Neuronske mreže s algoritmom evolucijskog računarstva

Rendulić, Bruno

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:561110

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bruno Rendulić

Neuronske mreže s algoritmom evolucijskog računarstva

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bruno Rendulić

Matični broj: 1709999330080

Studij: Poslovni sustavi

Neuronske mreže s algoritmom evolucijskog računarstva

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Nikola Ivković

Varaždin, rujan 2022.

Bruno Rendulić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu je objašnjeno funkcioniranje neuronskih mreža, genetskih algoritama kao podvrste algoritama evolucijskog računarstva, izgradnja jednostavne neuronske mreže, genetskog algoritma zvanog „Dawkinsova lasica“ te neuronske mreža koja koristi genetski algoritam za učenje rješavanja određenog zadatka. U teorijskim dijelovima je objašnjeno kako su neuronske mreže postavljene, funkcioniranje i učenje neuronskih mreža, što su to umjetni neuroni i kako oni ustvari funkcioniraju te je objašnjena izgradnja jednostavne neuronske mreže koja je napisana u C# jeziku koristeći program Visual Studio 2022. Ujedno je objašnjeno funkcioniranje algoritama evolutivnog računarstva te način rada i upotreba genetskog algoritma koji je potkrepljen primjerom Dawkinsonove lasice napisanim u C# jeziku. Naposljetku je objašnjena izgradnja neuronske mreže koja uči koristeći genetski algoritam u svrhe rješavanja logičke operacije ekskluzivno ILI. Cilj je bio pojasniti i istražiti kako algoritam evolucijskog računarstva utječe na neuronsku mrežu te kako uopće neuronska mreža funkcionira.

Ključne riječi: neuronske mreže; algoritmi evolutivnog računarstva; genetski algoritam; visual studio; C#; umjetni neuron;

Sadržaj

1. Uvod 1
2. O Neuronskim mrežama 2
2.1. Učenje neuronske mreže 4
2.2. Umjetni neuron 5
2.2.1. Kako radi umjetni neuron? 6
2.3. Izgradnja neuronske mreže 10
3. O algoritmima evolucijskog računarstva 19
3.1. Genetski algoritam 20
3.2. Dawkinsova lasica 22
4. Izgradnja neuronske mreže za rješavanje XILI uz pomoć genetskog algoritma 25
5. Zaključak 34
6. Popis slika 35
7. Literatura 35

1. Uvod

Tema ovog završnog rada je „Neuronske mreže s algoritmom evolucijskog računarstva“. Kroz rad se objašnjava što su to neuronske mreže, kako funkcioniraju te kako uče. Ujedno se objašnjava što su to algoritmi evolucijskog računarstva, njihovu primjenu i konkretni primjer za genetski algoritam kao podvrstu algoritma evolucijskog računarstva. Na kraju je prikazana izgradnja neuronske mreže koja koristi genetski algoritam u svrhu rješavanja logičke operacije XILI kako bi se dokazalo funkcioniranje neuronske mreže. Rad prikazuje korake u izradi jednostavne neuronske mreže koja rješava problem logičke operacije I (eng. AND), Dawnkinsonove lasice, genetskog algoritma koji dokazuje uspješnu implementaciju biološkog ekvivalenta evolucije u računarstvo te neuronske mreže koja koristi sličan genetski algoritam kako bi naučila rješavati operaciju XILI (eng. XOR) pomoću programa Visual Studio 2022 koristeći C# jezik.

Često možemo čuti kako je umjetna inteligencija vrhunac računalnih znanosti, ali shvaćanje takvih umjetnih inteligencija obično nam prolazi preko glave. Mnogo ljudi uopće ne zna kako umjetna inteligencija funkcionira, odnosno uopće kakvih vrsta umjetne inteligencije postoji. Cilj rada je detaljno reprezentirati funkcioniranje neuronskih mreža i samih neurona unutar mreže, funkcioniranje genetskog algoritma kao podvrste algoritama evolucijskog računarstva te njegove moguće upotrebe i na kraju izgraditi neuronsku mrežu koja koristi genetski algoritam kako bi sama mogla rješavati određeni zadatak.

Ovu temu sam odabrao zbog fascinacije umjetne inteligencije i već prijašnjih pokušaja izrade umjetnih inteligencija kada sam bio mlađi. Kada sam započeo sa radom, shvatio sam kako je iza teme stoji ustvari velik broj novih informacija za koje nikad nisam čuo, te me to inspiriralo da svoje istraživanje podijelim sa ostalima.

Pri izradi završnog rada su korišteni izvori sa interneta i knjige „Umjetne neuronske mreže“ (Novaković B., Majetić D., Široki M., 1998.), „Introduction to Evolutionary Algorithms“ (Yu X., Gen M., 2010). te „Deep learning“ (Kelleher D. J., 2019.). Svi programi su izrađeni u aplikaciji Visual Studio 2022 i to u jeziku C#. Fotografije koje su korištene su preuzete s interneta, uzete kao slike zaslona rezultata te izrađene kao vlastiti rad u aplikaciji GIMP kako bi kvalitetnije bili reprezentirani neki od primjera.

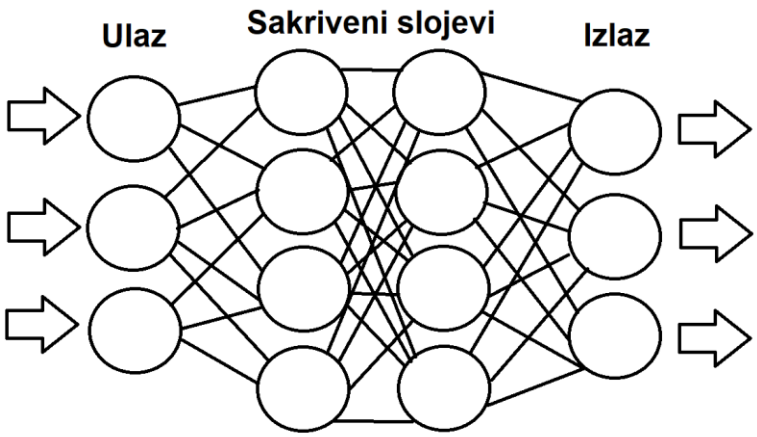
2. O Neuronskim mrežama

Umjetne neuronske mreže ili pojednostavljeno neuronske mreže je računski model inspiriran biološkim neuronskim mrežama mozga sa fokusom na ljudski mozak. Ljudski se mozak sastoji od velikog broja živčanih stanica, zvanih neuroni, koji su zaduženi za komunikaciju, te konačno odlučivanje o nekakvim tjelesnim radnjama, zaključivanju i drugim. Takva mreža neurona koji primaju signale te na jednu stranu procesiraju informaciju i donose odluke se naziva neuronska mreža. Neuronske mreže mogu naučiti modelirati odnose između ulaza i izlaza koji su nelinearni i složeni, izvoditi generalizacije i zaključke, otkriti skrivene odnose, obrasce i predviđanja, modelirati vrlo nestabilne podatke (kao što su podaci o financijskim ili vremenskim pojavama).

John D. Kelleher (2019. str. 67) je u svojoj knjizi sumirao rad mozga preko neuronskih mreža na tri točke:

- Mozak je sastavljan od velikog broja, koji se procjenjuje čak u milijardama, jednostavnih i međusobno povezanih jedinica koje se nazivaju neuroni.
- Funkcija mozga se može gledati kao obrada informacija koje su kodirane električnim signalima visokog ili niskog potencijala, odnosno aktivacijskih potencijala koji se šire neuronskom mrežom.
- Neuron prima skup podražaja od svojih prethodnika i preslikava ih u izlaz visokog ili niskog potencijala ili uopće ne šalje ikakav potencijal. Svi računski modeli imaju te značajke.

Slika 1 prikazuje jednostavnu neuronsku mrežu.



Slika 1: Jednostavna neuronska mreža (autorski rad)

Na prvi pogled ovo ne izgleda kao jednostavna mreža, ali je ustvari doista jednostavno za razumjeti.

Počevši od ulaznog sloja, možemo vidjeti kako je svaki on neurona, označen sa kružićem, spojen na više drugih neurona u sljedećem sloju. Ulazne neurone možemo gledati kao pokretače koji dobivaju određene parametre za rješavanje problema, odnosno podatke. Ti ulazni neuroni šalju svoj potencijal, odnosno koliko su jako aktivirani u ulaze neurona u skrivenom sloju.

Nekom određenom funkcijom određuje se jačina signala koja će se slati dalje u ostale povezane neurone u drugom sloju te će se takav proces ponavljati dok se ne dođe do izlaznih neurona. Izlazni sloj je zadužen za odlučivanje te davanje rješenja za određen problem.

Na slici 1 je reprezentirana jednostavna mreža, a postoje i jednostavnije, ali i kompliciranije mreže te se u ljudskom mozgu nalazi na milijarde takvih mreža. Možemo zaključiti kako paralelno postavljen skup neurona gradi jedan sloj neuronske mreže. Neuronske mreže ovisno o broju tih slojeva mogu biti jednoslojne i višeslojne. Slika 1 je primjer višeslojne mreže jer ima ulazni i izlazni sloj te neki broj slojeva između njih, tj. skrivenih slojeva. Te skrivene slojeve možemo gledati kao tzv. „Crne kutije“, jer je veći pogled na procesiranje podataka naspram donošenju finalnih odluka. Kada je neuronska mreža povezana tako da signali putuju u jednom smjeru, najčešće od ulaza prema izlazu, tada takve mreže prema Novakoviću i sur. (1998, str. 6-7) nazivamo unaprijednim mrežama. Često su takve mreže napravljene za jednostavne odluke bez potrebe povratka na jedan od neurona koji donosi odluku. Kada imamo povratnu vezu, odnosno nekakvu povratnu petlju gdje smjer signala može varirati, tada takvu mrežu možemo nazvati povratna neuronska mreža.

Postojanje više vrsta neuronskih mreža se vidjeti u srži problema koje rješavaju. Neki problemi će moći biti riješeni samo unaprijednim mrežama, dok će se drugi moći rješavati povratnima te neki treći problemi kombinacijama. Kada imamo milijune takvih mreža, poput ljudskog mozga, pokrivamo veliku domenu problema koji se mogu riješiti, ali i rješenja do kojih možemo doći.

