SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2587

OPTIMIZACIJA NEURONSKE MREŽE UZ POMOĆ GENETSKOG ALGORITMA

Luka Marasović

Zagreb, lipanj 2012.

Sadržaj

Uvod

Predmet ovog završnog rada je korištenje genetskog algoritma za optimiziranje neuronske mreže koja upravlja modelom jednostavnih organizama u simuliranom okruženju, te analiziranje utjecaja pojedinih parametara genetskog algoritma i neuronske mreže na brzinu konvergencije optimalnim rješenjima. Simulacijsko okruženje se sastoji od modela jednostavnih organizama koji preživljavaju konzumirajući energiju, bilo putem hrane koja se nalazi u simulaciji, ili od drugih organizama. Organizmi u sebi sadrže neuronsku mrežu, koja na temelju ulaznih podataka poziva akcije dostupne organizmu. Oni organizmi koji preživljavaju duže nego ostali dobivaju priliku stvaranja novih organizama. Jednom kad se u simulaciji nađe organizam koji je konzumirao dovoljnu količinu energije, tj. neuronska mreža zadovoljavajućih karakteristika simulacija se zaustavlja. Na kraju izvršenih ispitivanja raznih parametara se evaluiraju rezultati.

Poglavlje „Genetski algoritmi“ daje uvod u tehnike evolucijskog računanja, te genetske algoritme. Također je dan uvid u procese iz prirode koja su potakla razvoj ovih tehnologija.

Poglavlje „Neuronske mreže“ pokriva povijest umjetnih neuronskih mreža, pregled bioloških neuronskih mreža i procesa koji su inspirirali razvoj ove tehnologije.

Poglavlje „Programsko ostvarenje“ navodi i opisuje korištene tehnologije, te funkcionalne dijelove programske podrške.

Poglavlje „Rezultati“ navodi pristup ispitivanju, parametre zaustavljanja simulacije te sadrži grafove ispitivanja i kratka obrazloženja istih.

1. Genetski algoritmi

1.1. Prirodna selekcija

Charles Darwin, engleski znanstvenik i prirodoslovac koji je 1859. godine objavom knjige "Postanak vrsta" predstavio načelo prirodne selekcije, donosi i početnu točku ideje evolucijskih algoritama. Biološke vrste su prirodnom selekcijom pronašle rješenja za probleme nelinearnih interakcija i kompleksnih uvjeta kojima se potrebno prilagoditi. Isti koncept se može primijeniti na probleme gdje klasičan pristup rješavanju daje nezadovoljavajuće rezultate ili iziskuje previše vremena.

Teorija prirodne selekcije nalaže da su biljke i životinje koje postoje danas rezultat milijuna godina prilagodbe na uvjete okoliša. Unutar nekog područja može obitavati više organizama i natjecati se za iste resurse kako bi preživjeli. Oni organizmi koji su najsposobniji u izvršavanju ove zadaće propagiraju svoj genetski materijal - informacije o sebi na potomke. Organizmi koji su manje uspješni s druge strane, imat će manje prilika za razmnožavanjem i prosljeđivanjem informacija na sljedeće generacije. Obično kažemo da je organizam manje ili veće dobrote (engl. *fitness*) naspram drugog organizma. Tijekom vremena, cijela populacija organizama veće dobrote prevlada, te je prosjek dobrote nove populacije bolji od prethodne jer imaju poželjnije karakteristike za uvjete u kojem se nalaze.

Nositelj informacija o pojedinom organizmu je njegova molekula deoksiribonukleinske kiseline (DNK), koja predstavlja njegov genetski materijal koje je naslijedio od roditelja. Kada nastaje nove jedinka, ona križanjem dobiva pola genetskog materijala od jednog, a pola od drugog roditelja, tako da novonastala jedinka ima jednaku dužinu DNK. Kako je DNK nositelj informacija o roditeljima, dijete ima svojstva svojih roditelja. Još jedan značajan proces se odvija na genetskoj šifri a to je mutacija prilikom koje se određeni gen mijenja.

1.2. Evolucijsko računanje

Evolucijsko računanje (engl. *evolutinary computation* EC) je oponašanje evolucije na računalu. Algoritmi evolucijskog računanja preuzimaju principe evolucije i prirodne selekcije, te mogu poslužiti za pronalaženje optimalnih rješenja za zadani problem. Ključna razlika između evolucijskog algoritma pretrage i tradicionalnih algoritama jest prisutnost populacije. Evolucijski algoritmi obavljaju efikasnu usmjerenu pretragu prostora rješenja, i općenito imaju bolje rezultate od nasumične pretrage. Evolucijsko računanje je podijeljeno na četiri glavne skupine: [1]

1. genetski algoritmi (engl. *genetic algorithms*, GA)
2. genetsko programiranje (engl. *genetic programming*, GP)
3. evolucijske strategije (engl. *evolution strategies*, ES)
4. evolucijsko programiranje (engl. *evolutionary programming* EP)

Genetski algoritmi (u nastavku teksta GA) je jedna od četiri tehnika evolucijskog računanja, i ujedno jedna od popularnijih. U tradicionalnom pristupu izvedbi GA, za reprezentaciju genetskog materijala se koristi binarni kod ili binarni prikaz. Odabir prikaza je izuzetno bitan i utječe na performanse algoritma. Većina teorije genetskih algoritama je vezana uz binarni prikaz te se u praksi pokazalo da u većini primjera gdje se može iskoristiti daje najbolje rezultate. Glavni operator križanja (engl. *crossover*) koji se koristi je *bit-string crossover*, gdje *bit-string* djeteta nastaje tako da se uzme jedna sekvenca od jednog roditelja, a ostatak od drugog. Još jedan popularni operator je *bit-flipping mutation*, gdje se pojedini bit roditelja mijenja i tako nastaje potomak. Postoje još mnogi operatori križanja koji su u upotrebi, kao što je uniformno križanje i inverzija sekvenci binarnog koda. Odabir roditelja za križanje se izvodi probabilistički na temelju funkcije dobrote, koja određuje koliko je neka jedinka poželjna. Obično se stvara N novih jedinki od N odabranih roditelja. Prisutan je i operator mutacije, koji je analogan u biologiji, gdje se s malom vjerojatnošću, pojedini gen promjeni, tj. pojedini bit zapisa se invertira.[2]

GA se nakon početne inicijalizacije populacije vrti u petlji sa sljedećim koracima:

SELEKCIJA: Prvi korak je selekcija jedinki za reprodukciju. Selekcija se obavlja nasumično sa vjerojatnostima ovisnima o relativnoj dobroti jedinke u populaciji tako da su

bolje jedinke češće odabrane.

REPRODUKCIJA: Stvaraju se potomci od odabranih jedinki. Za stvaranje novih kromosoma (op.a. kromosom je nakupina DNK skupljena na mali prostor, naziv je analogan za genetski materijal ili opis potencijalnog rješenja u genetskom algoritmu), algoritam koristi operatore križanja i mutacije.

EVALUACIJA: Određuje se dobrota novih kromosoma.

ZAMJENA: Tijekom zadnjeg koraka, zamjenjuju se jedinke stare populacije sa novima.

Ovi koraci se ponavljaju dok populacija ne konvergira prema optimalnom rješenju.

1.3. Usporedba s drugim tehnikama

Kako bi bolje shvatili koje su mogućnosti GA, bitno je naglasiti ključne stavke u kojima se GA razlikuje od konvencionalnih tehnika optimizacije:

1. GA radi s kodiranom verzijom (npr. *bit-string*) problemskih parametara a ne sa samim parametrima, tj. GA radi s kodiranim rješenjem umjesto direktno s rješenjem.
2. Sve konvencionalne optimizacijske tehnike pretrage traže iz jedne točke dok se GA izvodi na cijeloj populaciji točaka rješenja. Ovo igra veliku ulogu u robusnosti genetskih algoritama jer se povećavaju izgledi pronalaska globalnog optimuma i pomaže izbjegavanje lokalnih optimuma.
3. GA koristi funkciju dobrote za evaluaciju umjesto derivacije, čime se omogućava aplikacija na kontinuirane i diskretne optimizacijske probleme ako je osmišljena dobra funkcija izračunavanja dobrote.
4. GA koristi probabilističke tranzicijske operatore, tj. GA ne koristi deterministička pravila.

1.4. Prednosti i ograničenja

Neke od prednosti genetskih algoritama su:

- Paralelizam
- Veliki prostor rješenja
- Kompleksni prostor funkcije dobrote
- Lako otkrivanje globalnog optimuma
- Lako upotrebljiv na različite probleme
- Snalazi se dobro s funkcijama šuma
- Snalazi se dobro s velikim, slabo razumljivim prostorom rješenja
- Robustan rad
- Otpornost na lokalne optimume

Neke od ograničenja genetskih algoritama uključuju:

- Prepoznavanje funkcije dobrote
- Definiranje reprezentacije problema
- Pojava prerane konvergencije
- Odabir parametara kao što su veličina populacije, vjerojatnost mutacije, križanja, način odabira jedinki
- Teško uključivanje informacija specifičnih problemu
- Loše prepoznavanje lokalnih optimuma
- Nepostojanje učinkovitog načina zaustavljanja
- Problem pronalaska globalnog optimuma
- Zahtjeva velik broj evaluacija funkcije dobrote

1.5. Primjene

Genetski algoritmi su korišteni za rješavanje NP-teških problema, za strojno učenje i evoluciju jednostavnih programa. Neke od primjena su sljedeće:

- Planiranje putanje robota
- Planiranje strategije
- Pronalazak oblika proteina
- Problem trgovačkog putnika
- Funkcije za kreiranje slika
- Dizajniranje zrakoplova, strujnih krugova, tipkovnica, komunikacijskih mreža
- Kombinatorne optimizacije
- Strojno učenje – dizajn neuronskih mreža, kako arhitekture tako i težina, poboljšavanje klasifikacijskih algoritama

2. Neuronske mreže

2.1. Povijest

Neuronske mreže se koriste na računalima još od 1950-ih godina. Kroz godine napravljene su mnoge varijante neuronske mreže, a jedna od najranijih praktičnih varijanti je perceptron koji je razvijen 1957. od Frank Rosenblatta na Cornellovom aeronautičkom laboratoriju. Perceptron je bio pokušaj razumijevanja ljudske sposobnosti pamćenja, učenja i kognitivnih procesa. 1960. Rosenblatt je demonstrirao rad „Mark I“ perceptrona koji je imao mogućnost „naučiti“ prepoznavati optičke uzorke.

