Zahvaljujem doc. dr. sc. Marku Đuraseviću na mentorstvu i pomoći pruženoj pri pisanju završnog rada.

Sadržaj

Uvod

Igranje igre s neuronskom mrežom predstavlja jednu od primjena umjetne inteligencije koja omogućuje računalu da nauči igrati igru bez da mu je unaprijed zadana strategija za pobjedu. Neuronska mreža obučava se na primjerima igre i kroz iterativni proces učenja poboljšava svoju sposobnost igranja igre. Ova tehnika može se primijeniti na različite igre, od klasičnih društvenih igara do kompleksnijih videoigara. U ovom radu odabrana je igra 2048.

Igra se igra tako što pomičući pločice u istom smjeru, igrači pokušavaju kombinirati pločice s istim brojem kako bi stvorili veće brojeve i osvojili što više bodova. Cilj igre je stvoriti pločicu s brojem 2048 ili više, ali igrači mogu nastaviti igrati i nakon što dostignu cilj. Igra se završava kada više nije moguće napraviti potez.

Neuronske mreže predstavljaju jednu od najvažnijih tehnika u području umjetne inteligencije, koja je inspirirana biološkim sustavom mozga i neurona. Igranje igre s neuronskom mrežom predstavlja izazovno i zanimljivo područje istraživanja u području umjetne inteligencije, jer se kroz ovu primjenu neuronskih mreža mogu pronaći novi načini rješavanja problema. Primjena neuronskih mreža u igrama započinje s oblikovanjem neuronske mreže. Neuronska mreža sastoji se od umjetnih neurona koji su povezani u slojevima i koji obrađuju informacije kako bi se donijela odluka. Za oblikovanje neuronske mreže potrebno je pripremiti skup podataka koje ćemo staviti na ulaz mreže. Nakon toga, mreža se obučava pomoću algoritma učenja kako bi naučila donositi odluke. Kroz iterativni proces učenja, neuronska mreža postaje sve bolja i bolja u igri.

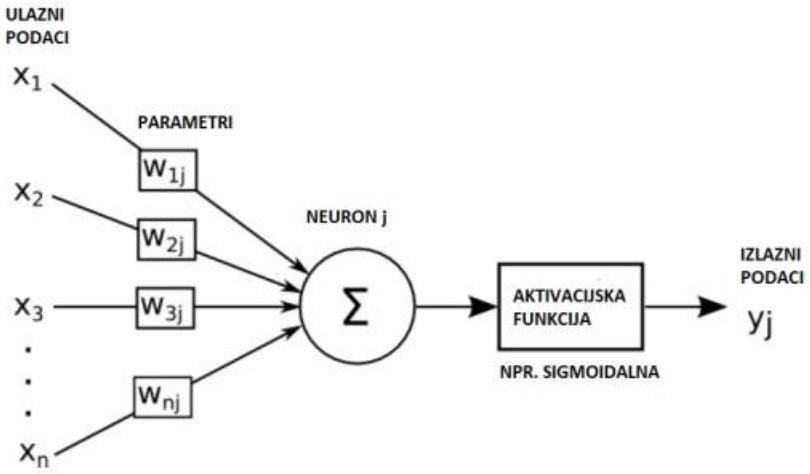
Osim što predstavlja izazovno područje istraživanja u području umjetne inteligencije, primjena neuronskih mreža u igrama ima i praktičnu primjenu. Na primjer, računalne igre mogu se koristiti za testiranje algoritama učenja i različitih strategija te za prikupljanje podataka o tome kako ljudi igraju igre. Ova saznanja mogu se iskoristiti u drugim područjima, kao što su robotika ili strojno prevođenje, gdje se također koriste neuronske mreže.

1. Neuronske mreže

Neuronske mreže, poznate i kao umjetne neuronske mreže (ANN) ili simulirane neuronske mreže (SNN), podskup su strojnog učenja i u središtu su algoritama dubokog učenja. Njihov naziv i struktura inspirirani su ljudskim mozgom, oponašajući način na koji biološki neuroni signaliziraju informacije jedni drugima.