Novaković i sur. (1998, str. 7) ujedno navode kako neuronske mreže možemo diferencirati i po metodama koje koriste kako bi učile. Ako neuronska mreža uči vračajući se po vezama unatrag, odnosno popravljanjem jačina izlaznih signala, tada se one nazivaju i povratno propagirane mreže. Ujedno postoje i suprotno propagirane, ali i statističke neuronske mreže.

2.1. Učenje neuronske mreže

Ponekad izlazi neuronskih mreža mogu biti netočni, što znači da imaju određenu stopu pogreške, jer određeni ulazi mogu dati očekivane rezultate, dok ostali daju netočne. Poput čovjeka, neuronska mreža može učiti na svojim pogreškama. Učenje neuronske mreže možemo shvatiti kao popravljanje jačina signala koji određeni neuroni šalju kroz mrežu.

Novaković i sur. (1998, str. 7-8) razlikuju nadzirano i nenadzirano učenje. Nadzirano učenje se odvija uz pomoć nekog vanjskog subjekta, tj. „učitelja“ koji korigira neuronsku mrežu dok se ne dobiju željeni rezultati. Učitelj promatra ponašanje neuronske mreže od samog stvaranja mreže, gdje se postavlja određena struktura mreže, odnosno broj ulaza, skrivenih slojeva, izlaza, neurona u mreži i drugih. Razlika očekivanih izlaza i stvarnih izlaza se naziva stopa pogreške, a ona se koristi za računanje novih težina i aktivacijskih potencijala neurona te se taj proces ponavlja kroz iteracije dokle god se stop pogrešaka ne umanji na prihvatljive razine.

Kod nenadziranog učenja, ne koristi se neki vanjski subjekt, već se mreža sama organizira. Razvijanje ovakvih neuronskih mreža je vrlo nejasno. Nenadzirana mreža pokušava oponašati podatke koje je dobila i koristi stopu pogreške u oponašanom izlazu da se ispravi. Učenje bez nadzora može se usporediti s načinom na koji djeca uče o svijetu bez uvida u nadzor odraslih. Nitko ne uči djecu da budu iznenađena i znatiželjna u vezi vrste životinja koju nikada prije nisu vidjeli.

Postoji više metoda učenja neuronskih mreža, ali je najčešća metoda povratne propagacije. Neuronske mreže općenito obavljaju nadzirane zadatke učenja, gradeći znanje iz skupova podataka gdje se točan odgovor daje unaprijed. Mreže zatim uče prilagođavajući se kako bi same pronašle pravi odgovor, povećavajući točnost svojih predviđanja. Da bi to učinila, mreža uspoređuje početne rezultate s danim točnim odgovorom ili ciljem. Naposljetku, rezultati funkcije troška se zatim vraćaju na sve neurone i veze kako bi se prilagodile pristranosti i težine. (Frank La, 2019.)

Funkcija troška, odnosno funkcija gubitka modelira podatke o učenju. Cilj ove funkcije je minimizirati gubitak između predviđenih i ciljnih rezultata. „Hiperparametri su podešeni da minimiziraju prosječni gubitak - nalazimo težine, w^t i pristranosti, b , koje minimiziraju vrijednost J (prosječni gubitak).“ Formula za računanje prosječnog gubitka glasi:

$$J(w^t, b) = \frac{1}{m}\sum_{i=1}^{m} L(y'^i, y^i)$$

Gdje je m broj ulaza, a $L(y'^{i}, y^{i})$ statistička usporedba očekivanog izlaza i stvarnog. (Vishal Yathish, 2022.)

Metoda povratne propagacije je ključ za to kako neuronska mreža uči određeni zadatak. Svaki neuron u neuronskoj mreži uzima ulazne vrijednosti pomnožene s težinom kako bi predstavio snagu izlazne veze sa drugim neuronom. Metoda širenja unatrag otkriva ispravne težine koje treba primijeniti na neurone u neuronskoj mreži uspoređujući trenutne izlaze mreže sa željenim ili točnim izlazima.

Općenito, povratno širenje ima veze s ponovnim izračunavanjem ulaznih težina za umjetne neurone. Delta pravilo, odnosno delta učenje koristi razliku između ciljane aktivacije i stvarno dobivene aktivacije. Delta pravilo pomaže povratnoj propagaciji uspostavljajući veze između ulaza i izlaza pomoću slojeva umjetnih neurona. Pomoću funkcije linearne aktivacije prilagođavaju se mrežne veze. Drugi način da se objasni Delta pravilo jest da ono koristi funkciju pogreške za izvođenje učenja spuštanja prema pragovima. U osnovi u usporedbi stvarnog izlaza s ciljanim izlazom, delta pravilo pokušava pronaći podudaranje. Ako nema podudaranja, program vrši promjene. (Frank La, 2019.)

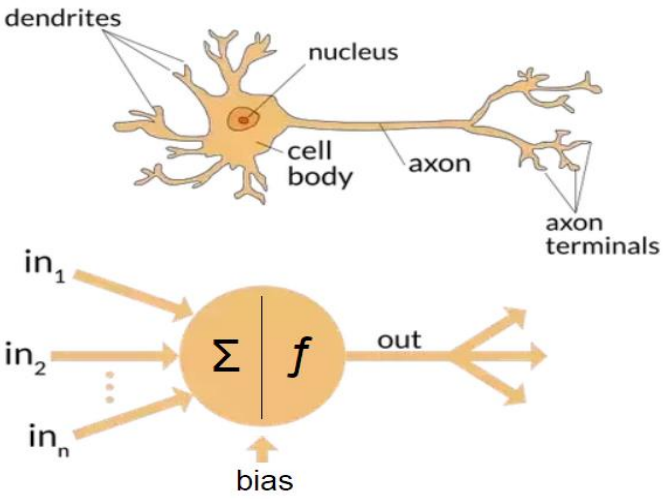
Stopa učenja (eng. Learning rate) se koristi tokom povratne propagacije u svrhe postavljanja količine promjena težine tokom ažuriranja. Stopa učenja kontrolira koliko brzo se model prilagođava problemu. Manje stope učenja zahtijevaju više iteracija učenja s obzirom na manje promjene u težinama pri svakom ažuriranju, dok veće stope učenja rezultiraju brzim promjenama i zahtijevaju manje iteracija učenja. Stopa učenja koja je prevelika može uzrokovati prebrzi dolazak modela do ispodprosječnog rješenja, dok stopa učenja koja je premala može uzrokovati zaglavljivanje procesa. Izazov treniranja neuronskih mreža uključuje pažljiv odabir stope učenja. Rakhecha A. (28. 5. 2019)

2.2. Umjetni neuron

Osnovni gradivni blok neuronske mreže je sami neuron. Za ove neurone se može reći da su dizajnirani s idejom oponašanja osnovnih funkcija biološkog neurona. Zbog lakšeg shvaćanja umjetnih neuronskih mreža, dobro je prvo shvatiti kako su uopće neuroni građeni te kako oni uopće funkcioniraju.

Neuroni su strukturirani u tri dijela: stanično tijelo, skupovi vlakna zvanih dendriti i jednog dugog vlakna zvanog akson. Dendriti služe kao ulazi signala u neuron, te su spojeni sa aksonima koji šalju signale drugih neurona u njih. Aksoni su izlazi iz neurona koji šalju signale u ostale neurone koji su spojeni na njih, ali samo ako stanično tijelo, sa dovoljno poticaja,

odluči poslati signal. Stanično tijelo služi kao okidač, odnosno određuje da li će se primljeni signal slati dalje. Naravno najbitniji dio samog neurona jest stanično tijelo koje, ako je signal koji je primljen, odnosno nekakav određeni podražaj dovoljno jak, šalje duž svog aksona električni impuls koji se zove aktivacijski potencijal. Aktivacijski potencijali se gledaju kao ulazi, ali kao i izlazi iz neurona. Ti izlazi nisu samo 0 ili 1, već mogu poprimit velik broj vrijednosti ovisno o vrsti aktivacijske funkcije. (Kelleher D. J. 2019. str. 65-67) Slika 2. prikazuje sličnost izgleda umjetnog neurona i pravog biološkog neurona. Možemo primijetiti kako u oba slučaja postoje izlazne i ulazne veze te stanično tijelo.



Slika 2: Biological and artificial neuron (izvor: quora.com, 2018.)

2.2.1. Kako radi umjetni neuron?

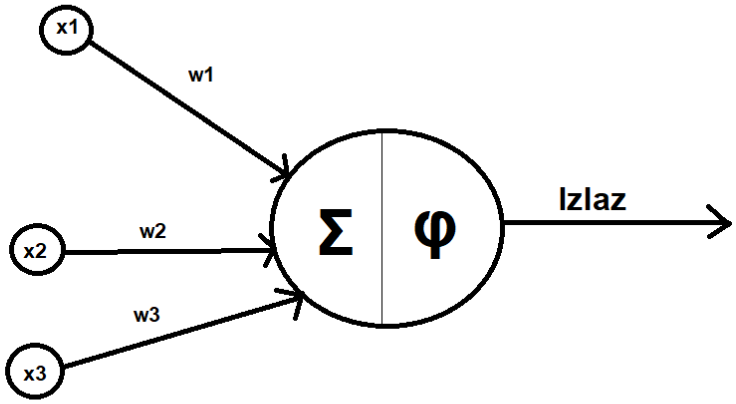
Već naveden aktivacijski potencijal može biti pozitivan ili negativan, odnosno visok ili niski, a kod novijih neuronskih mreža, čak i funkcija, odnosno varijabilan aktivacijski potencijal. Kada je aktivacijski potencijal jednak nuli onda veza neurona sa okolinom ne postoji. Takva se veza u crtanju sheme neurona tada ne crta jer neuron nije spojen. Aktivacijske potencijale možemo sagledavati kao sinapse pravog neurona u smislu da povezuje izlaze iz okoline neurona, odnosno aksone ostalih neurona spojenih na njega sa ulazima tj. dendritima. Novaković i sur. (1998, str. 5) navode kako intenzitet ulazne veze ovisi o sumiranoj težini ulaza koji se odvija u staničnom tijelu, a naziva se sumator. Određivanje ulaza i izlaza ustvari se provodi kroz dva koraka. Prvi je korak računanje težinskog zbroja ulaza, odnosno vrijednosti sumatora, a drugi korak uključuje preslikavanje sumirane vrijednosti u finalnu izlaznu funkciju i vrijednost neurona.

Drugi korak je ustvari aktivacijska funkcija koja može biti različitih vrsta. Funkcija može biti jednostavna poput odluke o početku pisanja završnog rada ili nešto kompliciranije. Funkcija se može odrediti u procesu dizajniranja neurona. Dobro je napomenuti kako aktivacije funkcije mogu biti linearne ili nelinearne.

