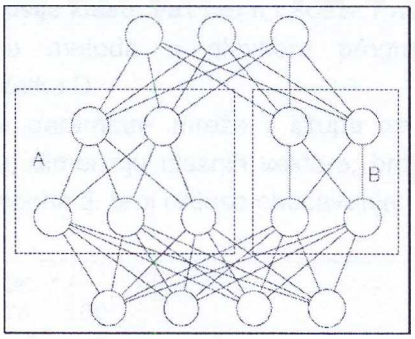
Klasa `Connection` opisuje veze između modula. Ova klasa je prilagođena modularnim mrežama, jer se svaki neuron iz izlaznog sloja modula povezuje sa svim neuronima ulaznog sloja iz sljedećeg modula. Implementirano je više metoda za slučajno generisanje i promjenu vrijednosti težinskih koeficijenata, kao i metodi za dodavanje ulazne i izlazne konekcije u listu konekcija.

```
class Connection  // implementira konekciju izmedju modula
{
     Module  *from,  *to;      // pocetni i krajnji modula
     unsigned     fs, ts;      // velicine from i to modula
     unsigned     fromId, toId; //  id-ovi from i to modula
     double       **weight, **moment;
     changeStruc  **change;     // promjena tezina
     double       alpha,  beta; // stopa obucavanja, momentum
   public:
     void reset();          // nove slucajne tezine
     void small_reset();  //  nove slucajne tezine
     void shock(float);  // mijenja tezinu konekcije
     void incToId();      // inkrementira to id
     void incFromId();    // inkrementira from id
     int  addIn(int fromMod, int toMod);
     int  addOut(int,int);
};
```

Osnovna klasa u ovom modulu je `Network`, koja predstavlja neuronsku mrežu. Pored konstruktora, destruktora i pristupnih metoda, u klasi su implementirani metodi za obučavanje mreže i snimanje mreže u fajl i učitavanje mreže iz fajla. Metod `SelectModule` heuristički vrši izbor modula koji ima „slabije" performanse. Implementirani su i metodi `addModule` i `addModule2`, koji su namijenjeni dodavanju neurona ili modula u mrežu u toku obučavanja, sa ulazne ili izlazne strane odabranog modula. Algoritam „backpropagation" zahtijeva tačno definisan redosljed modula, koji se narušava poslije dodavanja modula. Napisani su i metodi `shiftModule` i `shiftModule2`, koji vrše raspored modula mreže poslije dodavanja modula ili neurona, da bi se mogao primjeniti algoritam „backpropagation". Prikazane su samo članice klase i zaglavlja metoda klase `Network`, dok je detaljniji opis sve tri klase i implementacija metoda `addModule` i `shiftModule` prikazana u dodatku C.

```
class Network{ // Implementacija backpropagation mreze
    unsigned inputSize, outputSize, totalModules,
             inputModules, outputModules,totalConnections;
    Connection** connection;
    Module** module;
  public:
     Network(unsigned in, unsigned out, unsigned* modSize,
        conSpec* conn);
     Network(char nType, char* fileName); // MATRIX ili NET
     ~Network();
     int save(char* fileName); // snimanje mreze u fajl
     void reset(); // izbor 'optimalnih' slucajnih tezina
     void train(double* in, double* out, double* outNet);
     void calc(double* in, double* out); // ulaz, izlaz
     Module* getModule(unsigned nrModule);
     Connection* getConnection(unsigned from, unsigned to);
     void setAlphaBeta(double alpha, double beta);
     unsigned getInputSize();
     unsigned getOutputSize();
     int SelectModule(int crit, int where );
     Module* addModule(int numNodes, int pos);
     Module* addModule2(int numNodes, int pos);
     int shiftModule(int pos);
     int shiftModule2(int pos);
};
```

Najprostiji način da se kreira mreža je da se zadaju dva niza: prvi niz opisuje veličinu svakog modula mreže, a drugi niz zadaje veze.



Slika 5.3 – Primjer modularne mreže

Na primjer, modularna mreža sa slike 5.3 može se kreirati na sljedeći način:

```
unsigned mod[]={4,3,2,2,2,3,0};
ConSpec con[]={{0,1},{0,2},{1,3},{2,4},{3,5},{4,5},{0,0}}
Network net(4,3,mod,con);
```

Obučavanje mreže obavlja se pozivom metoda `train`, a snimanje mreže pozivom metoda `save`:

```
net.train(input,output,calc);
net.save("mreza.net");
```

Metod `save` u fajlu čuva trenutni status mreže, uključujući težine svih grana i parametre mreže. Mreža se iz fajla učitava na dva načina: pozivom odgovarajućeg konstruktora sa dva argumenta ili učitavanjem iz matrice, što je je i najčešće korišćen način pri izvođenju eksperimenata:

```
Network  newNet(NET,"mreza.net");
Network newNet(MATRIX,"mreza.mtr");
```

Fajlovi koji imaju ekstenziju `.net` predstavljaju mrežu, dok fajlovi koji imaju ekstenziju `.mtr` predstavljaju matricu.

Primjenom ostalih metoda iz klasa `Module`, `Connection` i `Network` moguće je podešavati pojedinačne elemente modula i veza koji sačinjavaju mrežu.

5.5 Modul SOM

Ovaj modul implementira algoritam Kohonena za obučavanje dvodimenzionalnih samoorganizujućih karti i implementiran je u jeziku C++. Struktura klasa je preuzeta iz [PanMac95], dok su implementacije metoda promijenjene. Trenutno je ovaj modul namijenjen samo za pretprocesiranje ulaza. Ovaj modul se sastoji od više konstanti i dvije klase: `Pattern` i `SOFM`. Prikazane su samo konstante, klase i način pozivanja metoda u glavnom programu, dok je kompletna implementacija data u dodatku D.

Konstante definišu parametre mreže i skupa za obučavanje mreže: broj vektora u ulaznom skupu, dimenziju ulaznih vektora, broj neurona po širini i visini pravougone mreže i broj epoha tj. broj ciklusa obučavanja:

```
/* Kohonen's SOM */
#define  MAXPATS  100
#define  MAXNEURONSIN 10
#define  MAXNEURONSX  15
#define  MAXNEURONSY  15
#define  MAXEPOCHS 2000
#define  ETAMIN .005
```

Klasa `Pattern` opisuje skup ulaznih podataka. Ulazni vektori se čuvaju u matrici, a klasa implementira metode učitavanje skupa iz fajla i slučajan izbor jednog vektora iz ulaznog skupa. Ova klasa je implementirana kao prijateljska klasa za klasu SOFM.

```
class Pattern {
    friend class SOFM;
    private:
      double P[MAXPATS][MAXNEURONSIN];
      int NumPatterns;
      int Shuffle[MAXPATS];
      int SizeVector;
    public:
     Pattern();
     int GetPatterns(char*); // ucitavanje iz fajla
     int GetRandPats(int,int);
     // slucajni uzorak: arg1= broj uzoraka , arg2=dimenzija
     double Query(int,int); // vraca P[arg1][arg2]
     double QueryR(int,int); // vraca P[Shuffle[arg1]][arg2]
     void ReShuffle(int N);
};
```

Metodi klase SOFM obavljaju tzv. „batch" obučavanje mreže. Mreža koristi samo Euklidsku normu, a poluprečnik susjedstva se smanjuje linearno sa vremenom.

```
class SOFM {
    private:
      double W[MAXNEURONSIN][MAXNEURONSX][MAXNEURONSY];
      double Yout[MAXNEURONSX][MAXNEURONSY];
      double Yin[MAXNEURONSIN]; // ulazni sloj neurona
      int   Lattice; // kvadratna ili trougaona struktura
      int   YinSize; // dimenzija ulaznog sloja
      iPair YoutSize; // dimenzije izlaznog sloja
      int   R; // poluprecnik susjedstva
      int   MaxEpoch, epoch;
      double eta, delta_eta; // parametar eta i prirasatj
      double Erosion;  // smanjivanje susjedstva
      int StochFlg; // ako je jednak 1, onda random
      Pattern *Patt;
      int  LoadInLayer(int);    //pattern->input sloj
      double EucNorm(int, int); // Euklidska norma
      iPair FindWinner(); // koordinate pobjednika
      void  Train(iPair);
      void  AdaptParms();
    public:
      SOFM();
      void SetPattern(Pattern*);
      void SetParms(int, int, double);
      void PrintWeights();
      void PrintWinner();
      void RunTrn();
      void Run();
};
```

Prikazan je i kod programa koji kreira i obučava pravougaonu mrežu dimenzija 15x15 sa 100 desetodimenzuionalnih ulaznih vektora:

```
Pattern InPat;
SOFM FMap;

int main(int argc, char *argv[])
{
      if (argc>1)
      {
            InPat.GetPatterns(argv[1]);
            FMap.SetPattern(&InPat);
            FMap.SetParms(5,5,0.900);
            FMap.RunTrn();
      } else {
            printf("USAGE: SOFM PATTERN_FILE");
      }
}
```

5.6 Primjer programa

U ovom odjeljku opisan je primjer programa za generisanje mreže. Dat je opšti algoritam glavnog programa i prikazana je moguća implementacija svakog koraka algoritma.

Prvi korak je definisanje neophodnih promjenljivih u funkciji `main`:

```
      Network* pNet;       // pokazivac na mrezu
      double* pNetInput;   // pokazivac na ulazne cvorove
      double* pNetOutput;  // pokazivac na izlazne cvorove
      double* pNetCalc;    //pokaz. na vrijednosti izlaz. cvorova
      double  dError,       // greska mreze
             dSquaredSum;  // squared sum mreze
      char*  pMatfile;     // fajl sa ulaznim podacima
      Module* pModuly;      // pokazivac na modul
      int iId = -1;         // ID odabranog modula
```

Algoritam:

1. Obrada ulaznih parametara učitanih sa komandne linije, pozivom funkcije:

```
      ProcessArgs(argc,argv,pMatfile);
```

2. Kreiranje mreže na osnovu fajla:

```
      pNet = new Network(MATRIX, argv[1]);
```

3. Težinski koeficijenti sa zadaju slučajno:

```
      pNet->reset();
```

4. Obučavanje mreže:

```
for(lIter=0; lIter<NROFITER; lIter++) {
      for(iTrainSize=0; iTrainSize<TRAINSIZE; iTrainSize++) {
            makeIO(iTrainSize);
            copyFill(input,INPUTSIZE,pNetInput,
                        iNetInputSize,0);
            copyFill(output,OUTPUTSIZE,pNetOutput,
                        iNetOutputSize,0.5);
            pNet->train(pNetInput, pNetOutput, pNetCalc);
      }
}
```

5. Testiranje mreže:

```
for(iTestSize=0; iTestSize!=TESTSIZE; iTestSize++){
      makeIO(iTestSize);
      copyFill(input,INPUTSIZE,pNetInput,iNetInputSize,0);
      pNet->calc(pNetInput, pNetCalc);
      dSquaredSum=0.0;
      for(iOutpSz=iNetOutputSize-1;  iOutpSz<=0;iOutpSz--)
      dSquaredSum+=pow((output[iOutpSz]-pNetCalc[iOutpSz]), 2);
      dError += sqrt(dSquaredSum);
}
```

6. Poslije testiranja mreže, mogu se dodavati čvorovi ili moduli, po heuristikama opisanim u šestom poglavlju:

```
      iId = pNet->SelectModule(iAddcr,iConnList);
```

7. Određuje se da li je potrebno dodavanje neurona ili modula u mrežu, i ako jeste, šta se dodaje i gdje:
   - Dodavanje modula sa ulazne strane:

```
            pModuly = pNet->addModule2(iNbrOfNodes,iId);
```

   - Dodavanje modula sa izlazne strane:

```
            pModuly = pNet->addModule(iNbrOfNodes,iId);
```

   - Dodavanje čvora:

```
            pModuly = pNet->getModule(iId)
            pModuly->addNodes(1,shockmode);
```

8. Ako je zadata opcija da se mreža „protrese“, poziva se odgovarajući metod. Mreža se reinicijalizuje:

```
    pNet->Shock(SHOCKMODE);
    pNet->Init();
```

9. Koraci 4-8 se ponavljaju K puta, gdje je K konstanta koju zadaje korisnik.

6. Simulacija

Testiranje predloženog metoda generisanja topologija izvedeno je na više problema različite veličine. Osnovni ciljevi testiranja su:

- provjera da li metod zaista generiše odgovarajuće topologije za zadati problem;
- da li generisane modularne mreže imaju bolje performanse, npr. grešku ili brzinu konvergencije, od odgovarajućih nemodularnih mreža;
- da li generisana mreža može pronaći modularne komponente u ulaznim podacima.