Razvoj perceptrona je počeo putem proučavanja biologije, te je dvojac Warren McCulloh i Walter Pitts bio prvi koji je opisao biološku neuronsku mrežu, i koji se smatraju tvorcima izraza „neuronska mreža“ (engl. *Neural network*). Razvili su pojednostavljeni model neurona, nazvan MP neuron, koji je imao u sebi okosnicu ideje da neuron daje izlaz jedino ako je vrijednost praga (engl. threshold) prijeđena. MP Neuron je funkcionirao kao vrsta skenirajućeg uređaja koji čita predefinirane ulazne i izlazne asocijacije kako bi odredio konačni izlaz, ali nije imao mogućnost učenja jer je imao fiksirane pragove. Umjesto mijenjanja pragova, ručno se mijenjala povezanost takve neuronske mreže. Upravo njegova nemogućnost učenja je značila veliku ograničenost naspram ljudskog mozga i neurona po kojem je perceptron modeliran, pa je Rosenblatt zaključio da model mreže koji uči može popraviti svoj odziv namještajući težine veza između neurona. Iako je perceptron pokazivao mnogo obećanja, ipak je imao neke prepreke koje nije mogao zaobići.[3]

Između 1959. i 1960., Bernard Wildrow i Marcian Hoff sa Sveučilišta na Standfordu, USA razvijaju ADALINE (Adaptive Linear Elements) i MADELINE (Multiple Adaptive Linear Elements) modele. Ovo su prve neuronske mreže koje je bilo moguće primijeniti na stvarne probleme, te je ADALINE model korišten za uklanjanje jeke iz telefonskih linija.

Period od 1969. do 1981. je rezultirao u povećanom zanimanju za neuronske mreže. Njihove sposobnosti su uveličane od pisaca i producenata knjiga i filmova, ljudi su vjerovali u mogućnosti neuronskih mreža koje su iste ili superiornije čak i od ljudskog mozga što je rezultiralo u mnogo razočaranja i kritike naspram umjetne inteligencije, čime je financiranje razvoja bilo u velikoj mjeri zanemareno.

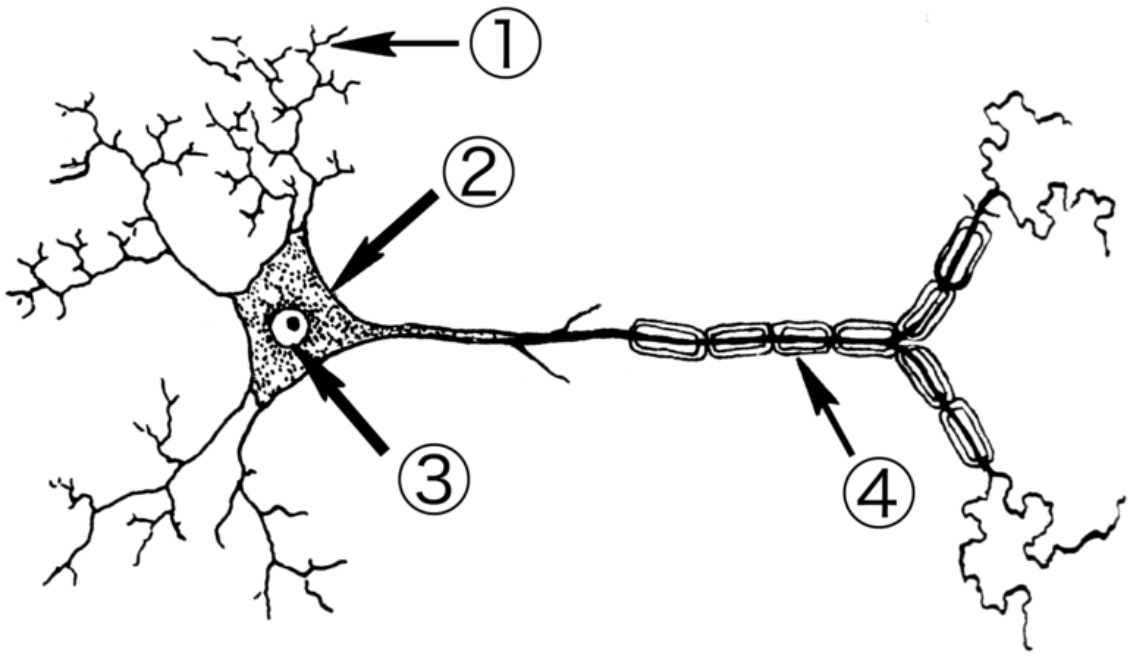
1986. godine je održana prva konferencija „Neural Networks for Computing“ koja je privukla preko 1800 sudionika. Sa početkom 1990-ih, neuronske mreže izlaze iz mraka i dolaskom mnogih tehnoloških dostignuća dolaze i mnoge inovacije i razvoj polja umjetnih neuronskih mreža diljem svijeta.

2.2. Biološke neuronske mreže

2.2.1. Struktura neurona

Kad govorimo o neuronskim mrežama na računalima, izuzimamo riječ „umjetne“ zbog konteksta, što se podrazumijeva i u ovom radu, osim ako nije naglašeno drugačije. Tradicionalno, pojam „neuronske mreže“ se odnosi na biološku neuronsku mrežu.

Biološka neuronska mreža je građena od neurona koji su fizički ili funkcionalno povezani u periferni živčani sustav ili središnji živčani sustav. Konstruiranje računala koje je kompleksno poput ljudskog mozga, imajući na umu da se sastoji od 6 milijardi međusobno povezanih neurona, je teška zadaća, pa je u tu svrhu osmišljen jednostavniji model koji se sastoji od manjeg broja jednostavnih modela neurona koji trebaju simulirati rad biološkog neurona.



Slika 1 Struktura neurona

Neuron ili živčana stanica je osnovna gradivna jedinica živčanog sustava i najsloženija je u ljudskom organizmu. Njihova uloga se može smatrati kao prihvaćanje, obrađivanje te odašiljanje podataka. Na slici 1 vidljiva su 4 karakteristična dijela strukture biološkog neurona:

1. Dendriti
2. Soma
3. Jezgra
4. Akson

Dendriti (od grčkog δένδρον *déndron*, „drvo") su kraći produžeci koji s osjetnih organa ili drugih živčanih stanica dovode živčano uzbuđenje na tijelo stanice.

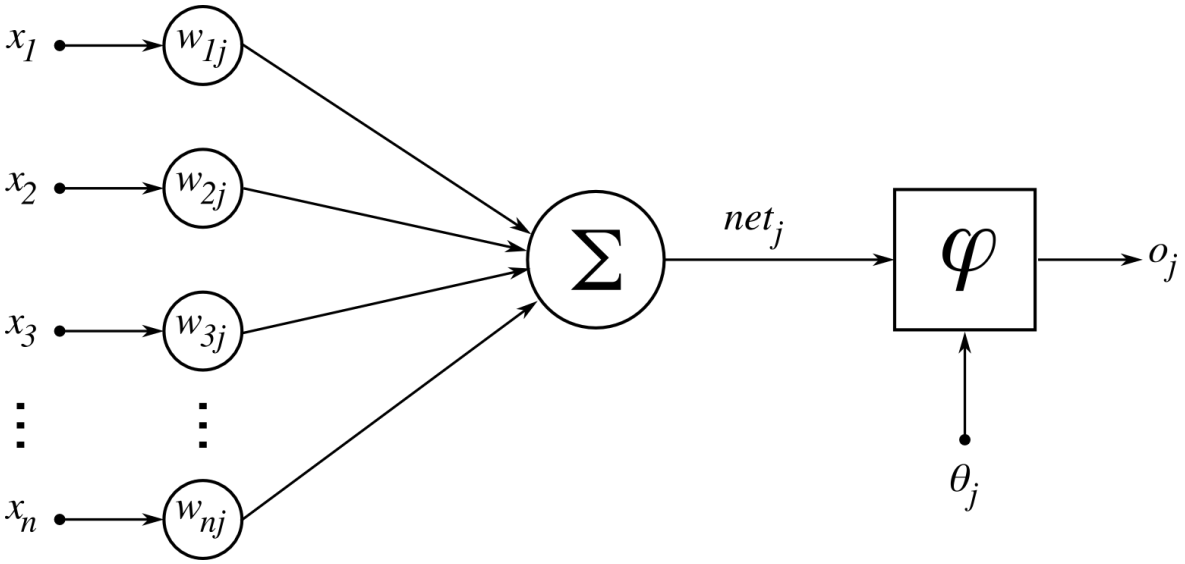
Soma (od grčkog *soma*, „tijelo") ili tijelo stanice, je deblji kraj neurona i sadrži jezgru koja regulira i proizvodi većinu potrebnih proteina neuronu kojem pripada iz pripadne joj DNK.