Linearne funkcije množe vrijednost sumiranih težina sumatora sa nakakvim određenim faktorom iz aktivacijske funkcije te se tako dobiva izlaz iz neurona. Nelinearne funkcije su nešto kompliciranije, u smislu da mogu poprimiti velik broj različitih oblika. Novaković i sur. (1998, str. 5-6) ističu kako se često koriste funkcije praga, osjetljivosti, sigmodalne, hiperbolične, harmoničke funkcije te je česta pojava i trigonometrijskih funkcija.

Najčešći oblik ovih aktivacijskih funkcija u današnjici je nelinearna funkcija. Kako bi razumjeli zašto je to slučaj, možemo si aktivacijske funkcije. Svaki neuron u mreži rješava samo jedan dio većeg problema, odnosno glavni se problem rješava kombiniranjem pojedinačnih rješenja. postaviti pitanje: Kako izgleda neuronska mreža? Kako je već prije navedeno, neuroni su osnovni temelj neuronskih mreža i stoga su temelj mrežom definiranog preslikavanja. „*Cjelovito preslikavanje ulaza u izlaze čini slijed takvih preslikavanja svojstvenih pojedinačnim neuronima mreže*.“ (Kelleher D. J. 2019. str. 77) Kada bi nam cijela neuronska mreža koristila linearne funkcije, a pogled je na nelinearne odnose koji se puno češće javljaju u svijetu, rezultati funkcija bili bi neprecizni. Jednostavnost modela uzrokuje komplikaciju sa podatcima i rezultatima. Načelno, korištenje nelinearnih funkcija kao aktivacijskih funkcija omogućuje neuronskim mrežama da uče nelinearno preslikavanje ulaza u izlaze. Aktivacijske funkcije ne moraju biti iste. Različiti. neuroni mogu koristiti različite aktivacijske funkcije, ali je

česti slučaj da će neuroni u istom sloju biti i istog tipa, odnosno imati slične, ako ne i iste. Slika 3 prikazuje presjek umjetnog neurona.



Slika 3: Presjek umjetnog neurona (autorski rad)

Neuron na slici prima određene signale iz njegovih prethodnika (x_1,x_2,x_3…x_n), a broj ulaznih priključaka može biti n. Svaki od ulaza prima i određenu težinu (w_1, w_2, w_3… w_n) aktivacijskog potencijala. Važno je napomenuti kako težina i aktivacijski potencijal nisu ista stvar. Aktivacijski potencijal x se može sagledati kao već postojeće mišljenje, odnosno određena jačina koja će se poslati iz neurona. Svaki neuron ima neku određenu jačinu sa kojom započinje, a to možemo sagledati kao pristranost neurona da je njegovom aktivacijom njegov dio rješenja točan, a prava točnost nije bitna. Težina na drugu ruku jest koliko će biti pojačan taj signal koji ulazi u drugi neuron.

Računanje ulaza, odnosno ukupne težine koja se događa u sumatoru a uključuje množenje ulaznih težina i zbrajanje dobivenih vrijednosti se može matematički zapisati na sljedeći način:

$$Z = (x_1 * w_1) + (x_2 * w_2) + \cdots + (x_n * w_n)$$

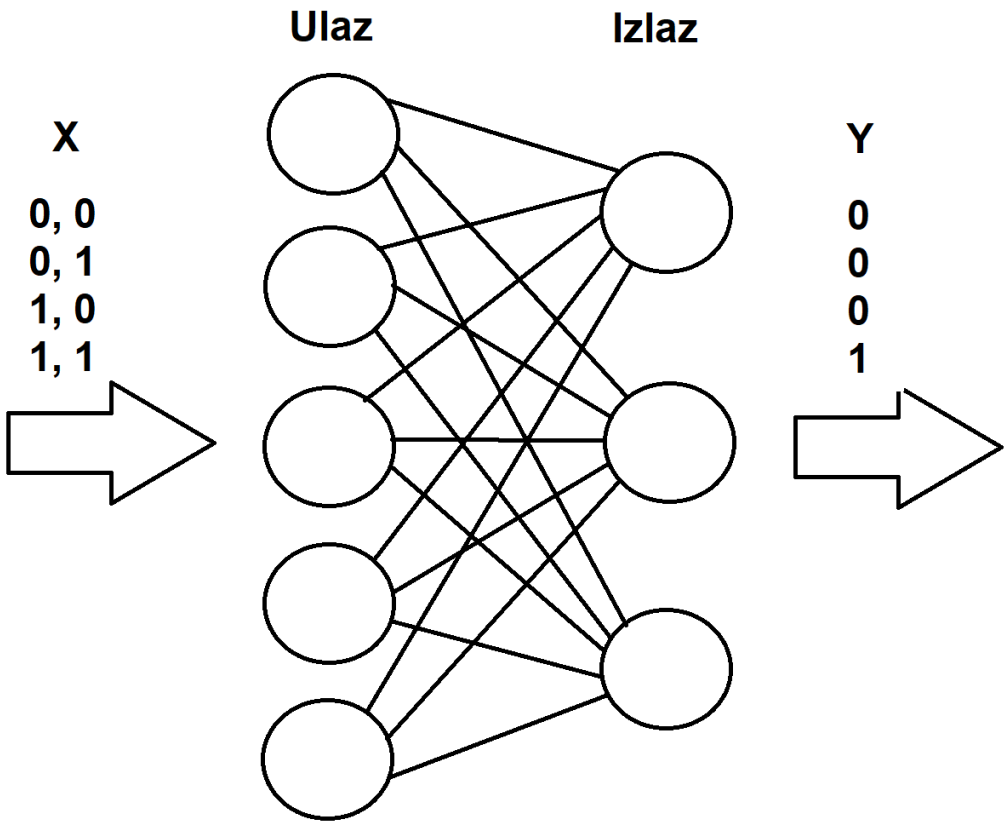
Sažetije funkciju možemo definirati kao: $Z = \sum_{i=1}^{n} x_i * w_i$

Sada možemo definirati ulazne parametre i vidjeti kako točno dolazi do odluka. Recimo da su Aktivacijski potencijali $x_1 = 0.44, x_2 = 0.12, x_3 = 0.86$, a težine $w_1 = 0.2, w_2 = 0.4, w_3 = 0.33$. Izračunati zbroj u sumatoru bi bio $Z = 0.2838$. Trenutno nam ovaj broj nema nikakvog značenja, ali zato prelazimo na aktivacijsku funkciju. (Kelleher D. J. 2019. str. 72)

U aktivacijskoj funkciji je određen nekakav okidač, odnosno prag aktivacije. Možemo uzeti za primjer funkciju osjetljivosti i reći da je okidač broj veći od 0.24, te bi se u našem slučaju neuron aktivirao. Naravno postoji mogućnost da se u mreži ovaj neuron nije trebao aktivirati, stoga se povratnom propagacijom popravljaju ulazne težine i aktivacijski potencijali, dokle god se ne ostvari očekivani rezultat, odnosno dok se dovoljno ne smanji stopa pogreške. Naravno ovi popravci se neće odvijati samo na jednom neuronu, već na svim neuronima koji su povezani od ulaza do izlaza, osim ako nije utvrđeno da određeni neuron ne utječe na povećanje ili smanjenje stope pogrešaka.

2.3. Izgradnja neuronske mreže

U svrhe pojašnjenja rada neuronske mreže izgrađena je jednostavna neuronska mreža sa ulaznim i izlaznim slojem, gdje ulazni sloj sadrži 5 neurona, a izlazni 3. Neuronska mreža je napravljena da rješava jednostavnu logičku operaciju AND, te da sama nauči koristiti istu. Neuronska mreža koristi povratnu propagaciju kako bi popravila svoje težine i aktivacijske potencijale. Na slici 4 je prikazana skica neuronske mreže. Ovakvu bi mrežu bilo moguće riješiti sa samo dva ulazna neurona i jednim izlaznim, ali u svrhe istraživanja korišteno ih je više kako bi se mogao vidjeti utjecaj broja neurona na konačno rješenje.



Slika 4: Skica neuronske mreže za logičku operaciju AND (autorski rad)

Aktivacijski potencijali su definirani u klasi ActivationPotencial tipa double i sadrže svoju vrijednost.

```
public class ActivationPotencial
{
    public double Value { get; set; }
}
```

Sljedeće su veze između neurona definirane u klasi Inputs. Kako bi se simulirao rad dendrita, potrebno je definirati aktivacijski potencijal koji prolazi kroz njega koji je tipa ActivationPotencial te samu težinu sinapsi između neurona tipa double zvanu Weight.

```
public class Input
{
    public ActivationPotencial Potencial { get; set; }
    public double Weight { get; set; }
    public Input()
    {
        Potencial = new ActivationPotencial();
    }
}
```

Tijek rada neurona je da on prima ulazne vrijednosti, radi ponderirani zbroj svih ulaznih signala u sumatoru iz svih ulaza i prosljeđuje ih u aktivacijsku funkciju koja je u ovom slučaju funkcija Activation sa jednostavnim pragom vrijednosti 1. Izlaz aktivacije dodijeljen je ActivationPotencial vrijednosti, odnosno aktivacijskom potencijalu neurona koji će putovati kroz akson neurona. Metoda UpdateWeights koja postavlja nove težine veza će se koristiti tokom učenja neuronske mreže.

```
public class Neuron
    {
        public List<Input> Inputs { get; set; }
        public ActivationPotencial ActivationPotencial { get; set; }

        public Neuron()
        {
            Inputs = new List<Input>();
            ActivationPotencial = new ActivationPotencial();
        }

        public void Trigger()
        {
            ActivationPotencial.Value = Sum();

            ActivationPotencial.Value =
Activation(ActivationPotencial.Value);
        }

        public void UpdateWeights(double new_weights)
        {
            foreach (var input in Inputs)
            {
                input.Weight = new_weights;
            }
        }

        private double Sum()
        {
            double Value = 0.0f;
            foreach (var input in Inputs)
            {
                Value += input.Potencial.Value * input.Weight;
            }

            return Value;
        }

        private double Activation(double input)
        {
            double threshold = 1;
            return input <= threshold ? 0 : threshold;
        }
    }
```