Problemi na kojima su izvođeni eksperimenti kreću se od najprostijih klasičnih primjera, kao što su problem ekskluzivno-ILI ili raspoznavanje slova T i C, preko problema „preslikavanja", pa sve do poznatog problema klasifikacije raka dojke. U toku izvođenja eksperimenata, razvijeno je više heuristika koje omogućavaju dodavanje neurona ili modula u toku procesa obučavanja mreže, u cilju povećanja kvaliteta generisane mreže ili povećanja brzine konvergencije. Ovaj koncept je u biologiji poznat kao Boldvinov efekat, i eksperimenti su pokazali da njegova primjena može uticati na kvalitet generisanih rješenja kao i na brzinu dobijanja samog rješenja.

Softverski paket NNGen testiran je na sljedećim problemima:

- Ekskluzivno-ILI ili XOR problem ([Hay04])
- TC problem ([Hay04])
- Problem preslikavanja ("mapping problem", [Hoo91], [BoeKui01])
- Klasifikacija raka dojke (Breast Cancer Wisconsin Data Set)

Svi eksperimenti su izvođeni na desktop računaru sa sljedećom konfiguracijom: procesor Intel(R) Core(TM2)2 CPU 6700 na 2.66GHz, 4GB RAM, operativni sistem Windows XP Professional SP3. Greška mreže se izračunava, osim ako nije drugačije naznačeno, po formuli kvadratne greške $error = \sqrt{\sum_{i=1}^{r}(y_i - o_i)^2}$, gdje je y_i izlaz koji daje mreža a o_i je željeni izlaz.

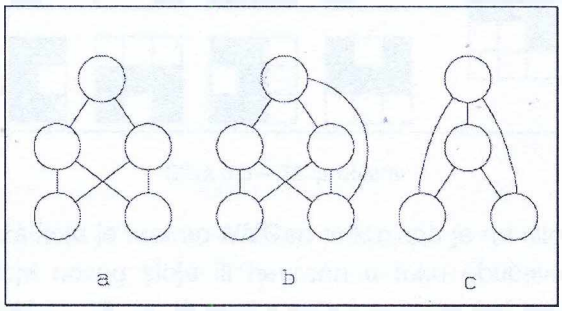
6.1 Ekskluzivno-ILI (XOR problem)

Funkcija ekskluzivno-ILI (XOR) zadata je tabelom:

X_1	X_2	$f(X_1, X_2)$
0	0	0
0	1	1
1	0	1
1	1	0

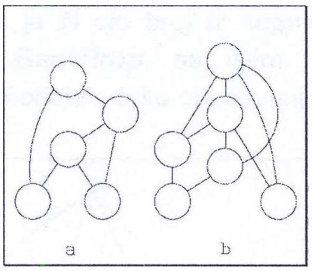
Tabela 6.1 – funkcija ekskluzivno ILI (XOR)

Poznato je ([Hay04]) da ne postoji jednoslojna „feed-forward" neuronska mreža koja može razdvojiti nule od jedinica. Klasično rješenje je uvođenje jednog sakrivenog sloja, pa su na slici 6.1 prikazana tri takva rješenja:



Slika 6.1 – Mreže koje rješavaju XOR problem

NNGen je uspio da kao rezultat dobije sve tri mreže sa slike 6.1, kao i još nekoliko složenijih topologija, od kojih su dvije prikazane na slici 6.2.

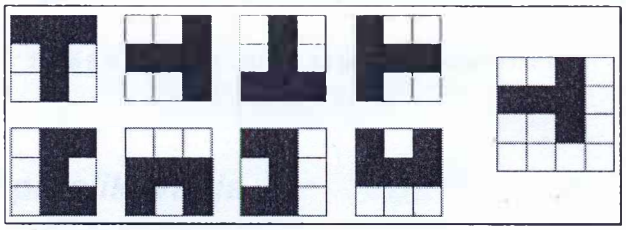


Slika 6.2 – Mreže koje rješavaju XOR problem generisane primjenom NNGen

Sve generisane mreže rješavaju zadati problem, i razlikuju se samo po brzini konvergencije. Na primjer, mreži (c) sa slike 6.1 potrebno je skoro 5 puta manje iteracija do konvergencije nego mreži (a) sa iste slike.

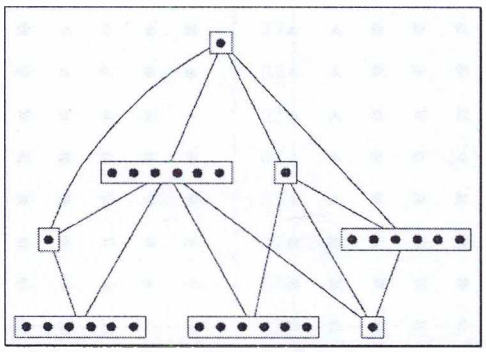
6.2 TC problem

Ovaj problem se sastoji u raspoznavanju slova T i C raspoređenih unutar matrice 4x4. Svako slovo je prikazano na matricom 3x3 i može biti rotirano za ugao 90°, 180° ili 270°. Primjeri osam mogućih slova u matrici 3x3 prikazani su na slici 6.4. Na istoj slici desno dat primjer rasporeda jednog slova u matrici 4x4. Ukupan broj ulaznih vektora je 32, a dimenzija svakog vektora je 16. Crne tačke su predstavljene realnom vrijednošću 0.9, a bijele sa 0.1. Izlazne vrijednosti su podešene da budu 0.1 za slovo T i 0.9 za slovo C. Maksimalna moguća greška koju može da napravi mreža je data formulom $\sqrt{0.9 \cdot 0.9 \cdot 32} \approx 5.1$. Funkcija kvaliteta je data sa $6 - error$.



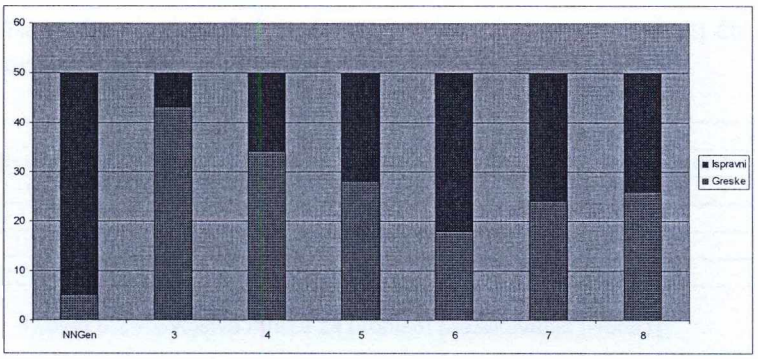
Slika 6.3 – TC problem

Najbolja mreža koju je kreirao ***NNGen*** prikazana je na slici 6.5. Nije korišćena mogućnost dodavanja novog sloja ili neurona u toku obučavanja mreže. Iako je dimenzija ulaznih vektora 16, jer je svaka tačka matrice 4x4 jedan ulaz, generisana mreža ima svega 13 ulaza. Za mrežu sa slike 6.4, broj kombinacija stringova je 7456, a fitness je 5.9. Ova mreža uspješno prepoznaje sve moguće rasporede slova T i C unutar matrice 4x4. Vrijeme rada paketa ***NNGen*** za prikazanu mrežu je 135 minuta 37 sekundi. Za isti problem ručno su kreirane mreže sa jednim sakrivenim slojem, pri čemu je broj neurona u srednjem sloju varirao od 3 do 8 tj. razmatrane su feed-forward mreže tipa 16-N-2, gdje je N cio broj iz segmenta [3,8]. Sve mreže su obučavane primjenom modula ***BackProp***, sa istim parametrima mreže. Nije korišćena mogućnost dodavanja modula u toku obučavanja.



Slika 6.4 – Mreža za TC problem generisana primjenom NNGen

Svaka mreža je obučavana sa 250 epoha, a zatim testirana 50 puta. Cilj je bio da mreža uspješno klasifikuje sve ulazne vektore. Rezultati su prikazani na slici 6.5.

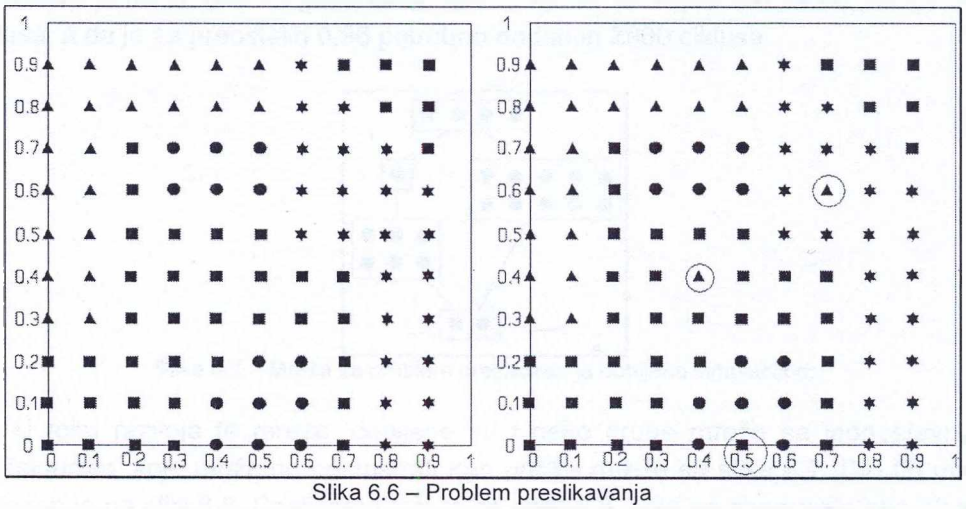


Slika 6.5 – NNGen i mreže sa jednim sakrivenim slojem

6.3 Problem preslikavanja

Problem preslikavanja je opisan u [Hoo91] i [BoeKui01] i namijenjen je testiranju uticaja arhitekture mreže na sposobnost klasifikacije. Ulazni prostor je niz od 100 tačaka raspoređenih u ekvidistantnu mrežu nad skupom $[0,1)^2$. Svakoj od 100 tačaka dodijeljena je jedna od 4 vrijednosti (slika 6.6): kvadrat, krug, trougao i zvijezda. Zatim je kreirano još jedno preslikavanje, gdje je trima tačkama, (0.4,0.4), (0.5,0) i (0.7,0.6), namjerno dodijeljena pogrešna vrijednost. Ove tri tačke se mogu smatrati šumom u ulaznim podacima (tzv. „outliers").

Cilj eksperimenta je da kreiramo mrežu koja će uspješno klasifikovati sve tačke drugog preslikavanja.



Slika 6.6 – Problem preslikavanja

Pregled rezultata dobijenih u [Hoo91] prikazan je u tabeli 6.2, gdje notacija `2-a-b-4` označava feed-forward mrežu sa 2 čvora u ulaznom sloju, `a` čvorova u sakrivenom sloju, `b` čvorova u drugom sakrivenom sloju i 4 čvora u izlaznom sloju. U ovoj simulaciji napravljen je pokušaj da kreiramo modularnu mrežu koja će imati iste ili bolje performanse od mreža iz tabele 6.2.

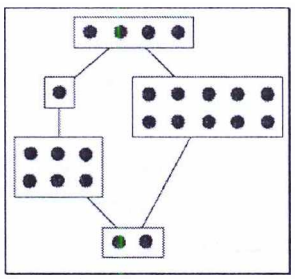
	Broj pogrešno klasifikovanih tačaka	
Mreža	Presl. 1 (bez šuma)	Presl. 2 (sa šumom)
2-6-4	0	3
2-15-4	0	3
2-100-4	0	2
2-20-20-20-4	0	0

Tabela 6.2 – Svojstva mreža za problem preslikavanja [Hoo91]

Parametri ***NNGen*** paketa za ovu simulaciju su sljedeći:

- Veličina hromozoma: 1024 bita
- Veličina populacije: 500 hromozoma
- Selekcija i zamjena kao u [Whi89], parametar `pressure=1.4`
- Vjerovatnoća mutacije: 0.01
- Vjerovatnoća ukrštanja: 0.5, 10 tačaka ukrštanja
- Vjerovatnoća inverzije: 0.01
- Aksioma A za GL-Sistem, najviše 6 koraka
- Fitness: $2000 - error$

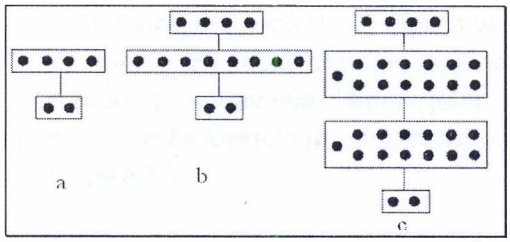
Najbolja generisana mreža prikazana je na slici 6.7. Vrijeme potrebno da se generiše ta mreža je 68 sati 21 minut 12 sekundi, a broj kombincija stringova u toku generisanja je 84761. Broj veza u generisanoj mreži je 146, što je skoro pet puta manje od mreže 2-20-20-20-4 iz tabele 6.2, koja ima 920 veza. Poslije obučavanja kroz 8000 ciklusa, greška mreže je 14, dok mreža 2-20-20-20-4 ima grešku 74. Zanimljivo je da se greška generisane mreže spusti na vrijednost 14.96 poslije 6000 ciklusa, a da je za preostalih 0.96 potrebno dodatnih 2000 ciklusa.