Akson je duži produžetak neurona, sa dužinom od nekoliko mikrometara do jednog metra. Počinje na tijelu stanice i glavna funkcija mu je prijenos živčanih impulsa sa some na druge živčane stanice ili izvršne organe poput mišićnih vlakana ili žlijezdi. [4]

2.2.2. Akcijski potencijal

Kako pristižu podražaji na dendrite, dolazi do promjena na membrani istih - otvaraju se ionski kanali i pozitivno nabijeni Na+ ioni prolaze kroz pore u neuron, čime se mijenja diferencijalni potencijal sa -70 mV na +40 mV na kratko vrijeme i neuron je u depolariziran. U tom trenutku pozitivno nabijeni K+ izlaze vani čime se potencijal vraća u početno stanje -70 mV. Nakon ovog ciklusa, neuron je neko vrijeme nepodražljiv. Nakon depolarizacije, brzo uslijedi polarizacija, te se sve odvija unutar 1 milisekunde. Neuron je u svom stanju mirovanja polariziran, s vanjske strane membrane dominira Na+ pozitivan navoj, a s unutarnje K+ negativan naboj. Izmjena ovih dvaju procesa nazivamo akcijski potencijal. Veličina akcijskog potencijala na mjestu podražaja je uvijek ili maksimalna ili je nema jer ako je stimulus veći od praga on izaziva potpunu depolarizaciju. Akcijski potencijal se ne događa odjednom na cijelom neuronu, već putuje niz membranu koja okružuje neuron. Putovanje akcijskih potencijala niz akson ili uz dendrit je osnovni mehanizam za prijenos informacija na mozgu.

2.3. Modeli neurona



Slika 2 Model neurona

Općeniti model umjetnog neurona dan je na slici 2 kojeg možemo dodatno razmatrati po ugrađenoj prijenosnoj funkciji koja slijedi nakon sumiranja ulaza ($x_0..x_n$) pomnoženih težinama ($w_0..w_n$). Najjednostavnija moguća aktivacijska funkcija je

$$f(net) = net$$

Ova aktivacijska funkcija je korištena na već spomenutoj ADALINE neuronskoj mreži. Izlaz je težinska suma ulaza. Druga mogućnost je korištenje funkcije skoka ili praga:

$$f(net) = \begin{cases} 0, za\ net < 0 \\ 1, inače \end{cases}$$

Ovakva prijenosna funkcija se naziva TLU (engl. Threshold Logic Unit) i daje Booleov izlaz. Prijenosna funkcija može davati i linearni izlaz na određenom dijelu:

$$f(net) = \begin{cases} 0, za\ net \leq a \\ net, za\ a < net < b \\ 1, za\ net \geq b \end{cases}$$

Najčešće korišteni oblik prijenosne funkcije jest sigmoidalna funkcija koja ima karakteristiku derivabilnosti što je bitna prednost pri postupku učenja neuronske mreže.

$$f(net) = \frac{1}{1+e^{-a \cdot net}}$$

Ova funkcija se naziva i logistička funkcija. Parametar a određuje nagib funkcije. [5]

2.4. Arhitektura važnijih neuronskih mreža

Svaka neuronska mreža je građena od pojednostavljenih modela bioloških neurona (Slika 2) koji primaju težinske ulaze, sumiraju ih, te primjenom prijenosne funkcije koja je ekvivalent funkcioniranju akcijskog potencijala u biološkom neuronu daju izlaz. Po načinu povezivanja između neurona razlikujemo neuronske mreže:

- Feed-forward mreže – signali putuju samo u jednom smjeru, od ulaza prema izlazu.
- Feedback mreže – signali mogu putovati u oba smjera zbog postojanja petlji
- Mrežni slojevi – najčešće 3 sloja: ulazni, skriveni i izlazni

2.5. Učenje

Najvažniji koncept koji donose neuronske mreže u svijet računalnih algoritama je sposobnost učenja koje je moguće ostvariti na više načina, ali je osnova svih njih modifikacija težinske matrice za veze između pojedinačnih neurona. U ovisnosti je li nam poznat izlaz iz mreže pri postupku učenja razlikujemo dva načina učenja:

- Učenje s učiteljem (engl. supervised learning) – učenje se odvija u obliku para (ulaz, izlaz)
- Učenje bez učitelja (engl. unsupervised learning) – mreža uči bez poznavanja izlaza

Moguća je pojava i *prenaučenosti* , pa su primjeri za učenje podijeljeni u 3 skupa:

1. Učenje
2. Ispitivanje
3. Provjera

Primjeri iz prvog skupa podešavaju težinske faktore, drugi skup provjerava rad mreže kako bi se uočio trenutak degradacije performansi mreže što je posljedica specijalizacije mreže na skup za učenje čime gubi svojstvo generaliziranja. Posljednji skup se koristi za provjeru točnosti i preciznosti neuronske mreže.

Promjena težinskih faktora se izvodi izračunom pogreške i minimiziranjem odstupanja. Učenje se događa kad mijenjamo težine pojedinih veza kako bi dobili poželjne rezultate. Ovaj proces zahtijeva puno ponavljanja i teško je razlučiti što je neuronska mreža stvarno naučila.

2.6. Prednosti i ograničenja

Prednosti neuronskih mreža:

- Dobre u procjeni nelinearnosti
- Rad s nejasnim podacima
- Robusnost na pogreške u podacima
- Prilagodljivost okolini
- Velik broj varijabli

Nedostaci:

- Ne postoji nužno skup za učenje
- Nema objašnjenja za dobivenu klasifikaciju
- Rezultat ovisi o početnom stanju mreže
- Ne postoji kvalitetan način obrade podataka

2.7. Primjena

Obično se primjenjuju u sljedećim problemima:

- Prepoznavanje uzoraka
- Klasifikacija
- Predviđanja

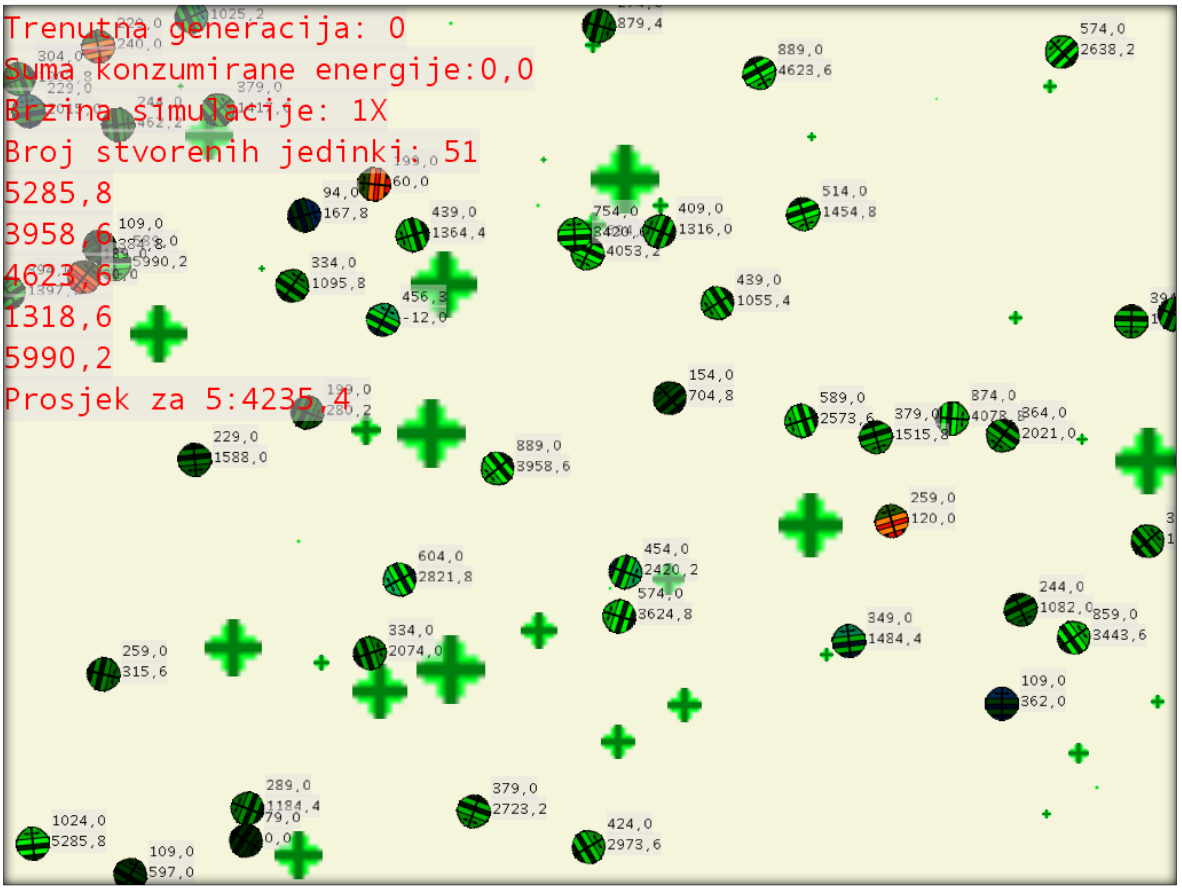
Prepoznavanje uzoraka je najčešći problem s kojim se bave neuronske mreže. Predstavljen im je uzorak, slika, zvuk ili već neki podaci, a neuronska mreža pokušava utvrditi da li ulaz odgovara uzorku kojeg je naučila prepoznati.

Klasifikacija je proces sličan prepoznavanju, no ovdje neuronska mreža želi podijeliti podatke u više grupa, stvarajući time klasifikaciju između podataka.

Predviđanje je također tipična primjena za neuronske mreže, gdje neuronska mreža na temelju ulaznih podataka klasificiranih vremenski želi utvrditi buduće vrijednosti. Preciznost rezultata ovisi o više faktora, kao što je kvantiteta i kvaliteta ulaznih podataka.