Kako bi uopće mogli koristiti neurone iz prijašnje klase potrebno je napraviti klasu za slojeve u neuronskoj mreži. Klasa Layers inicijalizira slojeve neuronske mreže te je ujedno zadužena za optimizaciju težina veza između neurona preko metode Optimize, a metodom Next ćemo okidati neurone u određenom sloju.

```
public class Layer
    {
        public List<Neuron> Neurons { get; set; }
        public string Name { get; set; }
        public double Weight { get; set; }
        public Layer(int count, double initialWeight, string name = "")
        {
            Neurons = new List<Neuron>();
            for (int i = 0; i < count; i++)
            {
                Neurons.Add(new Neuron());
            }

            Weight = initialWeight;
            Name = name;
        }

        public void Optimize(double learningRate, double delta)
        {
            Weight += learningRate * delta;
            foreach (var neuron in Neurons)
            {
                neuron.UpdateWeights(Weight);
            }
        }

        public void Next()
        {
            foreach (var neuron in Neurons)
            {
                neuron.Trigger();
            }
        }
    }
```

Sama neuronska mreža je definirana u klasi NeuralNetwork, a njene funkcije, zbog jednostavnijeg shvaćanja je najbolje zasebno objasniti. Dve najvažnije metode u ovoj klasi su metoda Build, koja povezuje slojeve neurona i time gradi neuronsku mrežu te metoda CreateNetwork, koja je ugniježđena u metodi Build te ona povezuje sve neurone iz različitih slojeva, dodjeljuje im aktivacijski potencijal i težinu veze.

```
public void Build()
        {
            int i = 0;
            foreach (var layer in Layers)
            {
                if (i >= Layers.Count - 1)
                {
                    break;
                }

                var nextLayer = Layers[i + 1];
                CreateNetwork(layer, nextLayer);

                i++;
            }
        }

  private void CreateNetwork(Layer connectingFrom, Layer connectingTo)
        {
            foreach (var from in connectingFrom.Neurons)
            {
                from.Inputs = new List<Input>();
                from.Inputs.Add(new Input());
            }

            foreach (var to in connectingTo.Neurons)
            {
                to.Inputs = new List<Input>();
                foreach (var from in connectingFrom.Neurons)
                {
                    to.Inputs.Add(new Input() { Potencial =
from.ActivationPotencial, Weight = connectingTo.Weight });
                }
            }
        }
```

Nakon što je neuronska mreža izgrađena i početni parametri su postavljeni, potrebno je izračunati vrijednost izlaza, odnosno izlaznog sloja, a to se odvija u metodi ComputeOutput koja aktivira sve neurone u mreži kako bi se iz njih mogli očitati izlazni podatci.

```
private void ComputeOutput()
        {
            bool first = true;
            foreach (var layer in Layers)
            {
                if (first)
                {
                    first = false;
                    continue;
                }

                layer.Next();
            }
        }
```

Nakon što su se svi neuroni aktivirali, inicijalne težine i aktivacijski potencijali su ostali isti. To je očekivano jer još nismo pokrenuli proces učenja neuronske mreže. Učenje neuronske mreže se odvija u metodi Train, a nju pozivamo nakon stvaranja ulaznih podataka X i očekivanih izlaza Y. Metoda započinje petljom kroz broj iteracija koje su ulazni parametar u metodu. U petlji se prvo pozivaju ulazni slojevi zbog inicijalizacije početnih podataka. Nakon što se inicijaliziraju ulazni slojevi, stvara se nova petlja koja se odvija onoliko puta koliko je ulaznih podataka, te se podatci preslikavaju u ulazni sloj. U petlji se ujedno pale neuroni te se prikupljaju izlazi. Nakon što se petlja izvrti, provjeravaju se izlazni rezultati sa očekivanim izlazima te se računa stopa pogreške, odnosno u ovome slučaju točnost izlaza. Tokom rada metode Train, ispisuje se točnost za svaku iteraciju, iako se ponekad za točnost koristi i naziv fitness (eng. spremnost) iako je taj term dosta vezan za genetiku, odnosno genetske algoritme.

```
        public void Train(Data X, Data Y, int iterations, double
learningRate = 0.1)
        {
            int iteration = 1;
            while (iterations >= iteration)
            {

                var inputLayer = Layers[0];
                List<double> outputs = new List<double>();
                for (int i = 0; i < X.NnData.Length; i++)
                {

                    for (int j = 0; j < X.NnData[i].Length; j++)
                    {
                        inputLayer.Neurons[j].ActivationPotencial.Value =
X.NnData[i][j];
                    }


                    ComputeOutput();

outputs.Add(Layers.Last().Neurons.First().ActivationPotencial.Value);
                }
                double accuracySum = 0;
                int y_counter = 0;
                outputs.ForEach((x) => {
                    if (x == Y.NnData[y_counter].First())
                    {
                        accuracySum++;
                    }

                    y_counter++;
                });
                OptimizeWeights(accuracySum / y_counter);
                Console.WriteLine("Iteration: {0}, Accuracy: {1} %",
iteration, (accuracySum / y_counter) * 100);
            Print();
                iteration++;
            }
        }
```

Metoda OptimizeWeights se koristi kako bi se optimizirale, odnosno popravile težine sinapsa između neurona u svrhe povećanja postotka točnosti.

```
private void OptimizeWeights(double accuracy)
        {
            float lr = 0.1f;

            if (accuracy == 1)
            {
                return;
            }

            if (accuracy > 1)
            {
                lr = -lr;
            }

            foreach (var layer in Layers)
            {
                layer.Optimize(lr, 1);
            }
        }
```

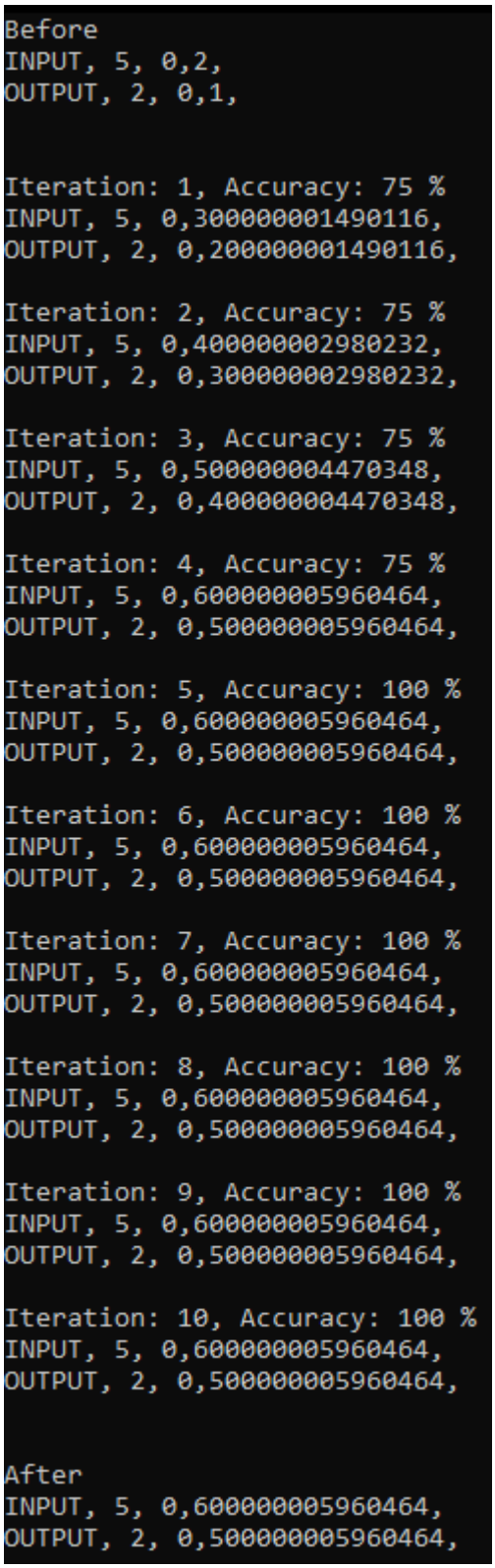
Metoda uzima trenutnu točnost, preskače optimizaciju ako je točnost na 100%, te u slučaju da točnost nije zadovoljena, poziva metodu Optimize za svaki sloj te za sve veze u tim slojevima kako bi se postavile nove težina sinapsi.

```
    public void Optimize(double learningRate, double delta)
        {
            Weight += learningRate * delta;
            foreach (var neuron in Neurons)
            {
                neuron.UpdateWeights(Weight);
            }
        }
```

Sama konzolna aplikacija prvo instancira novu neuronsku mrežu te joj dodaje ulazni i izlazni sloj sa određenim brojem neurona, inicijalnim težinama i imenom sloja radi lakšeg pretraživanja. Nakon toga se poziva metoda Build za neuronsku mrežu i ispisuju se početne težine. Nakon što je mreža izgrađena, stvaraju se novi ulazni podatci i očekivani rezultati te se poziva metoda Train sa 10 iteracija i stopa učenja je 0.1 zbog jednostavnosti neuronske mreže. Nakon što je mreža prošla proces učenja, ispisuju se novi korigirani potencijali.

```
static void Main(string[] args)
        {
            NeuralNetwork nn = new NeuralNetwork();
            nn.Layers.Add(new Layer(5, 0.2, "INPUT"));
            nn.Layers.Add(new Layer(2, 0.1, "OUTPUT"));

            nn.Build();
            Console.WriteLine("Before");
            nn.Print();

            Console.WriteLine();

            Data X = new Data(4);
            X.Add(0, 0);
            X.Add(0, 1);
            X.Add(1, 0);
            X.Add(1, 1);

            Data Y = new Data(4);
            Y.Add(0);
            Y.Add(0);
            Y.Add(0);
            Y.Add(1);

            nn.Train(X, Y, iterations: 10, learningRate: 0.1);
            Console.WriteLine();
            Console.WriteLine("After");
            nn.Print();
            Console.ReadLine();
        }
```

Rezultati ove mreže su prikazani na slici 5. U rezultatima neuronske mreže možemo vidjeti kako je nakon 4. iteracije postignuta tražena stopostotna točnost. Ujedno možemo vidjeti kako su iz ulaznih slojeva sa težinama od 0.2 i izlaznih slojeva sa težinama 0.1, nakon učenja neuronske mreže, korigirane težine na 0.60 i 0.50. Očekivano je da će se brzo postignuti tražena stopa točnosti, zbog malog broja slojeva i jednostavnosti zadatka koji izvršava. Broj neurona u ovome slučaju nije imao nikakvog utjecaja na konačni rezultat, već se zbog povratne propagacije nakon prvih par iteracija dobije stopostotna točnost. Ovakvim primjerom možemo vidjeti promjene težina veza dok neuronska mreža uči.