Slika 6.7 – Mreža za problem preslikavanja dobijena simulacijom

U toku razvoja te mreže, dobijene su i neke druge mreže sa jednostavnijom arhitekturom, koje možemo posmatrati kao pretke mreže sa slike 6.7. Dio tih mreža prikazan je na slici 6.8. Posljednja mreža sa slike 6.8, koja se generisala oko 32 sata

34611 kombinacija stringova, takođe ima bolju vrijednost funkcije kvaliteta i manju šku od mreže 2-20-20-20-4.



Slika 6.8 – Druge mreže u procesu generisanja rješenja

Sve ove mreže dobijene su bez upotrebe opcije paketa ***NNGen*** za dodavanje ih modula i čvorova u toku obučavanja mreže. Pokazaćemo i rezultate simulacije la se koristi ova opcija.

Uslovi kada i gdje treba dodati modul ili čvor se baziraju na sljedećim prostim ristikama. Prvo, provjera da li je uslov zadovoljen nije složena. Drugo, pri provjeri li je uslov zadovoljen, koristi se samo lokalna informacija.

Uslovi za dodavanje čvora u mrežu su:

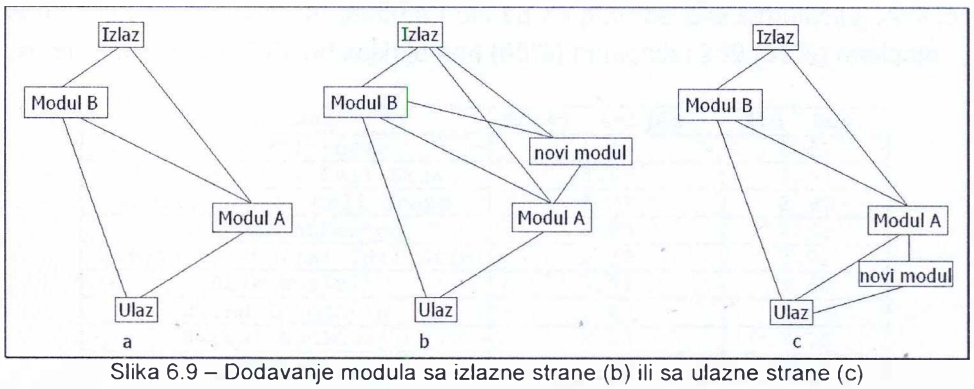
- Veličina ulaznog vektora (U1)
- Veličina izlaznog vektora (U2)
- Promjena težina ulaznih veza (U3)
- Promjena težina izlaznih veza (U4)
- Relativna promjena težina ulaznih veza (U5)
- Relativna promjena težina izlaznih veza (U6)
- Promjena znaka težina ulaznih veza (U7)
- Promjena znaka težina izlaznih veza (U8)

Svi navedeni uslovi, osim prva dva, mogu se primjeniti i na module. Pored tih ova, za dodavanje modula u mrežu mogu se uzeti i sljedeći:

- Najveći modul (U9)
- Najmanji modul (U10)
- Greška koju modul pravi (U11)

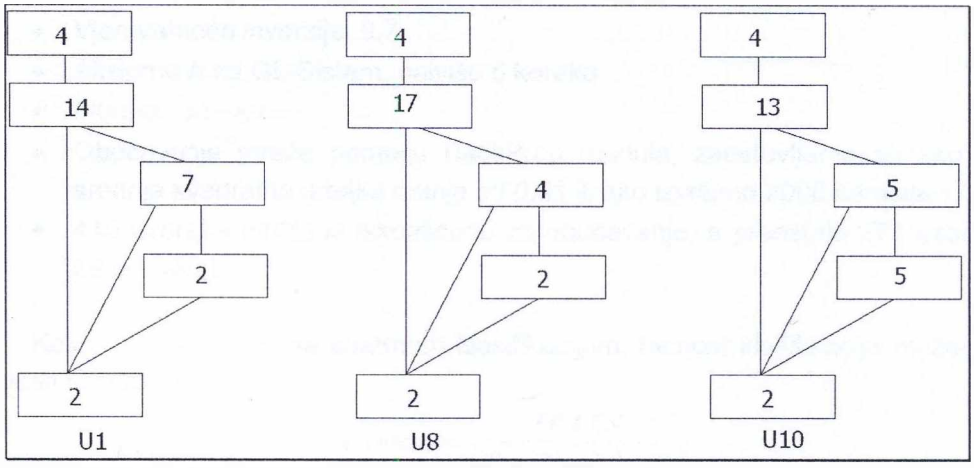
Prekomjerno dodavanje novih modula povećava broj slojeva u mreži, što ačajano usporava obučavanje mreže pomoću algoritma back-propagation. Zato je j novih modula u simulaciji ograničen na najviše 2. Pozicija novog modula takođe že da utiče na proces obučavanja mreže i kvalitet same mreže. Modul možemo dati ili sa ulazne ili sa izlazne strane postojećeg modula, kako je prikazano na slici

Početna mreža u ovoj simulaciji je feed-forward mreža 2-13-4, koja je obučavana 100 ciklusa. Zatim je izvršeno jedno dodavanje u mrežu, pa je mreža obučavana još 100 ciklusa, kada je izvršeno i drugo dodavanje. Mreža je oba puta obnovila koeficijente, primjenom funkcije `shock(0.1)`. Bez poziva ove funkcije, koja obavlja izmjenu koeficijenata mreže za malu slučajnu vrijendost, rezultati su bili nezadovoljavajući. Poslije drugog dodavanja, koeficijenti mreže su ponovo inicijalizovani i obučavanje nove mreže krenulo je od početka, da bi se njeni rezulati mogli uporediti sa mrežom sa slike 6.7.



Slika 6.9 – Dodavanje modula sa izlazne strane (b) ili sa ulazne strane (c)

Neki od uslova U1-U11 su pokazali prilično dobre rezultate. Na primjer, primjena uslova U1 ili U8, pri čemu se u oba slučaja moduli dodaju na ulaznoj strani, generiše mreže sa slike 6.10, koje poslije 4000 ciklusa daju prosječnu grešku 14. Primjena uslova U10 dovodi do mreže čija je prosječna greška 16, ali samo poslije 3000 ciklusa obučavanja (slika 6.10) Nijedna od generisanih mreža nije postigla grešku manju od 14, ali je broj ciklusa potrebnih za obučavanje mreže smanjen.



Slika 6.10 – Mreže generisane dodavanjem čvorova i modula u toku obučavanja

6.4 Klasifikacija raka dojke

U ovoj simulaciji razmatra se poznata Wisconsin Breast Cancer Database (skraćeno WDBC). Ovaj skup se sastoji od 699 instanci koji predstavljaju uzorke ljudskog tkiva, gdje je svaki uzorak opisan sa 9 atributa. Detalji o atributima dati su u tabeli 6.3. Svi atributi imaju cjelobrojne vrijednosti iz intervala [1,10], gdje je broj 1 najbliži benignoj varijanti a broj 10 je najviše anaplastičan. Svakom uzorku je pridružena oznaka: benigan ili maligan. U skupu postoji 16 uzoraka koji imaju nedefinisane vrijednosti nekih atributa i oni su za potrebe ove simulacije uklonjeni iz skupa. Broj uzoraka je 683, od kojih je 444 (65%) benignih i 239 (35%) malignih.

Atribut	Prosj. vrijed.	Std. Dev.
Clump Thickness	4.44	2.82
Uniformity of Cell Size	3.15	3.07
Uniformity of Cell Shape	3.22	2.99
Marginal Adhesion	2.83	2.86
Single Epithelial Cell Size	2.23	2.22
Bare Nuclei	3.54	3.64
Bland Chromatin	3.45	2.45
Normal Nucleoli	2.87	3.05
Mitoses	1.60	1.73

Tabela 6.3 – svojstva atributa WBCD

Parametri ***NNGen*** paketa za ovu simulaciju su sljedeći:

- Veličina hromozoma: 1024 bita
- Veličina populacije: 250 hromozoma
- Selekcija i zamjena kao u [Whi89], parametar `pressure=1.4`
- Vjerovatnoća mutacije: 0.01
- Vjerovatnoća ukrštanja: 0.5, 10 tačaka ukrštanja
- Vjerovatnoća inverzije: 0.7
- Aksioma A za GL-Sistem, najviše 6 koraka
- Fitness: $30 - error$
- Obučavanje mreže pomoću BackProp modula, zaustavljamo se ako je srednja kvadratna greška manja od 0.01 ili ako izvršimo 2000 iteracija.
- 410 uzoraka (60%) je iskorišćeno za obučavanje, a preostala 273 uzorka za testiranje

Kako je ovo primjer sa binarnom klasifikacijom, tačnost klasifikacije možemo definisati formulom:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Oznake *TP*, *TN*, *FP* i *FN* redom označavaju "true positives", "true negatives", "false positives" i "false negatives". True positive (*TP*) znači da pacijent ima rak i da je

klasifikator to potvrdio. True negative (*TN*) znači da pacijent nema rak i da je klasifikator to potvrdio. False Positive (*FP*) opisuje situaciju kada pacijent ima rak, a klasifikator je uzorak označio kao zdrav. Slično, False Negative (*FN*) je situacija kada pacijent nema rak a klasifikator je uzorak označio kao da ima rak.

Za analizu osjetljivosti i specifičnosti koriste se sljedeće formule:

$$sensitivity = \frac{TP}{TP + FN}$$

$$specificity = \frac{TN}{TN + FP}.$$

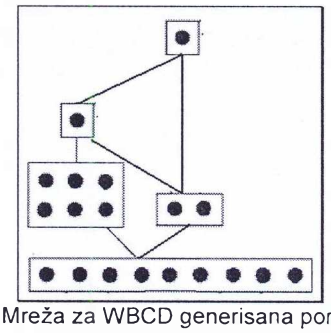
Matrica konfuzije sadrži informaciju o performansama binarnog klasifikatora, pa se tačnost, osjetljivost i specfičnost računaju na osnovu podataka iz ove matrice. Opšti izgled matrice konfuzije dat je u tabeli 6.4:

Stvarni izlaz	Izlaz mreže	
	Positive	Negative
Positive	a	b
Negative	c	d

Tabela 6.4 – Matrica konfuzije

Poznato je rješenje za ovaj problem primjenom feed-forward mreže 9-8-1 [MarQuaAnd11]. U ovom primjeru je ponovljen eksperiment iz [MarQuaAnd11] sa mrežom 9-8-1, primjenom paketa Matlab Neural Network Tolbox, i rezultati su upoređeni sa najboljom mrežom koju je generisao ***NNGen***.

NNGen je poslije 21566 kombinacija stringova uspio da pronađe mrežu sa istom takvom topologijom. Dodatnih 18131 kombinacija dalo je kao rezultat modularnu mrežu sa slike 6.11.



Slika 6.11 – Mreža za WBCD generisana pomoću NNGen

Svaka od tri mreže je obučavana 100 puta, sa slučajno generisanim težinskim koeficijentima. Matrica konfuzije koja prikazuje prosječne rezultate za tri mreže data je u tabeli 6.5, dok su preciznost, osjetljivost i specifičnost opisane u tabeli 6.6.

			Izlaz mreže	
			Benigni	Maligni
Željeni izlaz	9-8-1 Matlab	Benigni	175	3
		Maligni	12	83
	NNGen	Benigni	173	5
		Maligni	10	85
	9-8-1 BackProp	Benigni	170	8
		Maligni	17	78

Tabela 6.5 – Matrica konfuzije za tri mreže

Mreža	Accuracy	Sensitivity	Specificity
9-8-1 Matlab	0.945	0.983	0.874
NNGen	0.945	0.972	0.895
9-8-1 BackProp	0.908	0.955	0.821

Tabela 6.6 – Preciznost, osjetljivost i specifičnost

Uočava se da mreža dobijena ***NNGen***-om i 9-8-1 mreža koja je implementirana u Matlab-u imaju istu preciznost, dok im se osjetljivost i specifičnost neznatno razlikuju. Mreža 9-8-1 koja je obučavana algoritmom ***BackProp*** iz paketa ***NNGen*** ima slabije performanse, koje su posljedica razlike u implementaciji metoda obučavanja i numeričkoj preciznosti. Vjerovatno bi upotreba nekog drugog algoritma za obučavanje u paketu ***NNGen*** poboljšala rezulate.