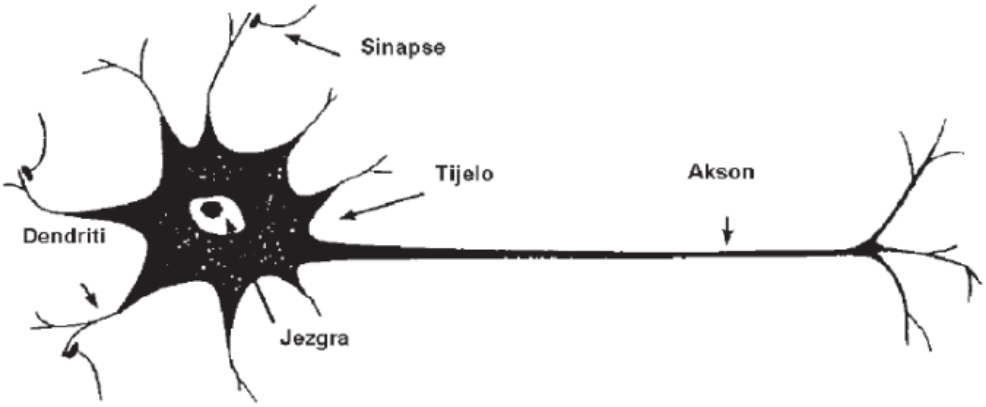
1.1. Umjetni neuron

Umjetni neuron (slika 1.1) je spojna točka u umjetnoj neuronskoj mreži. Umjetne neuronske mreže, poput biološke neuronske mreže ljudi, imaju slojevitu arhitekturu i svaki mrežni čvor ima sposobnost obrade ulaza i prosljeđivanja izlaza drugim čvorovima u mreži. U umjetnim i u biološkim arhitekturama, čvorovi se nazivaju neuroni, a veze su karakterizirane sinaptičkim težinama, koje predstavljaju značaj veze. Kako se novi podaci primaju i obrađuju, sinaptičke težine mijenjaju se i tako dolazi do učenja.



Slika 1.1: Umjetni neuron

Umjetni neuroni modelirani su prema hijerarhijskom rasporedu neurona (slika 1.2) u biološkim senzornim sustavima. Kako neuroni šalju signale kroz sve veći broj slojeva, mozak postupno izvlači više informacija dok ne postane siguran da može identificirati što osoba vidi. U umjetnoj inteligenciji ovaj proces finog ugađanja poznat je kao duboko učenje.



Slika 1.2: Neuron

Izlaz iz umjetnog neurona dobiva se formulom:

- Y – izlaz
- N – broj iteracija
- I - broj na kojoj je iteraciji
- X – ulaz
- W - težina