Slika 5: Rezultat jednostavne neuronske mreže (autorski rad)

3. O algoritmima evolucijskog računarstva

Mnogo izuma koje se koriste u današnjici možemo vidjeti kao rezultat primjene prirodnih principa i značajki u svrhe učenja i stvaranja ljudskih pandana. Takve izume možemo vidjeti u podmorskim sonarima koji su napravljeni na principu eholokacije kitova i dupina, oponašanje šišmiša, od kojega je nastao radar ili čak ptice, čijim je oponašanjem izumljen zrakoplov. Evolucija stvorenja se može gledati kao adaptiranja na okolinu i optimizacije za preživljavanje vrste. Kada oponašamo takav pogled genetike u dizajniranju i optimiziranju modernih algoritama, stvaramo takozvane algoritme evolucijskog računarstva ili kraće evolutivne algoritme.

Evolutivni algoritmi ili ponekad zvani evolutivna inteligencija su algoritmi bazirani na evoluciji, što znači da se mijenjaju kako bi efektivnije odrađivali zadatke optimizacije i učenja. Evolutivni algoritmi imaju tri glavne karakteristike koje navode Xinjie Yu i Mitsuo Gen (2010.):

1. Bazirani na populaciji – Kako bi se lakše riješili problemi koji se često ponavljaju ili imaju sličnu strukturu kao i već riješeni problem, evolutivni algoritmi sadrže populacije, odnosno skupine rješenja kako bi se optimizirali ili naučili riješiti problem na paralelan način.
2. Orijentirani na sposobnost – Kako je već navedeno da su evolutivni algoritmi bazirani na skupinama rješenja zvanih populacije, možemo zaključiti kako je jedno rješenje u tom skupu smatrano kao jedinka. Svaka jedinka je reprezentirana sa određenim genom, koji je u slučaju algoritama sami kod, a performansa toga koda se naziva vrijednost njegove sposobnosti. Preferencija se nalazi u sposobnijim jedinkama, jer su one temelj same optimizacije algoritama.
3. Vođeni varijacijom – Više varijacija jedinki je potrebno kako bi se oponašale promjene gena kao i u stvorenjima i to u svrhe efektivnije pretrage prostora rješenja, ali i stvaranja novih rješenja.

Evolucijski algoritmi postali su popularni alati za pretraživanje, optimizaciju, strojno učenje i za rješavanje dizajnerskih problema. Postoji mnogo različitih vrsta evolucijskih algoritama. Povijesno gledano, genetski algoritmi i evolucijske strategije dva su najosnovnija oblika evolucijskih algoritama. Algoritmi stavljaju velik naglasak na selekciju, rekombinaciju i mutaciju koja djeluje na genotip koji se dekodira i procjenjuje na prikladnost. Rekombinacija je naglašena u odnosu na mutaciju. Za optimizaciju su korišteni i genetski algoritmi i evolucijske strategije. Međutim, genetski algoritmi su se dugo smatrali kao višenamjenski alati za svrhe pretrage i optimizacije.

3.1. Genetski algoritam

Jedna od poznatijih podvrsta evolutivnih algoritama je genetski algoritam. Genetski algoritam je inspiriran biološkim procesom evolucije. Ovakve je algoritme moguće usporediti sa Darwinovom teorijom evolucije u prirodi i to zbog funkcioniranja genetskih algoritma koji simuliraju proces promjene nasljednih osobina populacije kroz određen broj generacija.

„Genetski algoritam (GA) je optimizacijski algoritam koji je inspiriran prirodnim izborom. To je algoritam pretraživanja temeljen na populaciji, koji koristi koncept opstanak najjačih.“ (Katoch, S., Chauhan, S.S., Kumar, V., 2021, str. 8094)

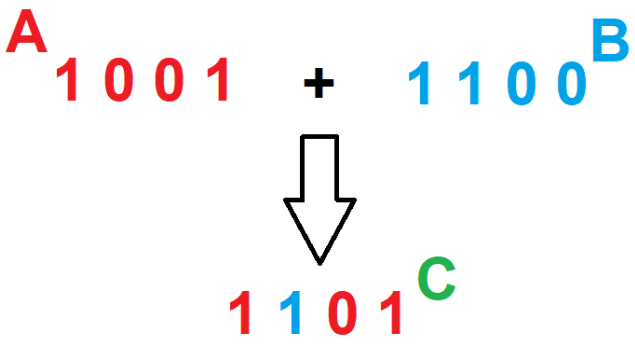
Evolucija je sama po sebi dosta kompliciranа, ali implementacija logike evolucije je u današnjici dosta pojednostavljena kako bi se mogli stvarati algoritmi koji se temelje na genetskoj strukturi i ponašanju kromosoma populacije. Što je ustvari kromosom u smislu računalstva? Kromosomi su moguća rješenja. Već je bilo navedeno kako su evolutivni algoritmi orijentirani na sposobnost pojedinca. Kumar M., i suradnici (1. 12. 2010). navode kako svaka jedinka ima jedan kromosom koji sadrži karakteristike jedinke, odnosno skupine gena. Ovime možemo zaključiti da je populacija jedinki ustvari zbirka kromosoma. Geni sadrže velik broj informacija različitog tipa, a to možemo zaključiti pogledom na ljudske ekvivalente. Geni koji sadrže informacije poput visine ili boje kože ne možemo izraziti sa samo jednim tipom podataka u računarstvu.

Funkcija sposobnosti karakterizira svaku jedinku u populaciji. Ta sposobnost pokazuje koliko je jedinka sposobna da se natječe sa ostalim jedinkama za šansu da se razmnožava. Atul Kumar (27. 4. 2022) navodi kako se iz populacije uzimaju najbolje jedinke čiji će potomci naslijediti gene roditelja kombinacijom kromosoma od oba roditelja. Vjerojatnost da će jedinka biti odabrana za reprodukciju temelji se na ocjeni njezine sposobnosti. Neke od jedinka u novoj će u iteraciji, odnosno generaciji, odumrijeti i biti zamijenjene novim jedinkama koje na kraju stvaraju nove generacije kada su sve mogućnosti parenja stare populacije iscrpljene. Svaka nova generacija stvara više boljih jedinki, odnosno gena tih jedinki iz prethodnih generacija. Rješenja koja novije generacije daju nisu nužno bolja, već su drugačija od prijašnjih rješenja. Bitno je shvatiti kako se to natjecanje odvija u svrhu pronalaska boljih rješenja, i nadom da će manje sposobnija rješenja izumrijeti.

Jedno od postavka genetskog algoritma je elitizam. Elitizam se koristi za smanjenje genetske varijacije time da omogućava reprodukciju samo onim jedinkama koje imaju najveću sposobnost.

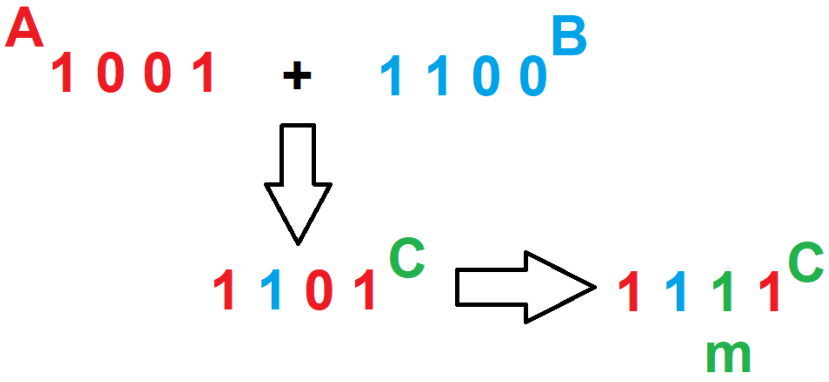
Atul Kumar (27. 4. 2022) ujedno navodi kako genetski algoritam koristi 3 važna operatora po kojima se razvija, odnosno koji se odvijaju na početnoj generaciji.

1. Odabir – Odabiru se one jedinke čiji rezultati sadrže zadovoljavajuću ocjenu spremnosti kako bi se omogućio prijenos kvalitetnijih gena na iduće generacije.
2. Križanje – Križanje je proces stvaranja novih potomaka preko roditelja koji su odabrani iz operatora odabira. Pozicije križanja gena se odabiru nasumično te se geni na tim mjestima izmjenjuju i pritom stvaraju novog potomka, odnosno jedinki. Križanje može stvoriti više od jednog potomka. Slika 6. prikazuje proces križanja roditelja A i B te stvaranja potomka C.



Slika 6: Križanje gena (autorski rad)

3. Mutacija – Kako bi došlo do varijacije u rješenjima tj. kako bi se izbjeglo podudaranost osobina gena, potrebno je uključiti nasumične gene u potomke. To znači da će se neki od gena u jedinki zamijeniti sa mutiranim genom koji razbija sličnosti rješenja. Slika 7. prikazuje mutaciju potomka C.



Slika 7: Mutacija gena (autorski rad)

Genetski algoritmi su jednostavni optimizatori za probleme koji se ne mogu učinkovito optimizirati jer su spori, heuristički i zahtijevaju mnogo resursa. Mogu se koristiti za optimizaciju nekih apstraktnijih dijelova, poput mrežne arhitekture itd., gdje se ne mogu izračunati valjani skalarni vektori ili čak koristiti druge dobro definirane matematičke procedure.