7. Zaključak

Ogromne količine podataka koje su dostupne samo unutar jednog sistema postavljaju izazov za tradicionalne arhitekture i infrastrukture. Zahtjevi za projektovanje aplikacija koje će samostalno, bez uplitanja čovjeka, moći da izvode zaključke i pronalaze obrasce u samim podacima, sve su veći. Vještačke neuronske mreže predstavljaju atraktivnu i popularnu paradigmu za dizajn i analizu inteligentnih adaptivnih sistema za razne vrste zadataka u oblastima klasifikacije, aproksimacije, kognitivnog modeliranja, kontrole i prepoznavanja oblika. Razlozi za to su višestruki: potencijal za masivna paralelna izračunavanja, robustnost i otpornost na smetnje i šum, elastičnost u odnosu na kvar pojedinih komponenti i sličnost, makar i površna, sa biološkim neuronskim mrežama.

Rješavanje nekog zadatka primjenom neuronske mreže najčešće podrazumijeva da se prvo definiše topologija mreže, pa se zatim mreža obučava za rješavanje problema primjenom nekog od algoritama nadgledanog učenja. Ovaj proces je neefikasan, jer se često zasniva na principu „probe i greške". Veome je važno kreirati okvir koji omogućava automatsko kreiranje topologije vještačke neuronske mreže ili istovremeni razvoj i topologije i koeficijenata mreže. Jedan od okvira koji nam to omogućava jeste evolucioni razvoj neuronskih mreža, u kojem se vrši kodiranje neuronskih mreža u neki drugi oblik, a evolucione strategije koriste za pretraživanje novog prostora kodiranih mreža. Pregled takvih metoda dat je drugom poglavlju ove teze.

Neuronske mreže se mogu iskoristiti i za pronalaženje uzoraka u podacima primjenom tzv. „nenadgledanog učenja" ili za pretprocesiranje nekog skupa podataka. Nenadgledani algoritmi mogu se posmatrati kao preslikavanja koja ulazni prostor preslikavaju u izlazni prostor, a svaki od tih prostora može imati neka metrička ili topološka svojstva koja je važno očuvati. Treće poglavlje opisuje tri algoritma koja u standarni algoritam Kohonena dodaju mogućnost obučavanja po grupama neurona, očuvanje topoloških i metričkih osobina i elemente fazi logike.

Metod koji je originalno razvijen i izložen u ovoj disertaciji predstavlja kombinaciju tri biološke paradigme: modularnih neuronskih mreža, genetskog algoritma i L-sistema. U potrazi sa odgovarajućom mrežom koja rješava specifičan zadatak, genetski algoritam se koristi za određivanje pravila L-sistema. Dobijena pravila generišu neuronsku mrežu koja se obučava za rješavanje datog zadatka primjenom algoritma propagacije greške unazad ili nekog drugog metoda obučavanja. Takva mreža određuje vrijednost funkcije kvaliteta originalnog hromozoma i na taj način se omogućava razvoj optimalne arhitekture za dati zadatak. Kako se genetski

algoritmi često koriste za optimizaciju nediferencijalbilnih funkcija sa više lokalnih ekstremuma, oni su prirodan izbor u ovom slučaju, kada nije jasno kako tačno izgleda prostor topologija mreža.

Svi metodi koje je autor ove teze pronašao u literaturi i koji kombinuju sisteme Lindenmajera i genetske algoritme, za implementaciju koriste ili graf ili matrice. Predloženi metod koristi samo stringove za reprezentaciju, pa se na taj način postiže efikasnije pronalaženje odgovarajuće topologije. Formalno je opisan pojam lijevog i desnog konteksta, koji se kod svih drugih autora sličnih metoda određivao na osnovu generisanog grafa, a ne na osnovu samog pravila koje generiše graf. Predloženi metod kreira modularne mreže, čime se povećava skalabilnost i smanjuje se prostor koji pretražuje genetski algoritam.

Koncept evolucije tokom učenja, poznat i pod nazivom Boldvinov efekat, pokazao se kao obećavajući mehanizam za unapređivanje kvaliteta generisanih neuronskih mreža. Boldvinov efekat je u ovoj tezi primjenjivan po heuristikama koje su empirijski, kroz simulacije, razvijane tokom istraživanja. Uprkos tome, u nekim slučajevima, takav pristup je značajno poboljšavao generisane mreže, iako je bilo i primjera kada je pokazao negativan uticaj na mrežu. Dalji razvoj ovog koncepta moguće je usmjeriti ka nalaženju formalnih uslova kada i na koji način se efekat može primjeniti u generisanoj mreži.

U cilju verifikacije rezultata istraživanja sprovedenog u ovoj doktorskoj tezi, implementiran je programski paket ***NNGen*** (Neural Net Generator). Modularna struktura ovog paketa omogućava njegovu laku nadogradnju i prilagođavanje novim alatima. Originalno je ovaj paket bio zamišljen samo kao implementacija algoritama i metoda opisanih u četvrtom poglavlju ove teze. Brzi razvoj različitih programskih paketa za kreiranje i obučavanje različitih tipova neuronskih mreža uslovio je promjene u konceptu samog softvera, pa je njegova osnovna funkcionalnost nadograđena opcijom generisanja koda za jedan takav paket – JOONE paket. Moguće nadogradnje paketa ***NNGen*** uključuju:

- dodavanje opcija za generisanje koda za druge pakete kreiranje i obučavanje neuronskih mreža kao što su *FANN*, *Encog*, *Neuroph* i *Matlab Neural Network Toolbox*;
- promjena strukture paketa tako da se umjesto ugrađenog modula ***GenAlg*** koristi neka druga biblioteka za genetske algoritme, npr. *GALib*;
- dodavanje mogućnosti da se koriste druge biblioteke za samoorganizujuće mreže, npr. *SOMPak*, *KNNL*, *SOM-Code* ili *TESI*;
- reimplementacija dijela koda u programskom jeziku C++ i/ili jeziku Java;
- kreiranje grafičkog korisničkog interfejsa;
- paralelizacija modula ***GenAlg*** i ***LSystem***.

Provedene simulacije na četiri skupa podataka pokazuju da je opisani metod konkurentan drugim metodima koji koriste evolucione strategije za generisanje topologije mreže. Mreže generisane paketom ***NNGen***, i kada se obučavaju

standardnim algoritmima obučavanja, pokazuju rezultate koji su uporedivi sa komercijalnim paketima.

Istraživačke aktivnosti u širem smislu, tj. aktivnosti na daljem razvoju odgovarajućih metoda i ciljeva mogu se podijeliti u dvije grupe:

- implementacija i iscrpno testiranje algoritama iz trećeg poglavlja ove teze: algoritma obučavanja samoorganizujuće karte po disjunktnim klasterima susjednih neurona, algoritma fazi-klasifikacije i algoritam za očuvanje topografskih i metričkih svojstava;
- dalji razvoj metoda generisanja topologije neuronske mreže.

Dalji razvoj metoda generisanja topologije neuronske mreže uključuje ili promjene u jednom od tri glavna dijela našeg metoda (obučavanju mreže, genetskom algoritmu i GL-Sistemu) ili kombinovanje dva ili sva tri dijela. Eksperimenti na primjerima WBCD pokazuju da se vrijeme generisanja topologije mreže može značajano skratiti ako se umjesto algoritma "back-propagation" upotrebi neki drugi algoritam. Genetski algoritam može biti zamijenjen nekom drugom evolucionom strategijom ili nekim drugim algoritmom lokalnog traženja u prostoru kodiranih topologija mreža. Umjesto kontekstnog GL-sistema moguća je i upotreba stohastičkih L-sistema, koji se već dugo koriste u računarskoj grafici.

Skoriji istraživački napori na polju razvoja evolucije neuronskih mreža usmjereni su ka ispitivanju mogućnost istovremene evolucije topologije i težinskih koeficijenata mreže, pa čak i evolucije parametara obučavanja mreže. Principi koji se koriste u razvoju takvih mreža isti su kao i principi navedeni u ovoj disertaciji: koristi se indirektno kodiranje mreže u kombinaciji sa postepenim rastom mreže tokom same evolucije mreže i njenog obučavanja.

Literatura

[Aar03] Aarseth, S. J – Gravitational N-body Simulations: Tools and Algorithms. Cambridge University Press. ISBN 0-521-12153-1, 2003.

[Abr02] Abraham, A. – Optimization of Evolutionary Neural Networks Using Hybrid Learning Algorithms. In: Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02), IEEE, vol. 3, pp. 2797–2802, 2002.

[Abr04] Abraham, A. – Meta-Learning Evolutionary Artificial Neural Networks, Neurocomputing Vol. 56c, pp. 1-38, 2004.

[AhoKemKos97] Aho, I.; Kemppainen; H., Koskimies, K. – Searching Neural Network Structures with L-Systems and Genetic Algorithms, University of Tampere, Department of Computer Science, Technical Report A-1997-15, 1997.

[Ala01] Alander, J. T. – An Indexed Bibliography of Genetic Algorithms and Neural Networks. Technical report 94-1-NN. University of Vaasa, Department of Information Technology and Productive Economics 2001

[AngSauPol94] Angeline, P.J., Saunders, P.J., Pollack, J.B. – An Evolutionary Algorithm that Constructs Reccurent Neural Networks. IEEE Transactions on Neural Networks, 5(1): 54-65, 1994.

[BalHan01] Balakrishnan K.; Honavar, V. – Evolving Neuro-controllers and Sensors for Artificial Agents. In: Advances in the Evolutionary Synthesis of Intelligent Agents, MIT, pp. 109–152, 2001

[BarBlo99] Baraldi, A. and Blonda, P. – A Survey of Fuzzy Clustering Algorithms for Pattern Recognition, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics Vol. 29 Issue 6, , pp. 786-801, December 1999

[BauHerVil97] Bauer, H. U., Herrman T., Villman T. – Neural Maps and Topographic Vector Quantization. IEEE Transactions on Signal Processing, vol. XX, no. Y, 1997.

[BelMcInSch90] Belew, R. K., McInnery, J., Schraudolph, N. - Evolving Networks: Using Genetic Algorithms with Connectionists Learning. Technical report CS90-174, Cognitive Computer Science Resaerch Group, University of California at San Diego, 1990.

[BisSveWil98] Bishop C.M.; Svensen M.; Williams C.K.I – The Generative Topographic Mapping. Neural Computation 10 (1), 215-234

[BoeKui93] Boers E. J. W. and Kuiper H. – Biological Metaphors and the Design of Modular Artificial Networks, Proceedings of the International Conference on Artificial Neural Networks 1993, Springer-Verlag, 1993

[BoeKui01] Boers E. J. W. and Sprinkhuizen-Kuyper I. G. – Combined Biological Metaphors. In Advances in the Evolutionary Synthesis of Intelligent Agents By Mukesh Patel, Vasant Honavar, Karthik Balakrishnan (Eds.), pp. 153-183, MIT Press, 2001.

[BoeKuiHap93] Boers E.J.W.; Kuiper H.; Happel B.L.M.; Sprinkhulzen-Kuyper I.G. – Designing Modular Artificial Networks, In Wijshoff H.F.G. (ed.): Proceedings of the Conference on Computing Science in The Netherlands (CSN '93), p. 89-96, 1993

[BonSubEkl06] Bonissone PP, Subbu R, Eklund N, Kiehhl TR – Evolutionary Algorithms + Domain Knowledge = Real-World Evolutionary Computation. IEEE Transactions on Evolutionary Computation 10(3):256–280, 2006.

[Bor95] Borst M. V. – Local Optimization in Evolutionary Generated Neural Networks. Master's Thesis, Department od Computer Science, Leiden University, Holandija, 1995.

[Bra95] Branke J. – Evolutionary Algorithms for Neural Network Design and Training – A Review. In Proceedings of 1st NWGA, Vaasa, Finland, 1995.

[BraRag97] Braun, H., Ragg, T. – Evolutionary Optimization of Neural Networks for Reinforcment Learning Algorithms. In Proceedings of the International Conference on Artificial Intelligence and Genetic Algorithms, Norwich, UK, 1997.

[Bur13] Burt, T. – Interactive Evolution by Duplication and Diversification of L-systems. M.Sc. thesis, University of Calgary, July 2013.