3. Programsko ostvarenje

3.1. Simulacijsko okruženje



Slika 3 Simulacijsko okruženje

Simulacijsko okruženje prikazano na slici 3 se sastoji od modela jednostavnih organizma koji se kreću u dvodimenzionalnom prostoru u kojem se nalazi hrana. Organizmi su predstavljeni kao krugovi sa crnim prugama te okomitom crtom kako bi vizualizirali smjer u kojem je orijentiran pojedini organizam. Dodatno pokraj svakog organizma stoje 2 broja koji predstavljaju količinu energije koju posjeduje, te količinu konzumirane energije. Organizam dobiva energiju na dva načina:

- Prelaskom preko hrane
- Prelaskom preko drugog organizma

Prilikom konzumiranja energije, organizam oduzima energiju pojedinoj hrani, tj. organizmu. Ako organizam dosegne preveliku količinu energije, biva kažnjen oduzimanjem određene količine energije kako bi se izbjeglo prejedanje.

Prilikom stvaranja pojedinog organizma, on sadrži inicijalnu količinu energije koja opada s prolaskom vremena. Količina energije te njeno podrijetlo definira boju organizma:

- Crna boja – organizam je tamniji što ima manje energije
- Crvena boja – organizam je crveniji ako je energija većinom od drugih organizama

Hrana je predstavljena zelenim plusevima čija veličina predstavlja količinu energije koju sadrži. Hrana također gubi energiju poput organizama, što je vidljivo postupnim smanjivanjem veličine. Broj hrane je konstantan u simulaciji, te potpuni gubitak energije uzrokuje stvaranje nove hrane s nasumičnom količinom energije.

Kretanjem svakog organizma upravlja zasebna neuronska mreža slojevite strukture koja se sastoji od 8 ulaza i 3 izlaza. Ulazi neuronske mreže pojedinog organizma imaju vrijednosti od 0 do 1 te imaju sljedeću strukturu:

- 3 ulaza hrane – organizam 'vidi' koliko hrane se nalazi lijevo, ispred te desno od njega. Vrijednost 0 označava odsustvo hrane.
- 3 ulaza za druge organizme – organizam 'vidi' koliko drugih organizama se nalazi lijevo, ispred te desno od njega. Vrijednost 0 označava odsustvo drugih organizama.
- 1 ulaz vlastite razine energije – organizam prima skaliranu vrijednost vlastite energije gdje vrijednost 1 označava maksimalnu energiju koju organizam može imati.
- 1 ulaz opasnosti – budući da organizmi nemaju ulaz koji bi označavao prisutnost drugog organizma koji nije u vidokrugu, ovaj ulaz predstavlja 'sluh' organizma gdje vrijednosti blizu 1 označavaju neposrednu udaljenost drugog organizma, a 0 odsutnost organizama unutar radijusa 'sluha'.

Izlazi neuronske mreže imaju vrijednosti od 0 do 1 te upravljaju akcijama kretanja organizma:

- Rotiranje lijevo
- Rotiranje desno
- Pomak naprijed

Budući da je broj organizama u simulaciji konstantan, jednom kad određeni organizam izgubi svu energiju, uklanja se iz simulacije te ga zamjenjuje novi. Pri stvaranju novog organizma, genetski algoritam odabire dvije jedinke trenutne populacije koje su konzumirale najviše energije, te mu stvara pripadajuću neuronsku mrežu koja je rezultat križanja i mutacije neuronskih mreža odabranih roditelja.

U lijevom gornjem kutu prikaza simulacije, vidljivi su statistički podaci izvođenja:

- Trenutna generacija
- Ukupna količina konzumirane energije svih jedinki
- Brzina simulacije
- Broj stvorenih jedinki
- Količina konzumirane energije za prvih N najboljih jedinki
- Prosjek konzumirane energije za prvih N najboljih jedinki

3.2. Tehnologije

Za izradu implementacije korišteno je Visual Studio 2010 razvojno okruženje. Visual Studio podržava više različitih programskih jezika kao što su:

- C#
- C/C++
- VB.NET
- F#

C# je jezik korišten u implementaciji zbog jednostavnosti i suvremenosti, te je ujedno programski jezik koji najviše reflektira Common Language Infrastructure (CLI) kojeg svi navedeni jezici moraju podržavati kako bi radili s .NET Frameworkom, infrastrukturi koja programerima nudi gotova rješenja i funkcionalnost za brži i pojednostavljeni razvoj aplikacija. Ova karakteristika slijedi iz činjenice da je C# - objektno orijentirani programski jezik koji je nastao s ciljem da .NET Framework dobije programski jezik koji bi maksimalno iskorištavao infrastrukturni potencijal.

3.2.1. XNA Framework

XNA Framework je skup alata za upravljanje i izradu video igara te se integrira u Microsoft-ovo Visual Studio razvojno okruženje. Naziv XNA dolazi od izvornog imena

"Xbox New Architecture" za koju je porodicu igraćih konzola prvotno bio namijenjen. Inačica XNA Framework 4.0 koja se koristi u ovom radu, donosi run-time environment podršku i za Windows programe. Iako tehnički donosi podršku za sve .NET podržane jezike, C# je jedini službeno podržan. XNA Framework brine o tehničkim detaljima platforme na kojem se izvodi simulacija, te daje grubu arhitekturu na kojoj je moguće izgraditi program.[6]

Glavni razred pri izradi projekta sa XNA Framework-om jest *Game Class*, koji sadrži u sebi više metoda koje kontroliraju tok simulacije, a to su:

- Initialize
- LoadContent
- UnloadContent
- Update
- Draw

Initialize - poziva se jednom nakon što je stvoren *Game Class*, i sadrži u sebi inicijalizaciju varijabli simulacije.

LoadContent - također se poziva jednom te pomoću nje dohvaćamo teksture, korištene fontove i druge sadržaje grafike, zvuka i sl. Sadržaj koji pozivamo u ovoj metodi, mora se prethodno dodati u zaseban dio projekta zvan *Content*, kako bi o njemu brinuo *Content Manager*.

UnloadContent - koristimo jedino ako program koristi sadržaje koje *Content Manager* ne prepoznaje.

Update - poziva se naizmjenično s metodom *Draw* tijekom izvođenja programa. Kao što joj samo ime govori, u nju je poželjno staviti sve aspekte simulacije koje je potrebno ažurirati, i ona se poziva do 60 puta u jednoj sekundi.

Draw - iscrtava grafiku pojedinog trenutka simulacije.

3.2.2. XNA Debug Terminal

XNA Debug Terminal je biblioteka čiji su autori Kevin. A. Cherry i Tiomthy W. Wright čiji je rad dostupan na web stranici *www.protohacks.net/xna_debug_terminal*. Biblioteka stvara prikaz iznad simulacije koji omogućava pozivanje i postavljanje vrijednosti varijabli te poziv metoda prilikom izvođenja simulacije. Ova funkcionalnost će omogućiti

interakciju simulacijske implementacije i olakšati praćenje vrijednosti u realnom vremenu te pokretanje simulacije uz različite parametre.

3.3. Struktura

Struktura izvornog koda implementacije je podijeljena u više razreda koje međusobno sudjeluju kako bi se program ispravno izvodio. Ti razredi su:

1. *Program*
2. *NeuralNetwork*
3. *GeneticAlgorithm*
4. *Entity*
5. *Food*
6. *Simulation*

3.3.1. Razred „Program“

Program se prvi poziva prilikom pokretanja programa. On služi za pozivanje objekta razreda *Game* koji je karakterističan XNA Framework-u. Također se unutar ovog razreda provjerava sustav na kojem se trenutno izvodi program, tj. je li program pokrenut na igraćoj konzoli ili osobnom računalu.

3.3.2. Razred „NeuralNetwork“

Ova razred implementira operacije nad neuronskom mrežom od kojih su važnije:

- `List<double> CalculateOutput (List<double> inputs)`
- `UpdateWeightMatrix(List<BitArray[][]> chromosome, double minWeight, double maxWeight)`

Prva metoda vraća listu izlaza tipa *double* na temelju dobivenih ulaza. Prilikom računanja prolazi kroz težinski zapis neuronske mreže koji je definiran kao lista dvodimenzionalnih polja ili matrica gdje je izlaz prethodne ulaz sljedećoj matrici. Varijabla težinske matrice ima sljedeći izgled:

```
List<double[][]> _weightMatrix
```

Pojedine matrice (tipa *double[][]*) varijable *_weightMatrix* sadrži težinski zapis između dva sloja u neuronskoj mreži. Slojevi su u potpunosti povezani prethodnim i sljedećim

slojem ako postoji čime ova struktura odgovara slojevitoj neuronskoj mreži čiji su slojevi potpuno povezani. Neuroni kao izlaz daju aktivacijsku funkciju, u ovom slučaju tangens hiperbolni, koji može poprimiti vrijednosti od -1 do 1, dok posljednji sloj neuronske mreže koristi sigmoidnu funkciju koja poprima vrijednosti od 0 do 1.

Iako je druga metoda zapravo pomoćna metoda, ona obavlja ključan dio implementacije, a to je prevođenje binarnog zapisa težinske matrice varijable *chromosome* (tip *List(BitArray[][])*) u brojčani zapis interne varijable *_weightMatrix* (tip *List(double[][])*) s kojom je kasnije moguće jednostavno obavljati izračun izlaza.