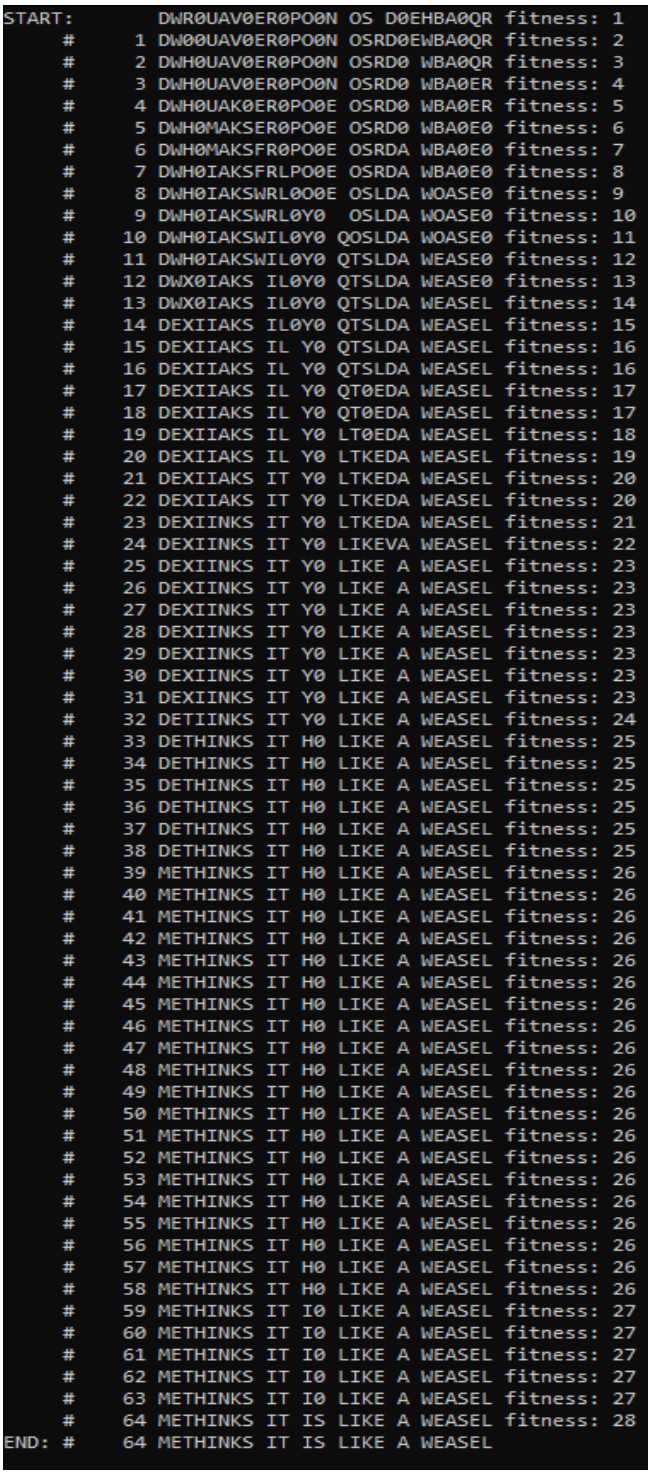
3.2. Dawkinsova lasica

Jedan od ranijih primjera genetskog algoritma je Dawkinsova lasica. Richard Dawkins u svojoj knjizi „***The Blind Watchmaker***“ (1986.), opisuje program lasice (eng .Weasel program). Sami program je baziran na teoremu beskonačnog majmuna, koji glasi da ako majmun koji nasumice lupa po pisaćem stroju, uz dovoljno vremena, može proizvesti sva Shakespearova djela. Scenarij je postavljen tako da proizvede niz besmislica , uz pretpostavku da će odabir svakog slova u nizu od 28 znakova biti nasumičan. Glavna varijacije teorema beskonačnog majmuna i Dawkinsonovog programa je postojanje slučajne pogreške, odnosno mutacije u kopiranju, što znači da preslikava izvornu besmislenu frazu, te ispituje potomke te fraze. Odabire se ona koja barem malo sliči ciljnoj frazi, odnosno ona koja ima najviše pogodaka gdje se nalaze točna slova u točnim pozicijama. Ovaj algoritam se smatra genetskim algoritmom, a to se može zaključiti evolucijom niza znakova, koji se zovu geni i koji se mijenjaju kroz generacije, naravno ovaj algoritam nije pravi genetski algoritam, već najpoznatiji primjer i temelj od kojeg su nastali genetski algoritmi sa operatorima selekcije, križanja i mutacije. Algoritam se može opisati na sljedeći način:

1. Započni s nasumičnim nizom od 28 znakova. Svi važeći znakovi su velika slova i razmak.
2. Napravi 100 kopija tog niza, s 5% vjerojatnosti po znaku da taj znak bude zamijenjen nasumičnim znakom.
3. Usporedi svaki novi niz s ciljem "MISLIM DA JE KAO LASICA" (eng. „METHINKS IT IS LIKE A WEASEL“) i svakom dodijeli rezultat prema broju slova u nizu koja su točna i na ispravnom položaju.
4. Ako bilo koji od novih nizova znakova ima točan rezultat (28), stani. (The Blind Watchmaker, 2013.)

Primjer ovakvog koda napravljen je u C# jeziku u programu Visual Studio 2022. Kod je repliciran i preveden sa NetLogo jezika u C#, a originalno ga je napisala Kristin Crouse (21. 11. 2019). Rješenje Dawkinsonove lasice se nalazi na slici 8.

```
  public static class Helper
    {
        public static Random Rng = new Random((int)DateTime.Now.Ticks);
        public static char NextCharacter(this Random self)
        {
            const string AllowedChars = "00000 ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            return AllowedChars[self.Next() % AllowedChars.Length];
        }

        public static string NextString(this Random self, int length)
        {
            return String.Join("", Enumerable.Repeat(' ', length)
                .Select(c => Rng.NextCharacter()));
        }

        public static int Fitness(string target, string current)
        {
            return target.Zip(current, (a, b) => a == b ? 1 : 0).Sum();
        }

        public static string Mutate(string current, double rate)
        {
            return String.Join("", from c in current
                                   select Rng.NextDouble() <= rate ?
Rng.NextCharacter() : c);
        }
    }

    internal class Program
    {

        static void Main(string[] args)
        {
            const string target = "METHINKS IT IS LIKE A WEASEL";
            const int C = 100;
            const double P = 0.05;

            string parent = Helper.Rng.NextString(target.Length);

            Console.WriteLine("START:       {0,20} fitness: {1}",
                parent, Helper.Fitness(target, parent));
            int i = 0;

            while (parent != target)
            {
                var candidates = Enumerable.Range(0, C + 1)
                    .Select(n => n > 0 ? Helper.Mutate(parent, P) :
parent);

                parent = candidates.OrderByDescending(c =>
Helper.Fitness(target, c)).First();

                ++i;
                Console.WriteLine("     #{0,6} {1,20} fitness: {2}",
                    i, parent, Helper.Fitness(target, parent));
            }

            Console.WriteLine("END: #{0,6} {1,20}", i, parent);
            Console.ReadLine();
        }
    }
```

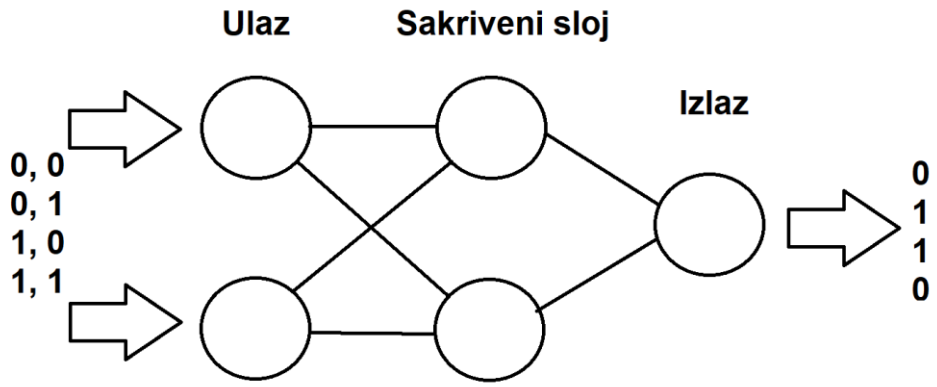


Slika 8: Rezultat Dawkinsonove lasice (autorski rad)

Kada bi pokušali koristiti brute force nasumičnim algoritmom pogađanja, i to najmoćnijim superračunalom današnjice, bilo bi nam potrebno nekoliko milijuna godina da se ispravno pogodi zadani niz od 28 znakova. Naime broj mogućih kombinacija u ovom slučajnom nizu je 27^{28} ili blizu 10^{40}. Dawkinsova lasica je ovakav problem riješila u 64 koraka, što dokazuje uspješnost genetskog algoritma.

4. Izgradnja neuronske mreže za rješavanje XILI uz pomoć genetskog algoritma

Sljedeći primjer prikazuje izgradnju neuronske mreže uz pomoć genetskog algoritma za rješavanje logičke operacije XILI (eng. XOR). Repozitorij za genetski algoritam je napravio Barry Lapthorn (2003.), čiji je kod izmjenio Lionel Monnier (2004.). Promjena koda se odnosila na optimizaciju i poboljšanje .NET 2.0 C# jezika. Repozitorij sadrži klase GA (genetski algoritam) i genom. Slika 9. prikazuje skicu neuronske mreže koja rješava logičku operaciju XOR. Za razliku od logičkih operacija AND i OR, za XOR nam je potreban dodatan sakriveni sloj kako bi neuronska mreža mogla riješit problem.



Slika 9: Skica neuronske mreže za logičku operaciju XOR (autorski rad)

Klasa genom sadrži metode za manipuliranje genima poput mutacije, križanja, stvaranja gena, preslikavanja i prikupljanja vrijednosti tih gena.

Metoda CreateGenes stvara nove gene korisnički zadane veličine sa vrijednostima između -20 i 20. To je metoda koja stvara gene sa nasumičnim vrijednostima.

```
  private void CreateGenes()
            {
            for (int i = 0; i < m_genes.Length; i++)
                m_genes[i] = (m_random.NextDouble() + m_random.Next(-20,
20));
        }
```

Metoda Crossover se odvija na svakoj jedinki u populaciji. Jedinka na kojoj se metoda vrši se gleda kao prvi roditelj i križa gene sa svim ostalim jedinkama u populaciji. Geni se križaju na nasumičnim mjestima, te se stvaraju dva djeteta.

```
public void Crossover(ref Genome genome2, out Genome child1, out Genome child2)
            {
                  int pos = (int)(m_random.NextDouble() * (double)m_length);
                  child1 = new Genome(m_length, false);
                  child2 = new Genome(m_length, false);
                  for(int i = 0 ; i < m_length ; i++)
                  {
                        if (i < pos)
                        {
                              child1.m_genes[i] = m_genes[i];
                              child2.m_genes[i] = genome2.m_genes[i];
                        }
                        else
                        {
                              child1.m_genes[i] = genome2.m_genes[i];
                              child2.m_genes[i] = m_genes[i];
                        }
                  }
            }
```

Metoda Mutate mutira nasumične gene određene jedinke ako se slučajnim brojem pogodi vrijednost manja od zadane vjerojatnosti za mutiranje.

```
public void Mutate()
            {
                  for (int pos = 0 ; pos < m_length; pos++)
                  {
                        if (m_random.NextDouble() < m_mutationRate)
                    m_genes[pos] = (m_genes[pos] + (m_random.NextDouble() +
m_random.Next(-20, 20))) / 2.0;
            }
            }
```

Klasa GA sadrži metode stvaranja generacija i jedinki, rangiranja, selekcije najboljih jedinki iz populacije i dohvaćanja najbolje jedinke iz svih generacija.