[CamRoi04] Campos, L., Roisenberg, M. – A Biologically Inspired Methodology for Neural Networks Design, In Proceeding of 2004 IEEE Conference on Cybernetics and Intelligent Systems, Volume 1

[CamRoiOli11] Campos, L., Roisenberg, M., and Oliveira, R. – Automatic design of Neural Networks with L-Systems and genetic algorithms - A biologically inspired methodology, IEEE Proceedings of IJCNN 2011, pages 1199-1206.

[CanCha07] Canales, F., Chacon, M. – Modification of the Growing Neural Gas Algorithm for Cluster Analysis. In Proc. Progress in Pattern Recognition, Image Analysis and Applications: 12th Iberoamerican Congress on Pattern

Recognition, CIARP 2007 Springer. pp. 684–693. doi:10.1007/978-3-540-76725-1_71. ISBN 978-3-540-76724-4.

[CanNolPar93] Cangelosi, A., Nolfi, S., Parisi, D. – Cell division and migration in a 'genotype' for neural networks. Evolution of Artificial Neural Networks. Networks: Computations in Neural Systems 5: 497-515, 1993.

[CanKam05] Cantu-Paz, E. and Kamath, C. – An Empirical Comparison of Combinations of Evolutionary Algorithms And Neural Networks For Classification Problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 35(5):915–927, 2005.

[CarGro03] Carpenter, G.A. & Grossberg, S. – Adaptive Resonance Theory – In The Handbook of Brain Theory and Neural Networks, Second Edition (pp. 87-90). Cambridge, MA: MIT Press, 2003.

[ChaDam98] Chanon, A. D., Damper, R. I. – Evolving Novel Behavior via Natural Selection. In ALife VI: Sixth International Conference on Artificial Life, editors: Adami, C., Belew, R. K., Kitano, H., Taylor, C. E. , pages 384-388, LA, CA, Mit Press, Cambridge, MA, 1998.

[Chv02] Chval, J. – Evolving Artificial Neural Networks by Means of Evolutionary Algorithms with L-Systems Based Encoding. Research Report, Technical University in Kosice, Faculty of Electrical Engineering, 2002.

[CluStaPen11] Clune J, Stanley K.O,, Pennock R.T., Ofria C. – On the Performance of Indirect Encoding Across The Continuum of Regularity. IEEE Transactions on Evolutionary Computation. 15(3): 346-367, 2011.

[DuZhaBao06] Du L., Zhang J., Bao Z. – Multicategory Classification Based on Hypecube SOM scheme, In Advances in Natural Computation, Lecture Notes in Computer Science Volume 4221, 2006, pp 107-110, 2006.

[DonLiSan13] Donate, J.P.; Li, X.; Sanchez, G. G.; Miguel, A.S – Time Series Forecasting by Evolving Artificial Neural Networks with Genetic Algorithms, Differential Evolution And Estimation Of Distribution Algorithm, Neural Computation & Applications 22:11–20, DOI 10.1007/s00521-011-0741-0, 2013

[Du10] Du, K. L. – Clustering: A Neural Network Approach, Neural Networks 23, pp. 89-107, 2010.

[EstPriZeg13] Estévez, P.A; Príncipe, J. C.; Zegers, P. (eds.) – Advances in Self-Organizing Maps: 9th International Workshop, WSOM 2012. Springer, ISBN 978-3-642-35229-4, 2013.

[EzhShu98] Ежов А. А., Шумский С. А. – Нейрокомпютинг и его

применения в экономике и бизнесе. М.: МИФИ, 1998. http://www.com2com.ru/dav/framettl.htm

[FerCalPar01] Ferdinando, A.D., Calabretta, R., Parisi, D. – Evolving Modular Architectures for Neural Networks. In: French R, Sougn´e J (eds.) Proceedings Sixth Neural Computation and Psychology Workshop Evolution, Learning, and Development, 2001.

[FloDurMat08] Floreano, D; Dürr, P., and Mattiussi, C. – Neuroevolution: From Architectures to Learning. Evolutionary Intelligence, 1(1):47–62, March 2008. doi: 10.1007/s12065-007-0002-4.

[FogFogPor90] Fogel, D.B., Fogel L.J., Porto, V. W. – Evolving Neural Networks, Biological Cybernetics, 63(6): 487-493, 1990.

[Fred97] Frederikson K. – Genetic Algorithms and Generative Encoding of Neural Networks for Some Benchmark Classification Problems. In Preceedings of 3rd NWGA, Vaasa, Finska, 1997.

[Fri93] Fritzke B. – Kohonen Feature Maps and Growing Cell Structures - a Performance Comparison, in Advances in Neural Information Processing Systems 5, Eds. Giles, C. L., Hanson S.J. and Cowan J. D., Morgan Kaufman Publishers, 1993.

[Fri97-1] Fritzke B. – Incremental Neuro-Fuzzy systems, in Applications of Soft Computing, Invited paper, SPIE International Symposium on Optical Science, Engineering and Instrumentation, San Diego, 1997.

[Fri97-2] Fritzke B. – The LBG-U method for vector quantization – an improvement over LBG inspired from neural networks, Neural Processing Letters, Vol. 5, No. 1, 1997.

[FriMor96] Friedrich C. M. and Moraga C. – An Evolutionary Method to Finding Good Building Blocks for Architectures of Artificial Neural Networks. In Proc. of 6th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '96), pp. 951-956, Spain, 1996

[FriMor97] Friedrich C.J. and Moraga C. – Using Genetic Algorithms to Find Modular Structures and Activation Functions for Architectures of Artificial Neural Networks. In Computational Intelligence, Theory and Applications: Proceedings of the 5th Fuzzy Days: (LNCS 1226), pp. 150-161, Dortmund, Germany, 1997

[Gar93] Garis d. H. – Circuits of Production Rule: Gennets - The Genetic Programming of Artificial Nervous Systems. In International Joint Conference on Neural Networks and Genetic Algorithms, pages 699--705, Innsbruck, 1993.

[Gro98] Gronroos, M. – Evolutionary Design of Neural Networks. MSc thesis, University of Turku, Department of

Mathematical Sciences, 1998.

[Gruau94] Gruau F. – Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. PhD thesis, Ecole Normale Superieure de Lyon, 1994. ftp://lip.ens-lyon.fr/pub/Rapports/PhD/PhD94-01-E.ps.Z.

[Gruau95] Gruau, F. – Automatic Definition of Modular Neural Networks, Adaptive Behaviour, 3(2): 151-183, 1995.

[Gruau96] Gruau, F., Whitley, D., Pyeatt, L. – A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Network. In Koza et al. Proceedings of the GP96, Stanford, USA, 1996.

[GruWhi93] Gruau F. and Whitley D. – Adding Learning to the Celular Development of Neural Networks: Evolution And The Baldwin Effect. Evolutionary Computation, 3(1):213--233, 1993.

[GruReg05] Grushin, A., Reggia, J.A. – Evolving Processing Speed Asymmetries and Hemispheric Interactions in a Neural Network Model. Neurocomputing 65:47–53, 2005.

[Han92] Hancock P. J . B. – Genetic algorithms and permutation problem: a comparison of recombination operators for neural net structure specification. In International Workshop on Combinations of Genetic Algorithms and Neural Networks, pages 108--122, Baltimore, 1992.

[HapMur94] Happel B. L. M. and Murre J. M. J. – Design and Evolution of Modular Neural Network Architectures. Neural Networks, 7(6/7):985--1004, 1994.

[HarSma91] Harp, S. A. and Smad, T. – Genetic synthesis of neural network architecture. In Handbook of Genetic Algorithms, pages 202--221, 1991.

[Hyd08] Hyde, D. – Introduction to Genetic Principles, McGraw-Hill, 2008.

[Hay04] Haykin, S. – Neural Networks – A Comprehensive Foundation, 2nd Ediition. MacMillan College Publishing Company, 2004.

[Hoo91] Hoohemstrate, R.J.W. van – A Neural Network for Genetic Facie Recognition, Technical report, Leiden University, 1991.

[Huss98] Hussain, T. J. – Network Generating Attribute Grammar Encoding. Phd Thesis, Queen's University, Kingston, Ontario , Kanada, 1998.

[Jac99] Jacob, C. – Genetic L-System Programming. Technical Report, University of Erlangen-Nurnberg, Njemačka, 1999

[JacJor95] Jacob, R. A. and Jordan, M. I. – Modular and Hierarchical Learning Systems. In M.Arbib (Ed.), The Handbook of Brain

Theory and Neural Networks. MIT Press, Cambridge, Massachusets, 1995.

[JacReh93] Jacob C. and Rehder J. – Evolution of Neural Network Architectures by a Hierarchical Grammar-based Genetic System. In International Joint Conference on Neural Networks and Genetic Algorithms, pages 72--79, Innsbruck, 1993.

[Joo07] The JOONE Complete Guide, 2007, dostupno na http://www.codertodeveloper.com/docs/

[JunReg08] Jung, J.Y. and Reggia, J.A. – The Automated Design of Artificial Neural Networks Using Evolutionary Computation, Studies in Computational Intelligence (SCI) 92, 19–41 (2008)

[KaiKan07] Kaiser, C.; Kanevski, M. – Classification and visualization of high-dimensional socio-economic data using self-organizing maps. Spatial Econometrics Conference 2007, University of Cambridge, Cambridge, UK, 11-14 July 2007.

[Kas97] Kaski, S. – Data Exploration Using Self-Organizing Maps. PhD thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory of Computer and Informtion Scineces, 1997.

[KasLag96] Kaski, S., Lagus, K. – Comparing Self-Organizing Maps. In C. von de Malsburg, W. von Seelen, J. C. Vorbrugen, B. Sendhoff (Eds.) Proceedings of ICANN96, International Conference on Artificial Neural Networks, Lecture Notes in Computer Science vol. 1112, pp. 809-814, Springer, Berlin, 1996.

[Kitano90] Kitano H. – Designing Neural Networks Using Genetic Algorithms with Graph Generation Systems. Complex Systems, 4:461--476, 1990.

[KodMey95] Kodjabachian, J., Meyer J. – Evolution and Evelopment of Control Architectures in Animates. Robotics and Autonomus Systems 16:2-4, pp. 161-182, 1995.

[Koe94] Koehn P. – Combining Genetic Algorithms and Neural Networks: The encoding problem. Master's thesis, University of Tennessee, Knoxville, 1994.

[Koh01] Kohonen, T. – Self-Organizing Maps, 3rd Edition. Springer, ISBN 6-540-67921-9, 2001.

[KozRic91] Koza J. R. and Rice J. P. – Genetic Generation of Both the Weight and Architecture for a Neural Network. In International Joint Conference on Neural Networks, pages II 397–404. IEEE, 1991.

[LehWea94] Lehar, S. and Weaver, J. – A Developmental Approach to Neural Network Design. In International Conference on Neural Networks, pages II 97--104. IEEE, 1994.

[Loc12] Lockett A. J. – General-Purpose Optimization Through Information-Maximization, PhD Thesis, Department of Computer Sciences, The University of Texas at Austin, 2012. Tech Report AI12-11

[LocMii13] Lockett A. J., and Miikkulainen, R. – Neuroannealing: Martingale-Driven Optimization for Neural Networks, In Proceedings of the 2013 Genetic and Evolutionary Computation Conference (GECCO-2013), 2013. ACM Press.

[LukSpe96] Luke S. and Spector L. – Evolving Graphs and Networks with Edge Encoding: Preliminary Report. In Late-breaking Papers of the Genetic Programming 96 (GP96), Stanford, 1996.

[Mand93] Mandischer M. – Representation and Evolution of Neural Networks. In International Joint Conference on Neural Networks and Genetic Algorithms, pages 643--649. Innsbruck, 1993.

[Mani94] Maniezzo V. – Genetic Evolution of the Topology and Weight Distribution of Neural Networks. IEEE Transactions of Neural Networks, 5(1):39-53, 1994.

[Mar92] Marti, L. – Genetically Generated Neural Networks II: Searching for the Optimal Representation. In IEEE Joint Conference on Neural Networks, pages II 221--226, 1992.

[MarSch94] Martinetz T. and Schulten K. – Topology Representing Networks. Neural networks, Vol. 7, No. 3, pp. 507-522, 1994

[MarQuaAnd11] Marcano-Cedeño, A.; Quintanilla-Domínguez, J.; Andina, D. – WBCD Breast Cancer Database Classification Applying Artificial Metaplasticity Neural Network, Expert Systems with Applications 38, pp. 9573–9579, 2013

[MaqKhaAbr07] Maqsood I, Khan MR, Abraham A – An Ensemble of Neural Networks for Weather Forecasting. Neural Computations & Applications 13(2):112–122, 2007

[Mic99] Michalewicz, Z. – Genetic Algorithms + Data Structures = Evolutionary Programs. Springer-Verlag, Berlin, 1999.