$$y = f\left(\sum_{i=0}^{n} x_i w_i\right) \qquad (1.1)$$

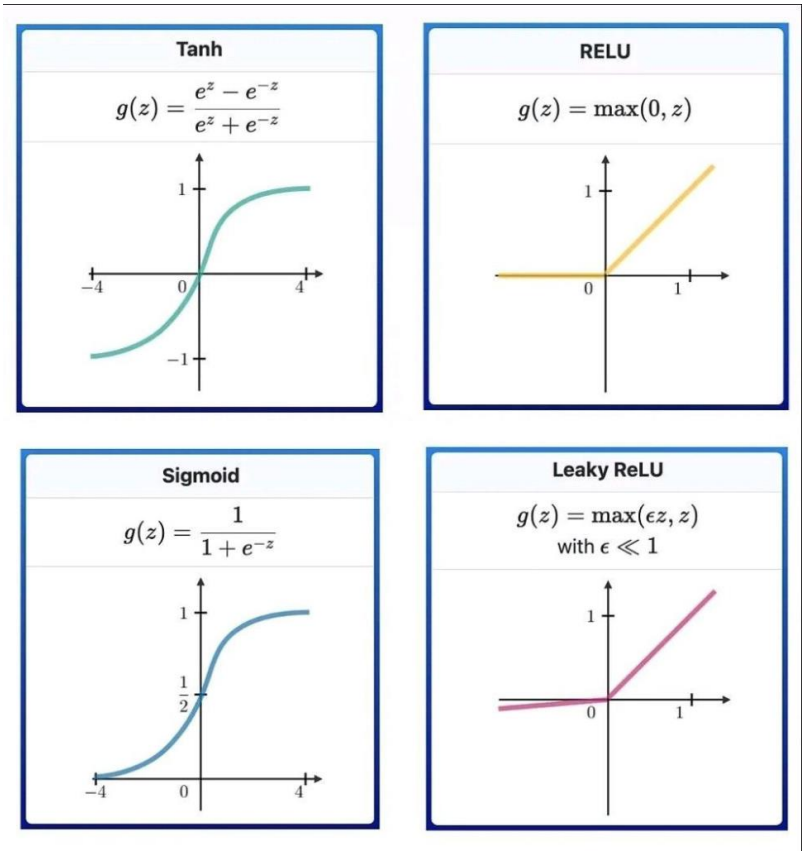
1.2. Aktivacijska funkcija

Aktivacijske funkcije su funkcije koje se koriste u neuronskoj mreži za transformiranje dobivenog izlaza u traženi izlaza. Funkcije manipuliraju prezentiranim podacima i proizvodi izlaz za neuronsku mrežu koja sadrži parametre u podacima (slika 1.3). Aktivacijske funkcije se u nekoj literaturi nazivaju i prijenosnim funkcijama. One mogu biti linearni ili nelinearni, ovisno o funkciji koju predstavljaju i koriste se za kontrolu izlaza neuronskih mreža u različitim domenama.

Linearna funkcija također je poznata kao pravocrtna funkcija gdje je aktivacija proporcionalna ulazu, tj. ponderiranom zbroju neurona. Problem s ovom aktivacijom je što se ne može definirati u određenom rasponu. Primjenom ove funkcije u svim čvorovima funkcija aktivacije radi poput linearne regresije. Završni sloj neuronske mreže radit će kao

linearna funkcija prvog sloja. Drugi problem je spuštanje gradijenta kada se vrši diferencijacija vrši, ima konstantan izlaz što nije dobro jer je tijekom širenja unatrag stopa promjene pogreške konstantna što može uništiti izlaz i logiku širenja unatrag.

Poznato je da su nelinearne funkcije najčešće korištene aktivacijske funkcije. One olakšavaju prilagodbu modela neuronske mreže različitim podacima i razlikovanje ishoda.



Slika 1.3: Nelinearne aktivacijske funkcije

Mora se napraviti pravilan izbor pri odabiru aktivacijske funkcije kako bi se poboljšali rezultati u dobivenim neuronskim mrežama. Sve aktivacijske funkcije moraju biti monotone, diferencijabilne i brzo konvergirajuće u odnosu na težine u svrhu optimizacije.