Genetski algoritam se pokreće pozivanjem metode Go koja prvo stvara tablicu sposobnosti kako bi se jedinke mogle rangirati.

Metoda CreateGenome stvara određen broj jedinki zadan brojem populacije te dopunjuje trenutnu generaciju sa njima.

```
private void CreateGenomes()
            {
                  for (int i = 0; i < m_populationSize ; i++)
                  {
                        Genome g = new Genome(m_genomeSize);
                        m_thisGeneration.Add(g);
                  }
            }
```

Metoda CreateNextGeneration se okida sljedeća, a ona je zadužena za stvaranje nove populacije sa zadanim brojem jedinki. Metoda prvo čisti listu za sljedeću generaciju kako bi se lista mogla popuniti sa novim jedinkama i testirati. Metoda prvo provjerava uvjet elitizma, što znači da ako je zadovoljen, kopirati će se određen broj najboljih jedinki iz prošle generacije. Nakon toga se za svaku jedinku u populaciji odvija razmnožavanje. Tj. stvaraju se po 2 djeteta. Roditelji se biraju nasumično te se odmah započinje križanje gena roditelja za djecu. Nakon križanja, odvija se mutiranje gena djece i ona su onda dodana u novu generaciju. U slučaju da elitizam uvjet nije zadovoljen, razmnožavati će se moći sve jedinke iz prošle generacije.

```
      private void CreateNextGeneration()
            {
                  m_nextGeneration.Clear();
                  Genome g = null;
                  if (m_elitism)
                        g = m_thisGeneration[m_populationSize -
1].DeepCopy();

                  for (int i = 0 ; i < m_populationSize ; i+=2)
                  {
                        int pidx1 = RouletteSelection();
                        int pidx2 = RouletteSelection();
                        Genome parent1, parent2, child1, child2;
                        parent1 = m_thisGeneration[pidx1];
                        parent2 = m_thisGeneration[pidx2];

                        if (m_random.NextDouble() < m_crossoverRate)
                        {
                              parent1.Crossover(ref parent2, out child1, out
child2);
                        }
                        else
                        {
                              child1 = parent1;
                              child2 = parent2;
                        }
                        child1.Mutate();
                        child2.Mutate();

                        m_nextGeneration.Add(child1);
                        m_nextGeneration.Add(child2);
                  }
                  if (m_elitism && g != null)
                        m_nextGeneration[0] = g;

                  m_thisGeneration.Clear();
            foreach(Genome ge in m_nextGeneration)
                        m_thisGeneration.Add(ge);
            }
```

Nakon što je nova generacija napravljena, rangira se svaka jedinka preko funkcije sposobnosti koja je definirana u glavnom programu. Nakon što su sve jedinke rangirane, metoda ih sortira i ispisuje se za svakih 100 generacija najveća sposobnost.

```
private double RankPopulation()
            {
                  m_totalFitness = 0.0;
                  foreach(Genome g in m_thisGeneration)
                  {
                        g.Fitness = FitnessFunction(g.Genes());
                        m_totalFitness += g.Fitness;
                  }
            m_thisGeneration.Sort(delegate(Genome x, Genome y)
                { return Comparer<double>.Default.Compare(x.Fitness,
y.Fitness); });

            //  now sorted in order of fitness.
            double fitness = 0.0;
            m_fitnessTable.Clear();
            foreach(Genome t in m_thisGeneration)
                  {
                        fitness += t.Fitness;
                        m_fitnessTable.Add(t.Fitness);
            }

            return m_fitnessTable[m_fitnessTable.Count - 1];
              }
```

Vijeth Dinesha je 2008. godine razvio NeuronDotNet repozitorij. NeuronDotNet nije tako širok u svom opsegu, ali njegov isključivi fokus na neuronske mreže ga čini možda lakše dostupnim, a možda i više predstavljenim u ovom području. Prednost ovog repozitorija je laka implementacija slojeva i veza između njih te implementirana funkcija povratne propagacije kako bi se ubrzalo popravljanje aktivacijskih potencijala i težina sinapsa. Repozitorij je danas napušten, ali je još uvijek često korišten u svrhe izgradnje laganih neuronskih mreža. Ideje i primjere korištenja metoda iz repozitorija su pronađene preko programa Kory Beckera (2009.) kako bi se riješio problem nedostatka korisničke dokumentacije. Metode koje se koriste iz ovog repozitorija su povratna propagacija, stvaranje mreže te postavljanje težina sinapsi preko povratne propagacije. Klase koje su korištene uključuju slojeve tipa LinearLayer te SigmoidLayer. Aktivacijska funkcija koja se koristi je sigmodalna funkcija zbog postojanja sigmoidnih slojeva.

```
static void Main(string[] args)
        {
            LinearLayer inputLayer = new LinearLayer(2);
            SigmoidLayer hiddenLayer = new SigmoidLayer(2);
            SigmoidLayer outputLayer = new SigmoidLayer(1);

            BackpropagationConnector connector1 = new
BackpropagationConnector(inputLayer, hiddenLayer);
            BackpropagationConnector connector2 = new
BackpropagationConnector(hiddenLayer, outputLayer);

            network = new BackpropagationNetwork(inputLayer, outputLayer);
            network.Initialize();

            GA geneticAlgorithm = new GA(0.50, 0.05, 200, 5000, 12);
            geneticAlgorithm.FitnessFunction = new
GAFunction(fitnessFunction);
            geneticAlgorithm.Elitism = true;
            geneticAlgorithm.Go();

            double[] weights;
            double fitness;
            geneticAlgorithm.GetBest(out weights, out fitness);
            Console.WriteLine("Best fitness overall: " + fitness);

            setNetworkWeights(network, weights);

            double input1 = 0;
            double input2 = 0;
            while (input1 == 0 || input1 == 1)
            {
                Console.Write("Input 1: ");
                input1 = Convert.ToDouble(Console.ReadLine());

                Console.Write("Input 2: ");
                input2 = Convert.ToDouble(Console.ReadLine());

                double[] output = network.Run(new double[2] { input1,
input2 });
                Console.WriteLine("Weight strength for given inputs: " +
output[0]);
            }
        }
```

U glavnom programu se prvo inicijalizira 3 sloja. Jedan ulazni sloj sa 2 neurona koji sadrže linearne funkcije pošto su ulazi u te neurone 1 ili 0, jedan sakriveni sloj sa isto dva neurona te izlazni sloj sa jednim neuronom. Nakon što su se stvorili slojevi, stvaraju se veze između slojeva tipa BackpropagationConnector koji povezuju ulazne i izlazne slojeve sa tajnim slojem. Kada smo stvorili veze i slojeve, možemo napraviti mrežu za povratnu propagaciju kako bi kasnije mogli podešavati težine i aktivacijske potencijale. Sada inicijaliziramo tu mrežu te ujedno inicijaliziramo genetski algoritam koji ima 50% stope križanja, 5% stope mutacije, populaciju veličine 200, proći će 5000 generacije, a pojedinci će imati 12 gena. Sada možemo genetskom algoritmu pridodati funkciju sposobnosti XOR.

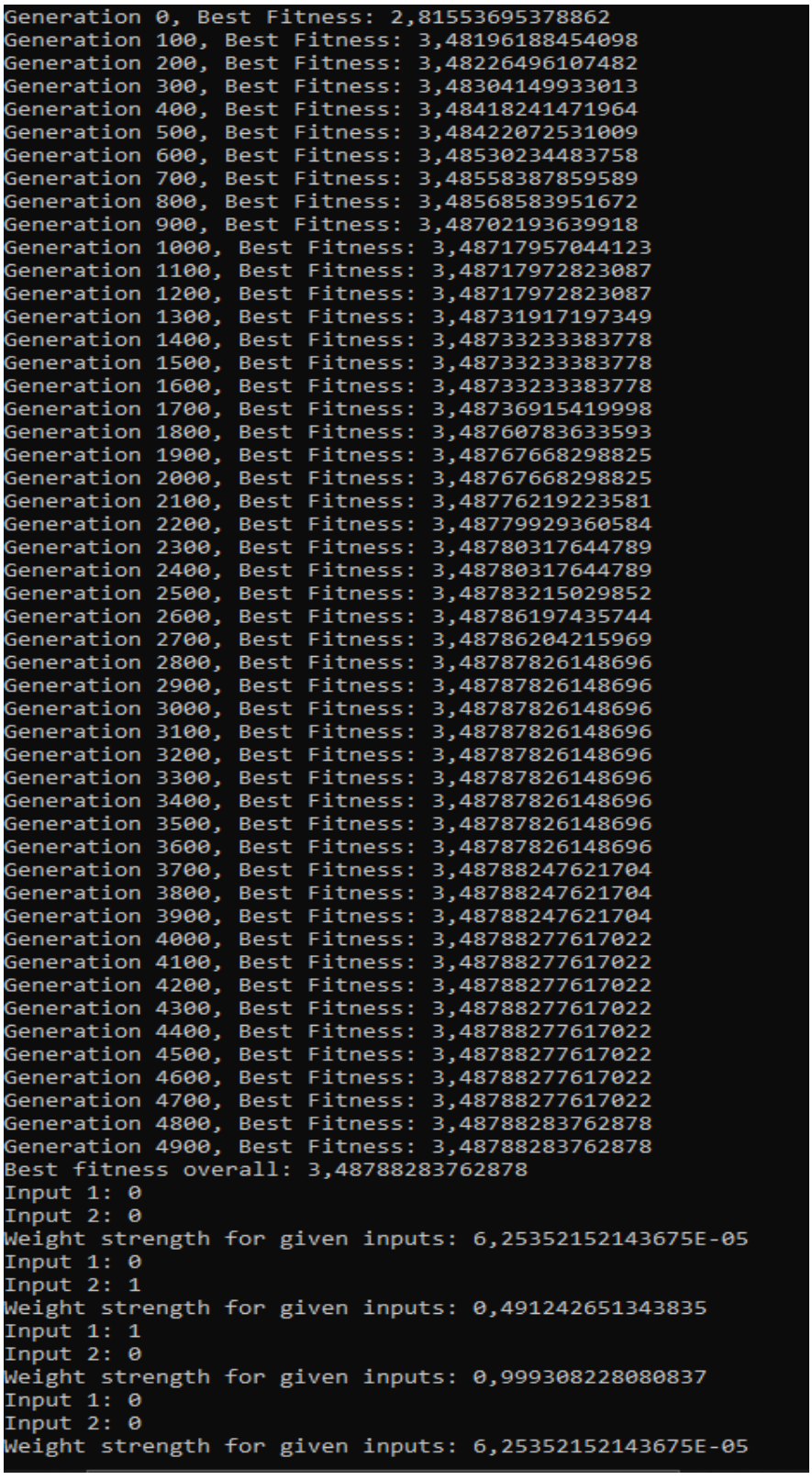
```
public static double fitnessFunction(double[] weights)
        {
            double fitness = 0;
            setNetworkWeights(network, weights);
            double output = network.Run(new double[2] { 0, 0 })[0];
            fitness += 1 - output;

            output = network.Run(new double[2] { 0, 1 })[0];
            fitness += output;

            output = network.Run(new double[2] { 1, 0 })[0];
            fitness += output;

            output = network.Run(new double[2] { 1, 1 })[0];
            fitness += 1 - output;

            return fitness;
        }
```