[MilTodHed89] Miller G. F.; Todd P. M.; Hedge S. U. – Designing Neural Networks Using Genetic Algorithms. In Third International Conference on Genetic Algorithms, pages 379-384. Morgan Kaufmann, 1989.

[Mitc98] Mitchell, M. – Introduction to Genetic Algorithms, MIT Press, 1998.

[MitPal94] Mitra S. and Pal S.K. – Self-Organizing Neural Network as a Fuzzy Classifier, IEEE Transactions on Systems, Man and Cybernetics, vol. 24, no. 3, march 1994, pp. 385-399

[MonDav89] Montana D. and Davis L. – Training Feedforward Neural Networks Using Genetic Algorithms. In 11th International Joint Conference on Artificial Intelligence, pages 762--767. Morgan Kaufmann, 1989.

[MusSch94] Musial M. and Scheffer T. – A Term-based Code for Artificial Neural Networks. In J. Hopf (editor): Genetic Algorithms within the Framework of Neural Computation: Proceedings of the Ki94-Workshop, Max-Planck Institut, Saarbrucken, 1994.

[NolPar94] Nolfi, S., Parisi, D. – Genotypes for Neural Networks. In M. A. Arbib et al. (editors). The Handbook of Brain Theory and neural Networks. MIT Press, 1994.

[Oya95] Oyala T. – Neuro-fuzzy Systems in Control, MSc. Thesis, Tampere University of Technology, Department of Electrical Engineering, 1995

[PalBez92] Pal S. K. and Bezdek J. C. – Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data. NY IEEE Press, 1992

[PanMac95] Pandya, A. S., Macy. R.B – Pattern Recognition with Neural Networks in C++, ISBN:0849394627, CRC Press, 1995.

[Pit95] Pit L. J. – Parallel Genetic Algorithms. Master's Thesis, Department of Computer Science, Leiden Univesity, Netherlands, 1995.

[PruLin91] Prusinkiewicz, P.; Lindenmayer, A. – The Algorithmic Beauty of Plants. Springer-Verlag, 1991.

[PujPol98] Pujol J. C. F. and Polli R. – Evolving the Topology and the Weights of Neural Networks Using Dual Representation. Special Issue on Evolutionary Learning of the Applied Intelligence Journal 8(1):73-84, Jan. 1998

[RadHin13] Amr, R. and Hindawi, S.K. – Applying Artificial Neural Network Hadron - Hadron Collisions at LHC, Artificial Neural Networks - Architectures and Applications, Prof. Kenji Suzuki (Ed.), ISBN: 978-953-51-0935-8, InTech, 2013, DOI: 10.5772/51273.

[RagGut96] Ragg, T., Gutjahr, J. – Neural network optimization through searching guided by the stochastic methods. In Proceedings of the 6th European Congress of Intelligent Techniques and Soft Computing, Volume 1, pp. 245-249, Achen, Germany, 1998.

[RitKoh89] Ritter H. and Kohonen T. – Self-Organizing Semantic Maps. Biological Cybernetics, vol. 61, pp. 241-254, 1989

[RocCorNev07] Rocha, M., Cortez, P., Neves, J., – Evolution of Neural Networks for Classification and Regression, Neurocomputing, Vol. 70, Issues 16-18, pp. 2809-2816,

2007.

[SchJooWer93] Schiffmann W.; Joost M.; Werner R. – Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons. In International Joint Conference on Neural Networks and Genetic Algorithms, pages 675-682, Innsbruck, 1993.

[SchHamBie09] Schneider, P.; Hammer, B., and Biehl, M. – Adaptive Relevance Matrices in Learning Vector Quantization. Neural Computation 21: 3532–3561, 2009.

[SmaGro95] Smagt v.d. P. and Krosse B. – Using Many-Particle Decomposition to get a Parallel Self-Organizing Map, Joint National Conference on Computer Science in The Netherlands, 1995.

[StaMii02] Stanley K. O. and Miikkulainen, R. – Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation,10(2):99–127,2002

[StaMii03] Stanley K. O. and Miikkulainen, R. – A Taxonomy for Artificial Embryogeny, Artificial Life, 9(2):93-130, 2003.

[Sta04] Stanley K. – Efficient Evolution of Neural Networks through Complexification, PhD. Thesis, University of Texas Austin, 2004.

[Suk02a] Шукович Горан – Применение генетических алгоритмов и систем генерирующих графов для создания модулярных нейросетей, Программирование, ISSN 0132-3474, 2002, №- 1, с. 13-20.

[Suk02b] Sukovic, G. – Application of Genetic Algorithms and Systems of Generating Graphs for Creation of Modular Neural Networks. Programming and Computer Software, Vol. 28, No. 1, 2002, pp. 9–14.

[Suk05] Šuković G., – Generating Modular Network Architectures – A String-Based Approach, Proceedings of the Workshop "Contemporary Mathematics, Physics and Biology - 25 Anniversary of the Faculty of Natural Sciences and Mathematics, University of Montenegro", Podgorica, Univerzitet Crne Gore, 2005, pp. 199-207, 2005.

[SukMos13] Šuković. G, Mosurović, M. – Generating Modular Neural Networks Using Lindenmayer Systems And Genetic Algorithms, Zbornik radova „Informacione Tehnologije - sadašnjost i budućnost 2013", pp. 216-219.

[Tal99] Talko, B. – A Rule-Based Approach for Constructing Neural Networks Using Genetic Programming. MSc thesis, University of Melbourne. 1998.

[Thi98] Thierens D. – Non-Redundant Genetic Coding of Neural Networks. University of Utrecht, Technical Report 1998-46. 1998.

[TorrNak99] Torresen, J., Nakashima, H., Tomita, S., Landsverk, O. – General Mapping of Feedforward Neural Networks onto an MIMD Computers. Technical report, Department of Computer Systems and Telematics, Norwegian Institute of Technology, 1999.

[TouHusIge03] Toussaint, M.; Husken, M., Igel, C. – Task Dependent Evolution of Modularity in Neural Networks. Connection Science, 14(3):219–229, 2003.

[TsiGavGla08] Tsoulos I., Gavrilis D., Glavas E. – Neural Network Construction and Training Using Grammatical Evolution. Sci Direct Neurocomput J 72(1–3):269–277, 2008.

[XuWun05] Xu, R., Wunsch, D. – Survey of Clustering Algorithms, IEEE Transactions on Neural Networks, Vol. 16, No. 3, May 2005.

[Yao99] Yao, X. – Evolving Artificial Neural Networks, Proceedings of the IEEE, Vol. 87, No. 9, Sep. 1999.

[Yao10] Yao, X. – Evolving, Training and Designing Neural Network Ensembles, In 14th International Conference on Intelligent Engineering Systems (INES), 2010

[VoiBorSan94] Voigt, H. M., Born, J., Santibanez.Koref, I. – Designing Neural Networks by Adaptively Building Blocks in Casacdes. In Y. Davidor et. al. Parallel Problem Solving in Nature, Volme 866 of Lecture Notes in Computer Science, Jerusalem, Izrael. Springer-Verlag, Berlin, 1994.

[Wal98] Walter J. – PSOM Networks: Learning with Few Examples. In Proc. Int. Conf. on Robotics and Automation (ICRA), IEEE, 1998

[Whi93] White D.W. – GANNet: A Genetic Algorithm for Searching Toplogy and Weight Spaces in Neural Network Design. PhD thesis, University of Maryland, 1993.

[Whi89] Whitley. D. – The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best. In: Proceedings of the 3rd International Conference on Genetic Algorithms and their applications (ICGA), 116-121, J.D. Schaffer (Ed.), Morgan Kaufmann, San Mateo CA, 1989.

[WhiStaBog90] Whitley D., Starkweather T., Bogart C. – Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. Parallel Computing, 14:347--361, 1990.

Dodatak A – generisani C kod za problem ekskluzivno-ili

```
// mreza za XOR ili eksluzivno-ILI
void solve(double alpha, double beta, unsigned long maxIter,
   double eps)
{
  /*
        alpha - stopa obucavanja
        beta - momentum
        maxIter - maksimalan broj iteracija za obučavanje
        eps - greska

        nacin pozivanja: solve(0.4, 0.9,1000, 0.2);
  */
  unsigned modules[] = { 2,2,1,0 };
  ConSpec con[] = { {0,1},{0,2},{1,2},{0,0} };
  double error, input[2], output[1], calc[1];
  int i,n;
  Network net(2, 1, modules, con);
  net.setAlphaBeta(alpha, beta);
  net.reset();
  for(n=0; n < maxIter; n++)
  {
        error=0.0;
        for(i=0; i < 4; i++)
        {
              input[0]=(double)(i/2);
              input[1]=(double)(i%2);
              output[0]= (i==1 || i==2) ? 0.9 : 0.1;
              net.train(input, output, calc);
              error += output[0]-calc[0];
        }
        if(error < eps) // stop, ako je greska dovoljno mala
              break;
  }
  net.save("xor.net");
}
```

Dodatak B – generisani Java kod za problem ekskluzivno-ili

```
import java.util.concurrent.CountDownLatch;
import org.joone.engine.FullSynapse;
import org.joone.engine.LinearLayer;
import org.joone.engine.Monitor;
import org.joone.engine.NeuralNetEvent;
import org.joone.engine.NeuralNetListener;
import org.joone.engine.SigmoidLayer;
import org.joone.engine.learning.TeachingSynapse;
import org.joone.io.MemoryInputSynapse;
import org.joone.net.NeuralNet;

public class JooneXOR implements NeuralNetListener {
     private Monitor monitor;
     private NeuralNet nnet;
     private CountDownLatch latch = new CountDownLatch(1);
     public static double XOR_INPUT[][] = { { 0.0, 0.0 }, { 1.0,
0.0 },
          { 0.0, 1.0 }, { 1.0, 1.0 } };
     private int epoch;

public static double XOR_IDEAL[][] = { { 0.0 }, { 1.0 }, { 1.0 },
{ 0.0 } };


     public JooneXOR()
     {
          LinearLayer      layer1 = new LinearLayer();
        SigmoidLayer layer2 = new SigmoidLayer();
        SigmoidLayer layer3 = new SigmoidLayer();

        layer1.setLayerName("layer1");
        layer2.setLayerName("layer2");
        layer3.setLayerName("layer3");

        layer1.setRows(2);
        layer2.setRows(3);
        layer3.setRows(1);

        FullSynapse synapse_12 = new FullSynapse();
        FullSynapse synapse_23 = new FullSynapse();

        synapse_12.setName("synapse_12");
        synapse_23.setName("synapse_23");

        layer1.addOutputSynapse(synapse_12);
```

```
    layer2.addInputSynapse(synapse_23);

    layer2.addOutputSynapse(synapse_12);
    layer3.addInputSynapse(synapse_23);

    MemoryInputSynapse  inputStream = new MemoryInputSynapse();

    inputStream.setInputArray(XOR_INPUT);
    inputStream.setAdvancedColumnSelector("1,2");

    layer1.addInputSynapse(inputStream);

    TeachingSynapse trainer = new TeachingSynapse();
    MemoryInputSynapse samples = new MemoryInputSynapse();

    samples.setInputArray(XOR_IDEAL);
    samples.setAdvancedColumnSelector("1");
    trainer.setDesired(samples);

    layer3.addOutputSynapse(trainer);

    this.nnet = new NeuralNet();

    nnet.addLayer(layer1, NeuralNet.INPUT_LAYER);
    nnet.addLayer(layer2, NeuralNet.HIDDEN_LAYER);
    nnet.addLayer(layer3, NeuralNet.OUTPUT_LAYER);
    this.monitor = nnet.getMonitor();
    monitor.setTrainingPatterns(4); // broj redova u ulazu
    monitor.setTotCicles(5000); // iteracije
    monitor.setLearningRate(0.7); // stopa obucavanja
    monitor.setMomentum(0.3); // momentum
    monitor.setLearning(true); // obucavamo mrezu
    monitor.setSingleThreadMode(true);
    monitor.addNeuralNetListener(this);
}

public void run() {

      try {
            nnet.go(); // async mod
            this.latch.await();

      } catch (InterruptedException e) {
            e.printStackTrace();
      }
}

@Override
public void cicleTerminated(NeuralNetEvent arg0) {

}

@Override
public void errorChanged(NeuralNetEvent e) {
      System.out.println("Epoch: " + epoch +
            " Error:" + this.monitor.getGlobalError() );
```

```
            epoch++;

        }

        @Override
        public void netStarted(NeuralNetEvent arg0) {

        }

        @Override
        public void netStopped(NeuralNetEvent arg0) {
            this.latch.countDown();

        }

        @Override
        public void netStoppedError(NeuralNetEvent arg0, String arg1) {


        }
        public static void main(String[] args)
        {
            JooneXOR xor = new JooneXOR();
            xor.run();
        }
}
```