3.3.3. Razred „GeneticAlgorithm“

GeneticAlgorithm u konstruktoru prima parametre:

- *populationNumber* – veličina populacije
- *chromosomeDesign* – definicija veličine slojeva neuronske mreže a samim time i kromosoma
- *geneLength* – duljina binarnog zapisa pojedinog gena u kromosomu
- *mutationRate* – vjerojatnost mutacije

Ovaj razred sadrži metode koje vrše operacije nad binarnim zapisom težinske matrice neuronske mreže. Za spremanje pojedinog gena kromosoma koristi se *BitArray* koja nudi binarni zapis varijabilne duljine te jednostavno izvođenje operacija nad istima. Struktura kromosoma je slična implementaciji težinske matrice u razredu *NeuralNetwork*, tj. ovdje govorimo o listi matrica binarnih zapisa određene dužine:

```
List<BitArray[][]> chromosome
```

Iako ovaj razred ima metode koje vraćaju ovakve strukture podataka, instancirani objekt razreda *GeneticAlgorithm* u sebi ne čuva kromosome cijele populacije zbog načina na koji se evaluira dobrota pojedinog rješenja unutar simulacije i način selekcije roditelja za novu jedinku. Sami kromosom, tj. varijabla *chromosome* je spremljena unutar objekta razreda *NeuralNetwork*, te se koristi samo ako je jedinka odabrana za razmnožavanje.

Dvije su bitne metode ovog razreda:

- `List<BitArray[][]>ChildFromParents(firstParent, secondParent)`
- `List<BitArray[][]>RandomBitArrayChromosome()`

Prva metoda prima dva kromosoma od roditelja te vraća jedan novi kromosom. Budući da je broj jedinki u simulaciji konstantan, stvara se samo jedna nova jedinka.

```
funkcija ChildFromParents(firstParent, secondParent)
 kromosom = stvori novi prazni kromosom
 za svaki gen od kromosoma
  gen = križaj(gen od firstParent, gen od secondParent)
  gen = mutiraj(gen)
 kraj
 vrati kromosom
```

Slika 4 Pseudokod funkcije stvaranja novog kromosoma

Funkcija stvaranja kromosoma na slici 4 djeluje po pojedinačnim genima jer kromosom nije linearan binarni zapis već varijabilan broj dvodimenzionalnih polja koja pak u sebi sadrže binarne zapise također varijabilne duljine, pa je jednostavnije imati jednu funkciju križanja i jednu funkciju mutacije koje djeluju bez obzira na parametre kromosoma. Ovakva struktura kromosoma je kako bi izbjegli kompliciranа prebacivanja linearnog binarnog zapisa u strukturu s kojom je moguće lako računati unutar *NeuralNetwork* razreda. Također je ovom implementacijom olakšano izvođenje operacija križanja s točkama prekida, gdje funkcija križanja samo nasumično uzima gene kromosoma od roditelja.

Druga metoda, *RandomBitArrayChromosome()* vraća nasumično generiran kromosom.

Koraci su slični pseudokodu na slici 4 jedino što umjesto operacija križanja i mutacije u četvrtom i petom redu, koristimo operaciju stvaranja nasumičnog gena.

3.3.4. Razred „Food“

Food predstavlja hranu koja je dostupna jedinkama u simulaciji dok na pojedini objekt ovog razreda možemo gledati kao na pojedinu biljku. Objekti sadrže dva svojstva: *FoodPosition* i *Energy* koje definiramo kao ulazne parametre prilikom pozivanja konstruktora.

```
Public Food(double energy, Vector2 _position)
```

Objekti razreda *Food* pri stvaranju mogu imati nasumičnu količinu energije između 0 i maksimalne definirane unutar simulacije.

3.3.5. Razred „Entity“

Entity predstavlja jedinke tj. organizme koje se kreću u simulaciji. Svaki objekt ovog razreda sadrži sljedeća svojstva:

- `NeuralNetwork Nn` – svaki objekt *Entity* ima pripadajući objekt razreda *NeuralNetwork* koji upravlja njegovim kretanjem
- `double EnergyConsumed` – predstavlja količinu energije koju je jedinka konzumirala iz simuliranog okruženja
- `double Energy` – količina energije koju posjeduje jedinka
- `float AngleOfMovement` – smjer u kojem se kreće jedinka
- `Vector2 EntityPosition` – koordinate gdje se nalazi jedinka gledano gdje se kao izvorišna točka uzima gornji lijevi kut ekrana u prozoru u kojem se izvodi simulacija

Entity također sadrži parametre koje nije moguće mijenjati nakon što se stvori objekt, a to su *rotatingSpeed* i *forwardSpeed* koje predstavljaju brzinu rotacije ili promjene kuta u kojem se jedinka kreće te brzinu kretanja naprijed. Za postavljanje njihovih vrijednosti koristimo konstruktor razreda *Entity*:

```
public Entity(Vector2 position, float angle, double
energy, double rotatingSpeed, double forwardSpeed)
```

Tri su metode kretanja dostupne svakom objektima *Entity*:

1. `RotateRight(double input)` – povećava vrijednost *AngleOfMovement* za vrijednost umnoška *rotatingSpeed* i *input*
2. `RotateLeft(double input)` – umanjuje vrijednost *AngleOfMovement* za vrijednost umnoška *rotatingSpeed* i *input*
3. `Move(double input)`- na pseudokodu slike 5. dana je implementacija *Move* metode, koja ovisno o orijentaciji jedinke izračunava njene koordinate u sljedećoj iteraciji simulacije.

```
funkcija Move(double input)
EntityPosition.X = cos(AngleOfMovement)*forwardSpeed*input
EntityPosition.Y = sin(AngleOfMovement)*forwardSpeed*input
kraj
```

Slika 5 Pseudokod funkcije kretanja

Metoda koja poziva navedene metode kretanja jest funkcija *ThinkAndAct* sa slike 6. koja prima parametre ulaza iz simulacije tj. niz brojeva *double* vrijednosti. Ti ulazi se prosljeđuju internom objektu razreda *NeuralNetwork* i njemu dostupnoj metodi *CalculateOutput* te se rezultati uzimaju kao ulazi za svaku od tri metode.

```
funkcija ThinkAndAct(lista inputs)
lista outputs = NeuronskaMreža.IzračunajIzlaz(inputs)
FunkcijeKretanja(outputs)
kraj
```

Slika 6 Pseudokod funkcije koja upravlja kretanjem jedinke

3.3.6. Razred „Simulation“

Ovo je najveći razred u implementaciji, te on nasljeđuje razred *Game* već spomenutog XNA Framework-a. U njemu se nalaze *override* metode, ili bolje rečeno implementacije metoda razreda *Game*:

1. *Initialize*
2. *LoadContent*
3. *Update*
4. *Draw*

Dodatno su osim navedenih implementirane pomoćne metode za prikaz i izračune simulacije te spremanje rješenja.

1. Initialize postavlja parametre simulacije na njihove početne vrijednosti, te određuje nativnu rezoluciju ekrana i postavlja način rada programa preko cijelog ekrana.

2. LoadContent učitava teksture i fontove koji se koriste prilikom iscrtavanja grafike koje smo prethodno učitali u *Content* dio projekta te su time dostupni *Content Manager-u* unutar koda.

3. Update metoda razreda *Simulation* je implementacija naslijeđene metode *Update* od XNA Framework razreda *Game*, i sadrži implementaciju izračuna simulacije. Svakim pozivanjem *Update* metode, provjerava se potencijalni unos s tipkovnice i osvježi stanje objektima.

```
funkcija Update()
trenutnoStanjeTipkovnice = Tipkovnica.DohvatiStanje()
ObradiUnos(trenutnoStanjeTipkovnice)
ako (!pauziranaSimulacija)
{
 za svaku jedinku iz jedinki
  IzračunajKonzumiranjeEnergije(jedinka)
  jedinka.Pozicija = PodesiPoziciju(jedinka.Pozicija)
  jedinka.PozoviFunkcijeKretanja(IzračunajUlaze(jedinka))
 kraj
 ako (jedinke.BrojJedinki > 2)
 {
  jedinke.Sortiraj()
  ako (jedinke.Prva.KoličinaKonzumiraneEnergije > maxEnerg)
  {
  pauziranaSimulacija = true;
  }
 }
 VremenskaPotrošnjaEnergije()
 ProvjeriEnergijeHrane()
 ProvjeriEnergijeJedinki()
}
```

Slika 7 Pseudokod Update metode

Vidljivo je unutar pseudokoda na slici 7 kako *Update* metoda sadrži mnogo pomoćnih funkcija koje obavljaju različite poslove:

- `ObradiUnos(trenutnoStanjeTipkovnice)` – provjerava koje su tipke pritisnute, te osigurava interakciju korisnika s simulacijom. Također sadrži mogućnost pozivanja *Debug Terminal*-a koji ima mogućnost pozivanja i manipulacije varijabli i funkcija unutar razreda *Simulation*.
- `IzračunajKonzumiranjeEnergije(jedinka)` – prijenos energije s hrane na jedinku, ili jedinke na jedinku je moguć samo ako su uvjeti zadovoljeni, a to je: međusobna udaljenost dva objekta na ekranu, te kut pod kojim jedinka gleda na drugi objekt.
- `PodesiPoziciju(jedinka.Pozicija)` – postoje četiri granična slučaja kad jedinka nije u mogućnosti se kretati dalje u istom smjeru, a to su stranice aplikacije van kojih jedinka može izaći. Kako bi se negirao ovaj efekt, jedinka se prilikom prolaska određene stranice stvara na nasuprotnoj strani.
- `jedinka.PozoviFunkcijeKretanja(IzračunajUlaze(jedinka))` –*PozoviFunckijeKretanja* iz pseudokoda je zapravo metoda *ThinkAndAct* razreda *Entity* koja kao parametar prima listu brojeva koju joj osigurava druga funkcija unutar programa, ovdje označena kao *IzračunajUlaze.*
- `jedinke.Sortiraj()` – jedinke se sortiraju po količini konzumirane energije, tako da indeks jedinke unutar liste jedinki označava njezinu dobrotu.
- `ProvjeriEnergijeHrane()` – broj objekata *Food* je konstantan, pa je potrebno proći kroz njihovu listu, te stvoriti nove ako je potrebno.
- `ProvjeriEnergijeJedinki()` – broj objekata *Entity* je konstantan, i u slučaju da je neka jedinka ostala bez energije zbog funkcije *VremenskaPotrošnjaEnergije,* potrebno je izbaciti iz simulacije te stvoriti novi objekt *Entity.* Roditelji nove jedinke su prve dvije jedinke u trenutnoj populaciji.