Metoda vraća sposobnost za svaku pojedinu jedinku. Mogući ulazi u ovu metodu su: 0 i 0, 0 i 1, 1 i 0 te 1 i 1. Sposobnost se u ovome slučaju računa tako da se sumirani potencijal i težina oduzmu od broja jedan u slučaju pogreške, odnosno ostave takve kakve jesu u slučaju pogotka. Što je izlaz bliži nuli u slučaju greške i bliži jedinici u slučaju pogotka, to je jedinka sposobnija.

Metoda setNetworkWeights za svaku sinapsu u konektoru postavlja novu težinu sinapse i novu jačinu aktivacijskog potencijala.

```
        public static void setNetworkWeights(BackpropagationNetwork
Network, double[] weights)
        {
            int i = 0;
            foreach (BackpropagationConnector connector in
Network.Connectors)
            {
                foreach (BackpropagationSynapse synapse in
connector.Synapses)
                {
                    synapse.Weight = weights[i++];
                    synapse.SourceNeuron.SetBias(weights[i++]);
                }
            }
        }
```

Kada pokrenemo program, prvo se prikaže svaka stota generacije iz razloga preglednosti. Za svaku generaciju je ujedno prikazana i najbolja sposobnost u generaciji. Prije nego li unesemo podatke za testiranje, prikazana je najbolja sposobnost kroz sve generacije.



Slika 10: Rezultat neuronske mreže sa genetskim algoritmom (autorski rad)

Možemo zaključiti da je neuronska mreža sa genetskim algoritmom uspješna iz testnih podataka jer sve izlazne težine koje su netočne uvelike mali brojevi koje ne bi mogli zaokružiti na 1. Točan ulaz 1 i 0 te 0 i 1 su jedini koji su poprimili veće brojeve. Na slici možemo vidjeti kako je ulaz 1 i 0 konkretno točan sa 0.99 sumom težina, ali ujedno i 0.49 težinu ulaza 0 i 1. Zašto je to tako? Već je prije bilo objašnjeno kako je genetički algoritam ustvari optimizator, a ne potpuni algoritam učenja. To znači da će kroz generacije doći do mutacija koje odstupaju od tog drugog odgovora. Jedno od rješenja bilo bi podići broj populacije, jer bi onda veći broj jedinki bio pristraniji tome ulazu te bi ujedno bila jača suma težina i aktivacijskih potencijala prema tome. Naravno ovakvo rješenje je dvosjekli mač. Pošto je ujedno jedan od razloga taj da je više promašaja zbog toga prvog ulaza kada je on 0, neuron koji okida na strani gdje često ulazi ta 0 biti će slabiji. Iako naravno kroz veći broj generacija, veća je i mogućnost mutacije koja kroz kontinuiranu selekciju dolazi do veće točnosti predviđanja.

5. Zaključak

Rezultat ovog završnog rada je izgrađena jednostavna neuronska mreža koja rješava logičku operaciju I, napisana u C# jeziku koristeći program Visual Studio 2022., genetski algoritam baziram na Dawkinsonovoj lasici te neuronska mreža koja uz pomoć genetskog algoritma rješava logičku operaciju XILI. U radu su prikazani svi kodovi i rezultati algoritama i neuronskih mreža, u svrhe objašnjenja funkcioniranja istih.

U teorijskim dijelovima rada je objašnjeno funkcioniranje neuronskih mreža i od čega su napravljene, odnosno kako su građene te su objašnjene funkcije i načini učenja neuronskih mreža metodama povratne propagacije. Ujedno je objašnjen i sam proces učenja. Neuronske mreže su objašnjene na način da se dobije uvid u apstrakciju mreže, pojašnjivanjem značenja, građe i funkcioniranja samih umjetnih neurona od kojih su građeni. Objašnjene su i usporedbe sa biološkim ekvivalentima, te su istaknute i formule koje se koriste u funkcioniranju neuronskih mreža i mrežnom učenju. Dodatno su pojašnjeni algoritmi evolucijskog računarstva kao alati za strojno učenje i optimizaciju od kojih je najpoznatiji genetski algoritam. Genetski algoritam kao podvrsta algoritma evolucijskog računarstva je detaljno pojašnjen u smislu njegovog funkcioniranja i njegovih operacija koje izvodi, poput mutacije, križanja i odabira. Genetski algoritam je prikazan primjerom Dawkinsonove lasice, čiji je kod i način korištenja detaljno opisan.

Neuronska mreža predstavlja kvalitetan način strojnog učenja simulirajući funkcioniranje bioloških neuronskih mreža u biološkom mozgu. Fokus rada je bio na teorijskoj podlozi kako bi se lakše razumjelo kako se uopće neuronska mreža gradi, kako funkcionira, zašto koristi algoritme evolucijskog računarstva, te kako ti algoritmi uopće funkcioniraju. U praktičnim dijelovima rada su prikazani rezultati te teorijske podloge izgradnjom jednostavne neuronske mreže, genetskog algoritma i neuronske mreže koja koristi genetski algoritam.

Svi programi, odnosno neuronske mreže i genetski algoritam, su izgrađeni u C# jeziku koristeći program Visual Studio 2022. Prednost takvoj jezika je ustvari to što je objektno orijentiran, te je instanciranje neurona, veza pa čak i samih neuronskih mreža bilo uvelike olakšano. Uspješnom izgradnjom ovih programa te kvalitetnim rješenjima koje daju u radu se može zaključiti korisnost strojnog učenja i umjetne inteligencije koja bi jednoga dana mogla potpuno simulirati ljudski mozak.

6. Popis slika

Slika 1: Jednostavna neuronska mreža (autorski rad) 2
Slika 2: Biological and artificial neuron (izvor: quora.com, 2018.) 6
Slika 3: Presjek umjetnog neurona (autorski rad) 8
Slika 4: Skica neuronske mreže za logičku operaciju AND (autorski rad) 10
Slika 5: Rezultat jednostavne neuronske mreže (autorski rad) 18
Slika 6: Križanje gena (autorski rad).....21
Slika 7: Mutacija gena (autorski rad).....21
Slika 8: Rezultat Dawkinsonove lasice (autorski rad).....24
Slika 9: Skica neuronske mreže za logičku operaciju XOR (autorski rad) 25
Slika 10: Rezultat neuronske mreže sa genetskim algoritmom (autorski rad) 32

7. Literatura

1. Novaković B., Majetić D., Široki M. (1998). *Umjetne neuronske mreže*, Zagreb: Fakultet strojarstva i brodogradnje.
2. Kelleher D. J. (2019). *Deep Learining*, Massachussetts, USA: Massachussetts Institute of Technology
3. Yu X., Gen M., (2010). *Introduction to Evolutionary Algorithms, Springer; 2010th edition.* England: Springer London
4. Bez autora, (2013), *The Blind Watchmaker*, pristupljeno 25. 8.2022. http://elaq.github.io/weasel/
5. Kristin Crouse (21. 11. 2019). "*Dawkins Weasel*" (Version 1.0.2). CoMSES Computational Model Library. Preuzeto 26.8.2022., https://www.comses.net/codebases/6042/releases/1.0.2/
6. La F. (4. 3. 2019). *How Do Neural Networks Learn?*, Volume 34 Number 4, pristupljeno 25.8.2022. https://docs.microsoft.com/en-us/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn
7. Rakhecha A. (28. 5. 2019). *Understanding Learning Rate*, pristupljeno 25.8.2022., https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de

8. Yathish V., (4. 8. 2022). *Loss Functions and Their Use In Neural Networks,* pristupljeno 26.8.2022., https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9
9. Katoch, S., Chauhan, S.S., Kumar, V. (veljača, 2021). *A review on genetic algorithm: past, present, and future*. *Multimed Tools Appl* 80, 8091–8126, pristupljeno 26.8.2022., https://doi.org/10.1007/s11042-020-10139-6
10. Kumar M., Husain M., Upreti N., Gupta D. (1. 12. 2010*). Genetic Algorithm: Review and ApplicationAvailable*, pristupljeno 28.8.2022., http://dx.doi.org/10.2139/ssrn.3529843
11. Kumar A., (25. 8. 2022). *Genetic Algorithms,* pristupljeno 27.8.2022., https://www.geeksforgeeks.org/genetic-algorithms/
12. Becker K., (6. 5. 2009). *Using Neural Networks and Genetic Algorithms in C# .NET* pristupljeno 30. 8. 2020., http://www.primaryobjects.com/2009/05/06/using-neural-networks-and-genetic-algorithms-in-c-net/
13. Lapthorn B. (2003). modificirao Monnier L. (28. 2. 2005). *A simpler C# genetic algorith*m (verzija 1.1), preuzeto 30. 8. 2020. sa https://www.codeproject.com/Articles/9702/A-simpler-C-genetic-algorithm
14. Dinesha V. (22. 8. 2008). *NeuronDotNet - Neural Networks in C#* (verzija 3.0), preuzeto 30. 8. 2020. sa https://sourceforge.net/projects/neurondotnet/
15. Biological and artificial neuron, [Slika 2] (2018.), Preuzeto 23.8.2022. sa https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network