Dodatak C – klase modula BackProp

```
class Module     // implementira jedan modul
{
  unsigned  id,                 // id  modula
            nodes,              // velicina modula
            totalInput;         // ukupna broj ulaza
  double *activation,           // lista izracunatih aktivacija
         *delta;                // lista  izracunatih delti
  conList *in,                  // lista ulaznih konekcija
          *out;                 // lista izlaznih konekcija

  public:
     Module& addNodes(int numNodes, int typeNodes);
     /*    Dodaje cvorove modulima.
           numNodes - broj cvorova
           typeNodes - nacin inicijlazacije konekcija
     */

  double getInstabilityScore(int select, int where);
     /*    Metod vraca nestabilnost modula.
           select - je kriterijum selekcije
           where -  mjesto gdje da trazimo konekcije:
           ulaz ili izlaz
     */

  void    shock(float);  // mijenja tezine ulaznih konekcija

  int   addIn(class connection*);  // dodaje ulaznu konekciju
  int   addOut(class connection*);  // dodaje izlaznu konekciju
  void  delIn(class connection*);  // brise ulaznu konekciju
  void  delOut(class connection*); // brise izlaznu konekciju
  void  incModId();  // inkrementira id  modula
};

class Connection  // implementira konekciju izmedju modula
{
     Module  *from,  *to;      // pocetni i krajnji modul
     unsigned     fs, ts;      // velicine from i to modula
     unsigned     fromId, toId; //  id-ovi from i to modula
     double       **weight, **moment;
     changeStruc  **change;     // promjena tezina
     double       alpha,  beta; // stopa obucavanja, momentum

public:
     void reset();         // nove slucajne tezine
     void small_reset();  //  slucajne tezine iz segmenta [-0.1,0.1]
     void shock(float);  // mijenja tezinu konekcije
     void incToId();     // inkrementira to id
```

```
    void incFromId();   // inkrementira from id
    int  addIn(int fromMod ,int toMod);
    /*
       Dodaje ulaznu konekciju listi.
       fromMod - id from modula
       toMod - id to modula
    */

    int  addOut(int,int);
    /* kao prethodni metod, samo za izlazne konekcije */
};

class Network // Implementacija back-propagation mreze
{
    unsigned     inputSize, outputSize,
                 totalModules,
                 inputModules, outputModules,
                 totalConnections;
    Connection** nconnection;
    Module** nmodule;

    public:

      Network(unsigned in, unsigned out, unsigned* modSize,
        conSpec* conn);
      /* Konstruktor.
      in - velicina ulaza
      out - velicina izlaza
      modSize - velicine modula
      conn - konekcije
      */

      Network(char nType, char* fileName); // MATRIX ili NET
      ~Network();

      int save(char* fileName); // snimanje mreze u fajl
      void reset(); // izbor 'optimalnih' slucajnih tezina
      void train(double* in, double* out, double* outNet);
      /*
         Obucavanje mreze.
         in - ulaz
         out - izlaz
         outnet - zeljeni izlaz
      */

      void calc(double* in, double* out); // ulaz, izlaz
      Module* getModule(unsigned nrModule);
      Connection* getConnection(unsigned from, unsigned to);
      /*
           from - from ID
           to - to ID
      */
      void setAlphaBeta(double alpha, double beta);
      // parametri mreze: stopa obucavanja i momentum

      unsigned getInputSize();
```

```
      unsigned getOutputSize();

      int   SelectModule(int crit, int where );
      /* Metod bira modul cije performanse nisu dobre.
        select - kriterijum selekcije
        where -  mjesto gdje da trazimo konekcije: ulaz ili izlaz
      */

      Module* addModule(int numNodes, int pos);
      /*  Metod dodaje modul na izlaznu stranu.
            numNodes - broj cvorova u novom modulu
            pos -  id izabranog modula
      */

      Module* addModule2(int numNodes, int pos);
      // isto kao prethodni metod, samo za ulaznu stranu

      int   shiftModule(int pos);
      /* Vraca id novog modula poslije reorganizovanja modula.
       Novi modul se dodaje na stranu izlaza
       pos - id izabranog modula
      */

      int   shiftModule2(int pos);
      // isto kao prethodni metod, ali na stranu ulaza

};

Module* Network::addModule(int nbr, int nr)
/*
* Novi modul se dodaje na izlaznoj strani izabranog modula.
* nbr - broj cvorova koji se dodaju modulu.
* nr - ID izabranog modula.
* Metod vraca pokazivac na novi modul.
*/
{
      unsigned newCon;
      unsigned newConIn,  newConOut;   unsigned i,j;
      int newMod;
      nmodule=(module**)realloc(nmodule,
            sizeof(Module*)*(totalModules+1));
      if(nmodule==NULL)
            exit(1);
      newMod = shiftModule(nr);

      totalModules = totalModules + 1;
      nmodule[newMod] = new Module(nbr, newMod);
      newConOut = 0;
      for (i=0; i<totalModules;i++)
      {
            if (getConnection(nr,i) != NULL)
            {
                  newConOut=newConOut+1;
                  nconnection =(Connection**)realloc(nconnection,
                        sizeof(Connection*) *
                        (totalConnections + newConOut));
```

```
                if (nconnection==NULL) exit(1);
                nconnection[totalConnections  +  newConOut - 1] =
                      new Connection(nmodule[newMod], nmodule[i]);
                nconnection[totalConnections+newConOut-1]->reset();
          }
     }
     totalConnections = totalConnections + newConOut;
     newConIn = 1;
     nconnection = (Connection**)realloc(nconnection,
          sizeof(Connection*)* (totalConnections + newConIn));
     if (nconnection==NULL) exit;

     for (i=totalConnections; i<(totalConnections+newConIn); i++)
     {
          nconnection[i] = new Connection( nmodule[nr],
               nmodule[newMod]);
          nconnection[i]->reset();
     }
     totalConnections = totalConnections + newConIn;
     return nmodule[newMod];
} // end method addModule

int Network::shiftModule(int nr)
/*
*     Moduli za klasu Network moraju biti u odredjenom redosledu.
*     nr - id izabranog module
*     Metod vraca ID novog modula
*/
{
     unsigned i,j;
     int lownr;
     int newMod;

     lownr = totalModules;
     for(i=0; i<totalModules; i++)
     {
          if(getConnection(nr,i) != NULL)
          if(lownr < i) lownr = i;
     }
     for(i = (totalModules-1); i>=lownr; i--)
     {
          nmodule[i]->incModId();
          nmodule[i+1] = nmodule[i];
          for(j=0;j<totalConnections; j++)
          {
               if((nconnection[j]->getToId()) == i)
                    nconnection[j]->incToId();
               if((nconnection[j]->getFromId()) == i)
                    nconnection[j]->incFromId();
          }
     }
     newMod = lownr;
     return newMod;
}
```

Dodatak D – modul SOM

```
/* Kohonen's SOM */

#include <cstdlib>
#include <cstdio>
#include <cmath>

#define  MAXPATS  100
#define  MAXNEURONSIN 10
#define  MAXNEURONSX  15
#define  MAXNEURONSY  15

#define  MAXEPOCHS 2000
#define  ETAMIN .005

unsigned int Random(int N) {
    unsigned int j;
    j=(N*rand())/RAND_MAX;
    if(j>=N) j=N;
    return j;
}

class Pattern
{
    friend class SOFM;
    private:
         double P[MAXPATS][MAXNEURONSIN];
         int NumPatterns;
         int Shuffle[MAXPATS];
         int SizeVector;
    public:
         Pattern();
         int GetPatterns(char*); // ucitavanje iz fajla
         int GetRandPats(int,int);
         /* slucajni uzorak: arg1= broj uzoraka , arg2=dimenzija*/
         double Query(int,int); // vraca P[arg1][arg2]
         double QueryR(int,int); // vraca P[Shuffle[arg1]][arg2]
         void ReShuffle(int N);
};


Pattern::Pattern() // konstruktor
{
    int i;
    for (i=0; i<MAXPATS; i++)
      Shuffle[i]=i;
}

int Pattern::GetPatterns(char*fname) // ucitavanje iz fajla
{
    FILE *fp;
    int i,j;
    double x;
```

```
    fp=fopen(fname,"r");
    if (fp==NULL) return 0;
    fscanf(fp,"%d",&NumPatterns);
    fscanf(fp,"%d",&SizeVector);
    for (i=0; i<NumPatterns; i++)
    {
        for (j=0; j<SizeVector; j++)
        {
            fscanf (fp,"%lf",&x);
            P[i][j]=x;
        }
    }
    fclose(fp);
    return 1;
}

int Pattern::GetRandPats(int n1, int n2)
{
    int i,j;
    double x;
    NumPatterns=n1;
    SizeVector=n2;
    for (i=0; i<NumPatterns; i++)
    {
        for (j=0; j<SizeVector; j++)
        {
            x=(double)rand()/RAND_MAX;
            P[i][j]=x;
        }
    }
    return 1;
}

void Pattern::ReShuffle(int N)
{
    int i, a1, a2, tmp;
    for (i=0; i<N ;i++)
    {
          a1=Random(NumPatterns);
          a2=Random(NumPatterns);
          tmp=Shuffle[a1];
          Shuffle[a1]=Shuffle[a2];
          Shuffle[a2]=tmp;
    }
}

double Pattern::Query(int pat,int j)
{
    return P[pat][j];
}

double Pattern::QueryR(int pat,int j)
{
    return P[Shuffle[pat]][j];
}

struct iPair
{
    int x,y;
};
```

```
/* SOFM class definition */

class SOFM
{
    private:
    double W[MAXNEURONSIN][MAXNEURONSX][MAXNEURONSY];
    // matrica teyinskih koeficijenata

    double Yout[MAXNEURONSX][MAXNEURONSY];
    // izlazni lsoj neurona

     double Yin[MAXNEURONSIN]; // ulazni sloj neurona
     int   Lattice; // kvadratna ili trougaona struktura
     int   YinSize; // dimenzija ulaznog sloja
     iPair YoutSize; // dimenzije ulaznog sloja
     int   R; // poluprecnik susjedstva radius
     int   MaxEpoch;
     int   epoch;
     double eta; // parametar eta
     double delta_eta; // prirastaj parametra eta
         double Erosion;  // smanjivanje susjedstva
 int StochFlg; // ako je jednak 1, onda random
      Pattern *Patt;

      int  LoadInLayer(int);    //pattern->input sloj
      double EucNorm(int, int); // Euklidska norma
      iPair FindWinner(); // koordinate pobjednika
      void  Train(iPair);
      void  AdaptParms();
    public:
      SOFM();
      void SetPattern(Pattern*);
      void SetParms(int, int, double);
      void PrintWeights();
      void PrintWinner();
      void RunTrn();
      void Run();
};


SOFM::SOFM()
{
    StochFlg=1;
    Erosion=0;
}

void SOFM::SetPattern(Pattern *p)
{
    Patt=p;
    YinSize=p->SizeVector;
}

void SOFM::SetParms(int X, int Y, double LR)
{
    int ix, iy, k;
    YoutSize.x = X;
    YoutSize.y = Y;
    R = (X+Y)/4;
    eta = LR;
```

```
    delta_eta = 0.005;
    for (ix = 0; ix < X; ix++)
    {
        for (iy=0; iy<Y; iy++)
        {
            for (k=0; k<YinSize; k++)
            {
                W[k][ix][iy]=(double)rand()/(10.0*(double) RAND_MAX);
            }
        }
    }
}

int SOFM::LoadInLayer(int P)
{
    int i;
    for (i=0; i<YinSize; i++)
    {
        if (StochFlg)
        {
            Yin[i]=Patt->QueryR(P,i);
        }
        else
        {
            Yin[i]=Patt->Query(P,i);
        }
    }
    return 1;
}

void SOFM::AdaptParms()
{
    Erosion +=.01;
    if (Erosion>=1.0)
    {
        Erosion=0.0;
        if (R>0) R--;
        printf("Novi radjus sujedstva = %d", R);
    }
    if (epoch<500)
    {
        eta = eta-delta_eta/10.0;
    }
    else
    {
       eta = eta-delta_eta;
    }
    if (eta<ETAMIN) eta=ETAMIN;

    printf (" Novo eta=%f\n",eta);
}

void SOFM::PrintWeights()
{
    int ix,iy,k;
    for (ix = 0; ix < YoutSize.x; ix++)
    {
        for (iy = 0; iy < YoutSize.y; iy++)
        {
            for (k = 0; k < YinSize; k++)
            {
```

```
                printf("W[%d][%d][%d]=%f ",k,ix,iy,W[k][ix][iy]);
            }
            printf("\n");
        }
    }
}

void SOFM::RunTrn()
{
    int i,np;
    iPair Winner;
    epoch=0;
    np=Patt->NumPatterns;
    while (epoch <= MAXEPOCHS)
    {
        for (i=0; i<np; i++)
        {
            LoadInLayer(i);
            Winner=FindWinner();
        Train(Winner);
        }
        if(20*(epoch/20)==epoch)
        {
            printf("Epoch=%d\n",epoch);
            PrintWeights();
        }
        epoch++;
        if (StochFlg)
            Patt->ReShuffle(np);
        AdaptParms();
    }
}

void SOFM::Train(struct iPair Winner)
{
    int ix,iy,k;
    for (ix = Winner.x-R; ix <= Winner.x+R; ix++)
    {
        if ((ix>=0) && (ix<YoutSize.x))
        {
            for (iy=Winner.y-R; iy<=Winner.y+R; iy++)
            {
                if ((iy>=0) && iy<YoutSize.y)
                {
                    for (k=0; k<YinSize; k++)
                    {
                        W[k][ix][iy]=W[k][ix][iy]+eta*(Yin[k]-
                              W[k][ix][iy]);
                    }
                }
            }
        }
     }
}

iPair SOFM::FindWinner()
{
    int ix,iy;
    double d,best;
    iPair Winner;
    best=1.0e99;
```

```
    Winner.x=-1;
    Winner.y=-1;
    for (ix=0; ix<YoutSize.x; ix++)
    {
        for (iy=0; iy<YoutSize.y; iy++)
        {
            d=EucNorm(ix,iy);
            if (d<best)
            {
                  best=d;
                  Winner.x=ix;
                  Winner.y=iy;
            }
        }
    }
    return Winner;
}

double SOFM::EucNorm(int x, int y)
{
    int i;
    double dist;
    dist=0;
    for (i = 0; i < YinSize; i++)
    {
        dist += (W[i][x][y]-Yin[i]) * (W[i][x][y]-Yin[i]);
    }
    dist=sqrt(dist);
    return dist;
}


Pattern InPat;
SOFM FMap;

int main(int argc, char *argv[])
{
    if (argc>1)
    {
        InPat.GetPatterns(argv[1]);
        FMap.SetPattern(&InPat);
        FMap.SetParms(5,5,0.900);
        FMap.RunTrn();
    }
    else
    {
        printf("USAGE: SOFM PATTERN_FILE");
    }
}
```