Jedna od važnijih funkcija unutar pseudokoda sa slike 6. jest *IzračunajUlaze,* koja unutar izvornog koda implementacije nosi naziv *CalculateInputs.* Upravo njene povratne vrijednosti treba genetski algoritam optimizirati tako da pronađe pravi odziv na njih. tj. nađe pravi skup vrijednosti za težinsku matricu neuronske mreže koja za ove ulaze optimalno poziva funkcije dostupne jedinki.

```
funkcija IzračunajUlaze(jedinka)
lista rezultati = nova lista
lista ulaziOdHrane = nova lista
lista ulaziOdJedinki = nova lista
za svako oko iz očiju
 za svaku biljku iz hrane
  ako (udaljenost(jedinka,biljka) < radijusVidljivosti)
  { ako (kutIzmeđu(oko,biljka) < vidljiviKut)
   koeficijent = radijusVidljivosti – udaljenost
   sumaEnergijeOdHrane += biljka.Energija*koeficijent
  }
 kraj
 ulaziOdHrane.Dodaj(sumaHrane)
 za svaki entitet iz jedinki
  ako (udaljenost(jedinka,entitet) < radijusVidljivosti)
  { ako (kutIzmeđu(oko,entitet) < vidljiviKut)
    koeficijent = radijusVidljivosti – udaljenost
    sumaEnergijeOdJedinki += entitet.Energija*koeficijent
  }
 kraj
 ulaziOdJedinki.Dodaj(suma)
kraj
najvećaVrijednostHrane = max(ulaziOdHrane)
najvećaVrijednostJedinki = max(ulaziOdJedinki)
za sve ulazeOdHrane
 ulazOdHrane/najvećaVrijednostHrane kraj
za sve ulazeOdJedinki
ulazOdJedinki/najvećaVrijednostJedinki kraj
rezultat.Dodaj(ulaziOdHrane)
rezultat.Dodaj(ulaziOdJedinki)
rezultat.Dodaj(IzračunajOpasnost(jedinka))
vrati rezultat(RazinaEnergije(jedinka))
```

Slika 8 Pseudokod funkcije za izračun ulaza neuronskim mrežama

Pseudokod funkcije *IzračunajUlaze* vidljiv je na slici 8 Ova funkcija izračunava ulaze koji se prosljeđuju neuronskoj mreži. Kako bi dobili vrijednosti, funkcija određuje koji objekti *Food* i *Entity* su vidljivi kojem 'oku' jedinke:

- Lijevo oko
- Desno oko
- Središnje oko

Svako oko sadrži sumu skaliranih vrijednosti energije i relativne udaljenosti za pojedine objekte. Budući da se pri izračunu posebno gledaju objekti *Food* i *Entity*, imamo 3 ulaza za *Food* , i 3 ulaza za *Entity* objekte. Ove dvije skupine ulaza se skaliraju na vrijednosti od 0 do 1 tako da se podjele s najvećom sumom unutar skupine. Funkcija *IzračunajUlaze* implementira i izračun ulaza energije jedinke, te izračun udaljenosti najbliže druge jedinke.

Slika 9 Prikaz kromosoma

4. Draw metoda razreda *Simulation* prolazi kroz liste objekata *Food* i *Entity* te iscrtava na ekranu pripadajuće teksture s obzirom na lokaciju pojedinih objekata (Slika 9). Prilikom iscrtavanja tekstura *Food* objekata, veličina pojedine teksture se skalira s obzirom na količinu energije koju objekt posjeduje. *Entity* objekti imaju i svojstvo kuta, tj. varijablu *AngleOfMovement* pa prilikom iscrtavanja rotiramo teksturu. *Draw* metoda poziva pomoćne funkcije za izračun količine crvene, zelene i plave boje pri prikazu teksture *Entity* objekta u ovisnosti o količini dostupne mu energije i raspodjele konzumirane energije od objekata *Food* i *Entity*, te prikaz kromosoma najbolje jedinke vidljivo na slici 9.

4. Rezultati

4.1. Parametri

Ispitivanje se izvodi nad određenim skupom parametara unutar simulacije koji su od posebnog interesa, a to su parametri neuronskih mreža i genetskog algoritma. Svi parametri se nalaze unutar razreda *Simulation*, te im je moguće postaviti inicijalne vrijednosti u *Initialization* metodi ili tijekom izvođenja simulacije pozivom *XNA Debug Terminal*-a.

Parametri koje ispitujemo:

- `int geneLength` – duljina pojedinog gena u kromosomu
- `double mutationRate` – vjerojatnost mutacije pojedinog bita binarnog zapisa kromosoma
- `double neuralNetworkMaxWeight` – gornja vrijednost na koju se skalira *double* vrijednost prilikom pretvaranja binarnog zapisa u broj
- `double neuralNetworkMinWeight` – donja vrijednost na koju se skalira *double* vrijednost prilikom pretvaranja binarnog zapisa u broj
- `List<int> nnDesign` – dizajn umjetne neuronske mreže koja upravlja pojedinom jedinkom, brojevi u listi definiraju broj neurona u pojedinim slojevima

Parametri koji utječu na uvjete simulacije ne ulaze u ispitivanje te imaju početne vrijednosti zadane unutar programa:

- `graphics.PreferredBackBufferWidth = 1024` – broj točki za širinu prikaza
- `graphics.PreferredBackBufferHeight = 768` – broj točki za visinu prikaza
- `entityEnergy = 250` – energija jedinke pri stvaranju
- `entityMaxEnergy = 2000` – maksimalna energija koju jedinka može.doseći prije kažnjavanja.
- `entityOverFeedingPenalty = 1000` – kazna za prejedanje

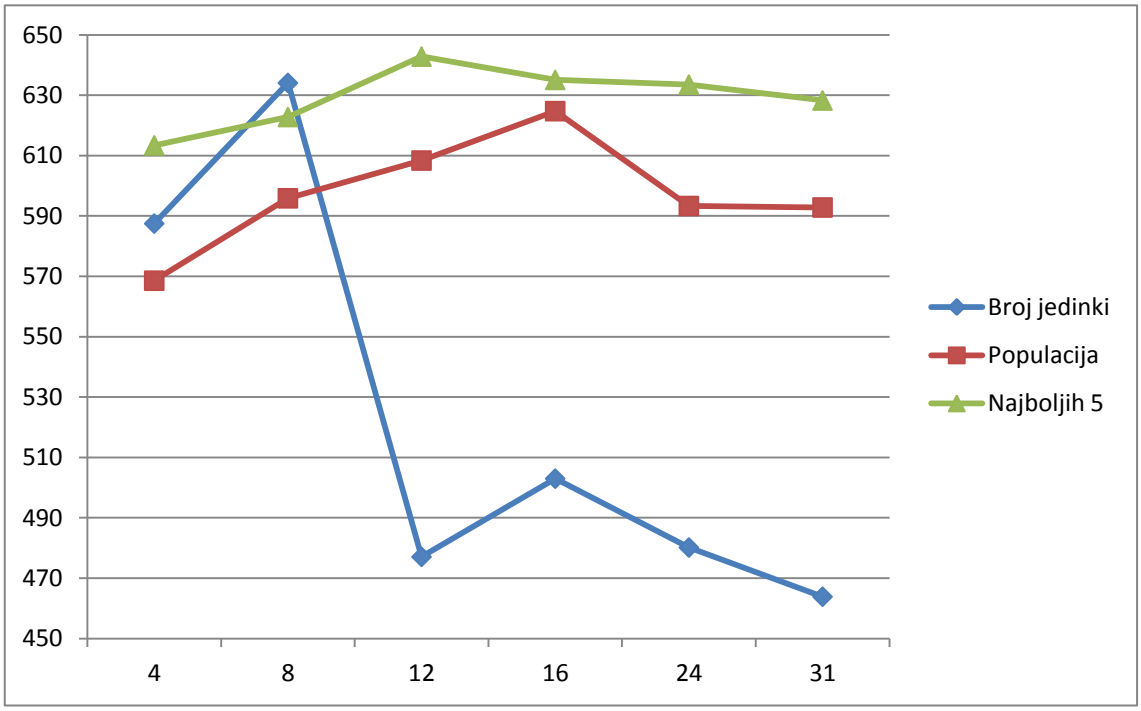
- `entityForwardSpeed = 8` – brzina kretanja u pikselima (od eng. *pixel*), jedinka se pri svakom koraku simulacije može maksimalno pomaknuti za ovaj iznos
- `entityRotateSpeed = 25` – brzina rotacije u stupnjevima
- `entityTimeEnergyLoss = 0.95` – količina energije koju jedinke gube svaki korak simulacije
- `visibilityRange = 300` – udaljenost na koju jedinka vidi druge jedinke
- `visibilityAngle = 35` – kut gledanja u stupnjevima za svako oko koje jedinka posjeduje
- `foodEatingSpeed = 10` – brzina konzumacije energije iz hrane u jednom koraku simulacije
- `entityEatingSpeed = 35` – brzina konzumacije energije iz drugih jedinki u jednom koraku simulacije
- `foodNumber = 40` – broj objekata *Food* u simulaciji
- `foodEnergy = 100` – maksimalna količina energije za pojedini objekt *Food*
- `foodEnergyLoss = 0.1` – količina energije koju objekti *Food* gube svaki korak simulacije

4.2. Uvjeti zaustavljanja

Simulacija se zaustavlja ako neka jedinka iz populacije konzumira količinu energije koja je 70 puta veća od vrijednosti varijable `entityMaxEnergy`.