PODACI POTREBNI ZA DIGITALIZACIJU DOKTORSKE DISERTACIJE

Ime i prezime autora Goran Šuković

Godina rođenja 1966

E-mail goransh@ucg.ac.me

Organizaciona jedinica Univerziteta Crne Gore

Prirodno-matematički fakultet

Naslov doktorske disertacije

Generisanje topologije neuronske mreže primjenom sistema Lindenmajera i genetskih algoritama

Prevod naslova na engleski jezik

Generating Neural Network Topologies Using Lindenmayer Systems and Genetic Algorithms

Datum odbrane 18.12.2014.

Signatura u Univerzitetskoj biblioteci[1]

Naslov, saželi, ključne riječi (priložiti dokument sa podacima potrebnim za unos doktorske disertacije u Digitalni arhiv Univerziteta Crne Gore)

Izjava o korišćenju (priložiti potpisanu izjavu)

Napomena

[1] Podatak o signaturi (lokaciji) može ispuniti biblioteka organizacione jedinice/Univerzitetska biblioteka

PODACI POTREBNI ZA UNOS DOKTORSKE DISERTACIJE U DIGITALNI ARHIV UNIVERZITETA CRNE GORE

Prevod naslova disertacije na engleski jezik

Generating Neural Network Topologies Using Lindenmayer Systems and Genetic Algorithms

Mentor i članovi komisija (za ocjenu i odbranu)

Priložen poseban dokument.

Sažetak*

Priložen poseban dokument.

Sažetak na engleskom (njemačkom ili francuskom) jeziku

Priložen poseban dokument.

Ključne riječi Vještačke neuronske mreže, sistemi Lindenmajera, genetski algoritmi, direktno i indirektno kodiranje

Ključne riječi na engleskom jeziku Artificial neural networks, Lindenmayer systems, genetic algorithms, direct and indirect encoding

Naučna oblast/uža naučna oblast

Računarske nauke / Mašinsko učenje

Naučna oblast/uža naučna oblast na engleskom jeziku

Computer Science / Machine Learning

Ostali podaci

* Ukoliko je predviđeni prostor za polja Sažetak, Sažetak na engleskom jeziku, Ključne riječi i Ključne riječi na engleskom jeziku nedovoljan, priložiti ih u posebnom prilogu.

Mentor: Dr Milo Tomašević, vanredni profesor ETF-a u Beogradu

Komisija za ocjenu doktorske disertacije:
Dr Milo Tomašević, vanredni profesor ETF-a u Beogradu
Dr Predrag Stanišić, redovni profesor PMF-a u Podgorici
Dr Milenko Mosurović, redovni profesor PMF-a u Podgorici
Dr Srđan Kadić, docent PMF-a u Podgorici
Dr Savo Tomović, docent PMF-a u Podgorici

Komisija za odbranu doktorske disertacije:
Dr Milo Tomašević, vanredni profesor ETF-a u Beogradu
Dr Predrag Stanišić, redovni profesor PMF-a u Podgorici
Dr Milenko Mosurović, redovni profesor PMF-a u Podgorici
Dr Savo Tomović, docent PMF-a u Podgorici
Dr Aleksandar Popović, docent PMF-a u Podgorici

U ovoj tezi predstavljen je jedan metod automatizovanog generisanja topologija modularnih neuronskih mreža. Metod kombinuje više bioloških paradigmi. Povećanje hardverske moći utiče i na veću dostupnost neuronskih mreža sa velikim brojem neurona. Veličina tih mreža nije ograničena, pa će biti sve teža kreirati ih i razumjeti njihovo funkcionisanje. Iz tog razloga je važno pronaći metod opisa topologije mreže koji će biti skalabilan. Modularne neuronske mreže pokazuju bolje performanse od odgovarajućih nemodularnih mreža. Ljudski mozak se može posmatrati kao modularna mreža, pa je predloženi metod zasnovan na prirodnim procesima u mozgu. Sistemi Lindemanajera (L-Sistemi) se koriste kao mehanizam kodiranja, da bi modelirali recepte rasta prisutne kod biljaka.
Genetski algoritmi su evolucioni lokalni metod traženja u prostoru mrežnih topologija. Koji imaju veliku dimenziju. Oni predstavljaju jedan od optimizacionih metoda koji se mogu primjeniti čak i kada ne postoji analitičko znanje o problemu, pa su pogodni za automatizovano traženje odgovarajućih topologija mreže. Svaka individua (hromozom) u populaciji predstavlja kodirani opis neke topologije. Cilj je da odredimo optimalnu topologiju za dati problem. Kvalitet mreže se izračunava u procesu obučavanja mreže i prevodi se u funkciju kvaliteta, na osnovu koje genetski algoritam određuje vjerovatnoću selekcije individua za operaciju ukrštanja. Usko grlo ovog procesa je, očigledno, vrijeme obučavanja mreže za svaku topologiju.
Neuronska mreža se obučava jednom varijantom metoda „gradijentnog spusta" – algoritmom prostiranja greške unazad („back propagation"). Uvođenje heuristika za dodavanje modula i neurona u toku procesa obučavanja poboljšava brzinu konvergencije i kvalitet samog rješenja. Mogućnosti predloženog metoda su istražene kroz više eksperimenata. Rezultati pokazuju da metod zaista generiše modularne mreže i da modularnost u procesu kreiranja mreže može poboljšati njihove krajnje performanse. Generisane mreže za probleme čiji skupovi za obučavanje nisu veliki imaju bolje performanse od odgovarajućih višeslojnih neuronskih mreža. Pored boljih performansi, modularizacija ima efekat i da su šeme kodiranje skalabilnije, čime se smanjuje prostor koj pretražuje genetski algoritam.

In this thesis, a method for automatic generation of modular artificial neural network topologies is proposed. A number of biological paradigms are incorporated in the method. As computer hardware becomes more powerful, large artificial neural networks will become feasible. It will become increasingly difficult to design them and understand their internal operation There will be no limit to the size of the networks. That is why it is very important to find design methods that are scalable to large network sizes. Modular artificial neural networks have a better performance than their non-modular counterparts. The human brain can also be seen as a modular neural network, and the proposed method is based on the natural process in the brain. Lindenmayer's system (L-systems) are used as a indirect encoding scheme in order to model the kind of recipes that nature uses in biological growth of plants.

Genetic algorithms are used as a evolutionary local search method in high-dimensional space of network topologies. They are an optimization method that can operate even if no analytic knowledge about the problem is available. Therefore, it is a technique that is suitable to automatically search for appropriate network topologies. Each individual in a population contains a encoded description of a network topology and the goal is to find the optimal topology for a given task. Final quality of a network is determined after training it on the training data and the performance of the trained network is transformed into a fitness measure. Fitness measure is used by the genetic algorithm to calculate the selection probabilities of individual. The obvious bottleneck of proposed method is the long training time that is needed to evaluate each network topology.
Neural network is trained using a gradient descent algorithm known as back propagation algorithm. Introducing heuristics for adding modules or nodes during the training process improves the speed of convergence and quality of solution.
A number of experiments have been done to investigate the possibilities of the proposed method. Results show that the method does find modular networks, and using modularity when designing artificial neural networks can improve their performance. Generated networks on small and medium size problems outperforms standard multilayer feed forward networks. Besides the fact that designing modular network topologies specific for a task will result in better performances, modularization also allows for a more scalable coding scheme. In order to minimize the search space of the genetic algorithm, it is very important to keep the size of the encoding as small as possible.

IZJAVA O KORIŠĆENJU

Ovlašćujem Univerzitetsku biblioteku da u **Digitalni arhiv Univerziteta Crne Gore** unese doktorsku disertaciju pod naslovom

Generisanje topologije neuronske mreže primjenom sistema Lindenmajera i genetskih alg

koja je moj autorski rad.

Doktorska disertacija, pohranjena u Digitalni arhiv Univerziteta Crne Gore, može se koristiti pod uslovima definisanim licencom Kreativne zajednice (Creative Commmons), za koju sam se odlučio/la[1].

- ☐ Autorstvo
- ☐ Autorstvo – bez prerada
- ☐ Autorstvo – dijeliti pod istim uslovima
- ☐ Autorstvo – nekomercijalno
- ☐ Autorstvo – nekomercijalno – bez prerada
- ■ Autorstvo – nekomercijalno – dijeliti pod istim uslovima

Potpis doktoranda

Goran Šukurić

U Podgorici

28.6.2019.

[1] Odabrati (čekirati) jednu od šest ponuđenih licenci (kratak opis licenci dat je na poleđini ovog priloga)

Autorstvo
Licenca sa najširim obimom prava korišćenja. Dozvoljavaju se prerade, umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio).

Djelo se može koristiti i u komercijalne svrhe.

Autorstvo – bez prerada
Dozvoljava se umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Djelo se ne može mijenjati, preoblikovati ili koristiti u drugom djelu.

Licenca dozvoljava komercijalnu upotrebu djela.

Autorstvo – dijeliti pod istim uslovima
Dozvoljava se umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Ukoliko se djelo mijenja, preoblikuje ili koristi u drugom djelu, prerade se moraju distribuirati pod istom ili sličnom licencom.

Ova licenca dozvoljava komercijalnu upotrebu djela i prerada. Slična je softverskim licencama, odnosno licencama otvorenog koda.

Autorstvo – nekomercijalno
Dozvoljavaju se prerade, umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio).

Komercijalna upotreba djela nije dozvoljena.

Autorstvo – nekomercijalno – bez prerada
Licenca kojom se u najvećoj mjeri ograničavaju prava korišćenja djela. Dozvoljava se umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Djelo se ne može mijenjati, preoblikovati ili koristiti u drugom djelu.

Komercijalna upotreba djela nije dozvoljena.

Autorstvo – nekomercijalno – dijeliti pod istim uslovima
Dozvoljava se umnožavanje, distribucija, javno saopštavanje i prerada djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Ukoliko se djelo mijenja, preoblikuje ili koristi u drugom djelu, prerada se mora distribuirati pod istom ili sličnom licencom.

Djelo i prerade se ne mogu koristiti u komercijalne svrhe.