4.3. Prikaz rezultata

Svako ispitivanje se izvodi 10 puta, a u prikazu rezultata se uzima prosjek praćenih vrijednosti. Prilikom ispitivanja cilj je pronaći parametre koji daju najbolji prosjek konzumacije energije cijele populacije uz što manji broj jedinki. Budući da bolji prosjek populacije ujedno znači i teže uvjete preživljavanja svakoj pojedinačnoj jedinki, i općenito prilagođenije jedinke, naglasak prilikom odabira optimalnih parametara je prosjek populacije i prosjek najboljih 5 jedinki. U prikazu rezultata na slikama 10.-14. i tablici 1. prosjek populacije je smanjen za faktor 100 a prosjek 5 najboljih za faktor 200 zbog preglednosti.

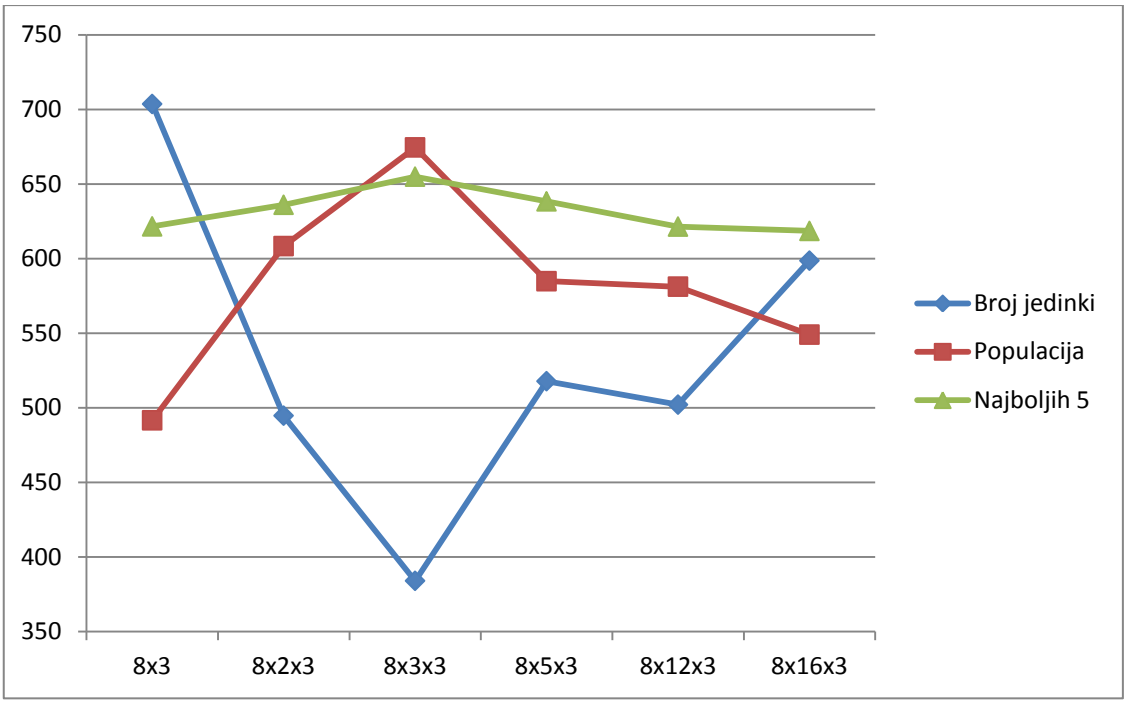


Slika 10 Rezultati za duljinu binarnog zapisa

Na slici 10 je prikazana ovisnost broja stvorenih jedinki i prosječne konzumirane energije o duljini binarnog zapisa pojedinog gena iz kromosoma. Broj stvorenih jedinki je najmanji za duljinu binarnog zapisa od 12 mjesta, dok prosječna konzumirana energija doseže najbolje rezultate za vrijednost 16.

Vrijednost ostalih parametara prilikom ispitivanja duljine binarnog zapisa:

- `mutationRate = 0.05`
- `neuralNetworkMaxWeight = 4`
- `neuralNetworkMinWeight = -4`
- `nnDesign = {8, 5, 3}`

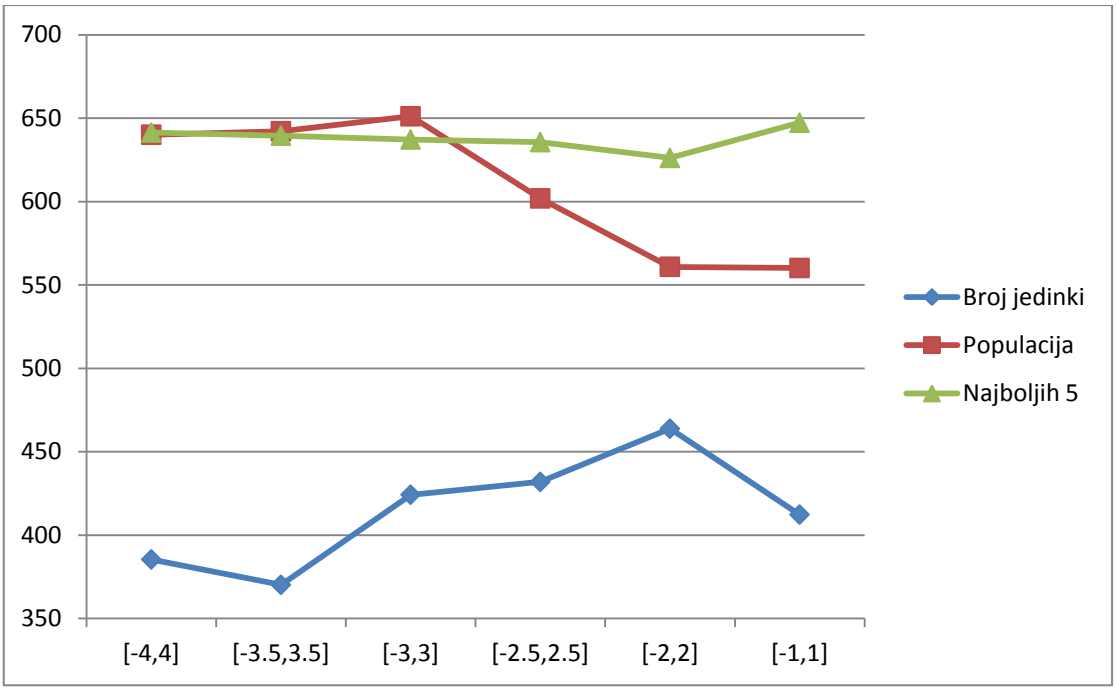


Slika 11 Rezultati za strukturu neuronske mreže

Na slici 11 prikazana je ovisnost prosječne konzumirane energije i broja novostvorenih jedinki o strukturi neuronske mreže. Rezultati ukazuju da je poželjna struktura neuronske mreže koja ima međusloj od 3 neurona.

Vrijednost ostalih parametara prilikom ispitivanja strukture neuronske mreže:

- `mutationRate = 0.05`
- `neuralNetworkMaxWeight = 4`
- `neuralNetworkMinWeight = -4`
- `geneLength = 16`

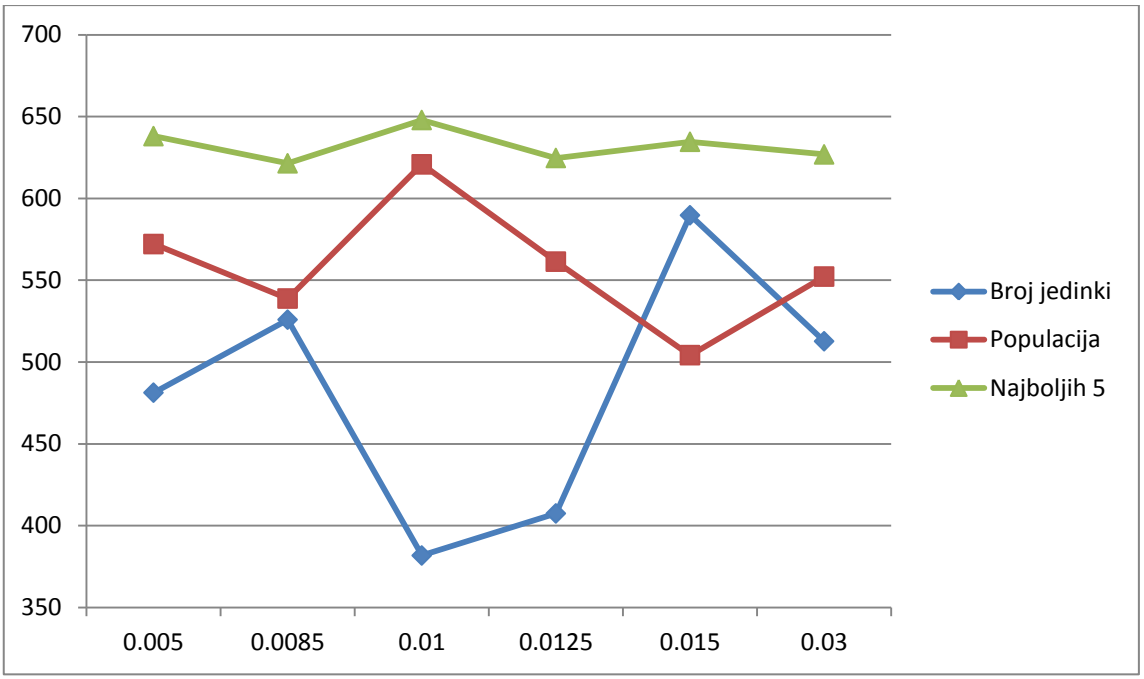


Slika 12 Rezultati za interval vrijednosti težinske matrice

Slika 12 prikazuje ovisnost rezultata o intervalu na koju se skalira pojedina vrijednost u težinskoj matrici neuronske mreže. Najbolji rezultati se postižu interval [-3,3].

Vrijednost ostalih parametara prilikom ispitivanja intervala težinske matrice:

- `mutationRate = 0.05`
- `nnDesign = {8, 3, 3}`
- `geneLength = 16`



Slika 13 Rezultati za različite stope mutacije

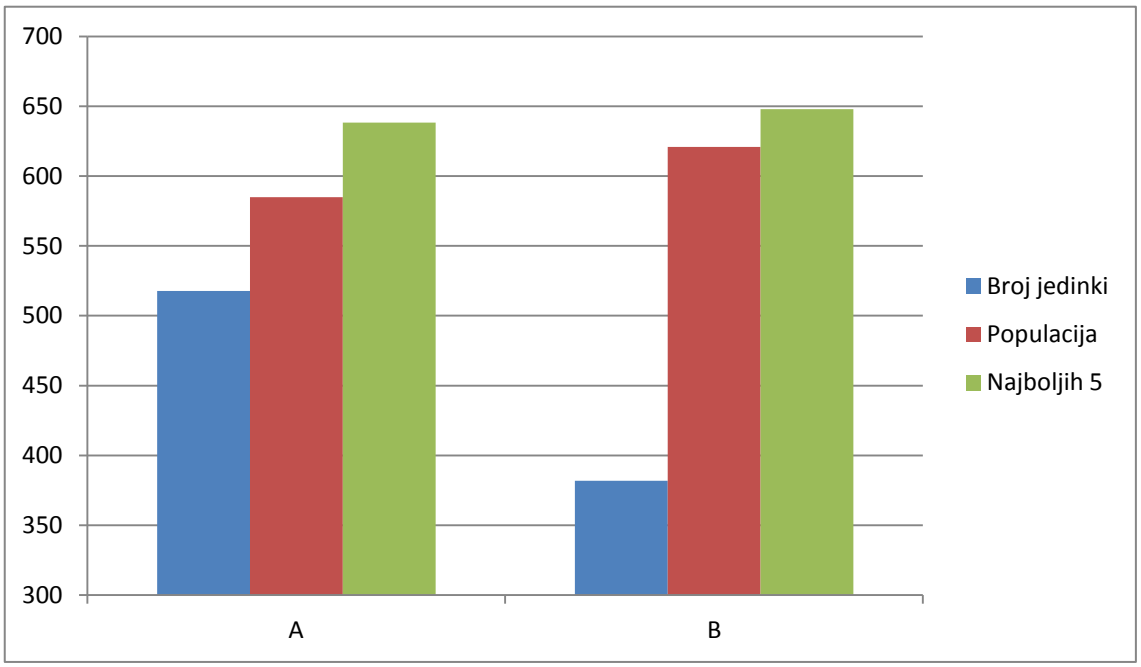
Na slici 13 prikazana je ovisnost rezultata o stopi mutacije. Najbolji rezultati su postignuti upotrebom stope mutacije od 1%.

Vrijednost ostalih parametara prilikom ispitivanja stope mutacije:

- `nnDesign = {8, 3, 3}`
- `neuralNetworkMaxWeight = 3`
- `neuralNetworkMinWeight = -3`
- `geneLength = 16`

Tablica 1 Usporedba rezultata

Broj jedinki		Populacija		Najboljih 5	
A	B	A	B	A	B
462	339	61389	57515	123675	127198
644	366	54068	57039	133642	135068
335	270	75810	69056	135429	123781
657	231	55713	65849	126870	128307
421	506	56384	66026	129185	132391
402	383	69965	59456	135255	122126
520	524	55174	69491	122548	138058
472	382	58164	64806	115054	132638
325	284	63102	61103	129572	131068
940	533	35050	50460	125456	125157
517,8	381,8	584,819	620,801	638,343	647,896



Slika 14 Usporedba rezultata

U tablici 1 dana je usporedba dva skupa parametara i pojedinačni rezultati ispitivanja te su prosječni rezultati grafički prikazani na slici 14 Prvi skup parametara, sa oznakom „A“ odgovara početnim parametrima i sadrži sljedeće vrijednosti:

- `geneLength = 16`
- `mutationRate = 0.05`
- `neuralNetworkMaxWeight = 4`
- `neuralNetworkMinWeight = -4`
- `nnDesign = {8, 5, 3}`

Drugi skup parametara s oznakom „B“ je dobiven koristeći rezultate ispitivanja i sadrži sljedeće vrijednosti:

- `geneLength = 16`
- `mutationRate = 0.001`
- `neuralNetworkMaxWeight = 3`
- `neuralNetworkMinWeight = -3`
- `nnDesign = {8, 3, 3}`

Broj novih jedinki potrebnih za dostizanje uvjeta zaustavljanja je smanjen, dok je ukupni prosjek populacije veći.

Zaključak

Cilj ovog rada je optimizacija neuronskih mreža unutar simulacijskog okruženja u kojem se modeli jednostavnih organizama bore za preživljavanje. Njihovim akcijama upravlja umjetna neuronska mreža, a njihovo razmnožavanje te prijenos genetskog zapisa o neuronskoj mreži nadgleda genetski algoritam.

Ispitivanja nad skupom parametara utvrdila su pozitivnu korelaciju boljih rezultata sa precizno odabranom strukturom neuronske mreže i stope mutacije genetskog algoritma.

Interval vrijednosti koje može poprimiti težina veze između neurona tj. vrijednost u težinskoj matrici pokazuje ovisnost o odabranoj aktivacijskoj funkciji. Poželjno je imati interval koji može doseći većinu izlaznih vrijednosti aktivacijske funkcije.

Veći binarni zapis neuronske mreže općenito daje bržu konvergenciju, no prevelike vrijednosti utječu na kvalitetu dobivenih rješenja što je također karakteristika prerane konvergencije. Također postoje oscilacije konvergencije unutar ispitivanja koje je moguće pripisati početnoj nasumično stvorenoj populaciji koja je relativno mala zbog simulacijskog okruženja.

Literatura

[1] SIVANANDAM,S.N.; DEEPA, S.N. *Introduction to Genetic Algorithms*. Berlin: Springer, 2008.

[2] GOLUB, M. *Genetski algoritam* ,Zagreb, 2004

[3] HEATON,J. *Introduction to Neural Networks for C#*. St. Louis: Heaton Research Inc., 2008.

[4] Neuron, http://hr.wikipedia.org/wiki/Neuron 28.5.2010

[5] Multilayer perceptron, http://en.wikipedia.org/wiki/Multilayer_perceptron 28.5.2010

[6] MILLER, T.; JOHNSON D. *XNA Game Studio 4.0 Programming* Boston: Addison-Wesley, 2010.

Sažetak

Optimizacija neuronske mreže uz pomoć genetskog algoritma

U radu je istražena primjena genetskog algoritma za selekciju umjetnih neuronskih mreža koje upravljaju modelima organizama u simuliranom okruženju. Korištene su slojevite neuronske mreže čiji neuroni imaju nelinearne aktivacijske funkcije. Cilj simulacije je postići ravnotežu kod jednog ili više organizama unutar populacije koji mogu preživjeti zadane uvjete dug period vremena. Uvjeti zahtijevaju konzumiranje energije koju dobivaju tako da je oduzimaju od biljki ili drugih organizama. Rezultati pokazuju velik utjecaj skrivenog sloja neuronske mreže i stope mutacije genetskog algoritma na rezultate.

Ključne riječi: genetski algoritmi, neuronske mreže, simulacija jednostavnih organizama

Summary

Neural network inference with genetic algorithms

This thesis investigates the application of genetic algorithm for selection of artificial neural networks that govern the model organisms in simulated environment. Layered neural networks whose neurons have nonlinear activation functions were used. The goal of simulation is to achieve balance in one or several organisms within population that can survive conditions for longer period of time. Environment demands energy consumption they acquire by stealing it from plants or other organisms. The results show large influence of hidden layer in neural network and mutation rate of genetic algorithm.

Key words: genetic algorithms, neural networks, simulation of simple organisms

Privitak

Instalacija programske podrške

1. Za izvođenje simulacije potrebno je imati instalirane .NET Framework 4.0 te XNA Framework 4.0.
2. Cijeli sadržaj programske podrške kopirati na računalo i pokrenuti izvršnu datoteku *ZavrsniRad.exe*.

Upute za korištenje programske podrške

Dostupne su kratice na tipkovnici:

- *Escape* – izlaz iz programa
- „S“ – pokreće izvođenje simulacije
- „P“ – zaustavlja izvođenje simulacije
- „U“ – nastavlja izvođenje simulacije
- „C“ – uključuje prikaz kromosoma trenutno najbolje jedinke
- „H“ - isključuje prikaz kromosoma trenutno najbolje jedinke
- Tilda – poziva naredbeni redak koji omogućava tekstualni unos naredbi
- F1-F5 – mijenja brzinu izvođenja simulacije

Tekstualne naredbe:

- `Start(populationNumber, nnDesign, geneLength, mutationRate, neuralNetworkMinWeight, neuralNetworkMaxWeight)` - parametre je moguće mijenjati prilikom poziva funkcije npr. `Start(30, nnDesign, 8, 0.3, -3, 3)` ili zasebno: `nnDesign.Add(3)` ili `populationNumber = 30`
- `ResetSimulation()` – resetira simulaciju
- `Save(„name“)` – sprema trenutno stanje simulacije s nazivom „name“ u direktorij izvršne datoteke
- `Load(„name“, broj)` – učitava spremljeno stanje simulacije s nazivom „name“ . *Broj* je cjelobrojna vrijednost koja označava koliko jedinki želimo pozvati iz spremljene populacije.