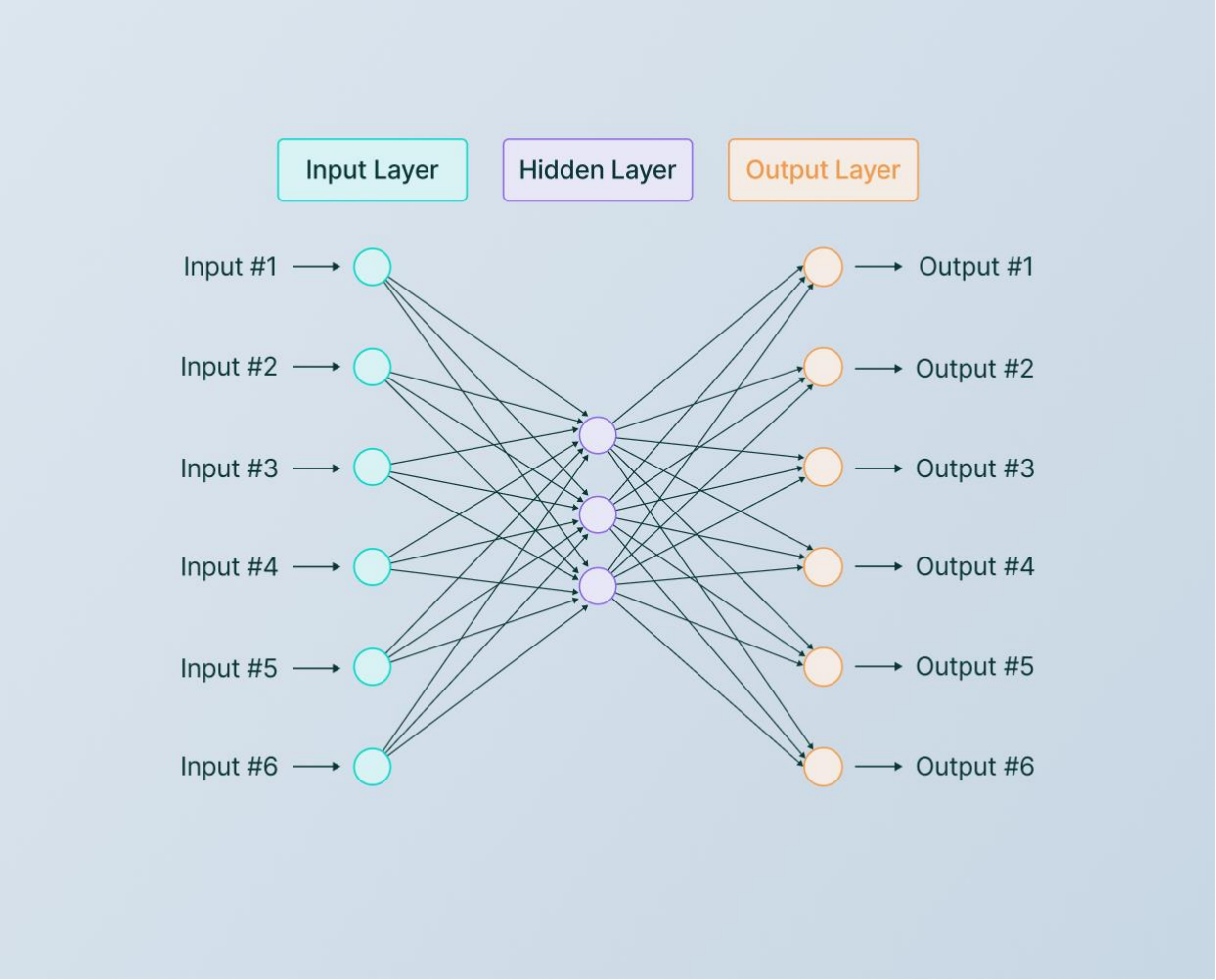
U radu je korištena relu aktivacijsku funkciju kod izrade svoje neuronske mreže:

```
public double reLu(double input) {
        if(input <= 0) {
            return 0;
        } else {
            return input;
        }
    }
```

Kod 1.1: RELU funkcija

1.3. Arhitektura neuronske mreže

Neuronske mreže složene su strukture napravljene od umjetnih neurona koji mogu primiti višestruke ulaze da bi proizveli više izlaza. Ovo je primarni posao neuronske mreže – pretvoriti ulaz u smislen izlaz. Obično se neuronska mreža sastoji od ulaznog i izlaznog sloja s jednim ili više skrivenih slojeva između njih. Također je poznata kao umjetna neuronska mreža arhitektura u neuronskoj mreži funkcionira poput ljudskog mozga i vrlo je važna.

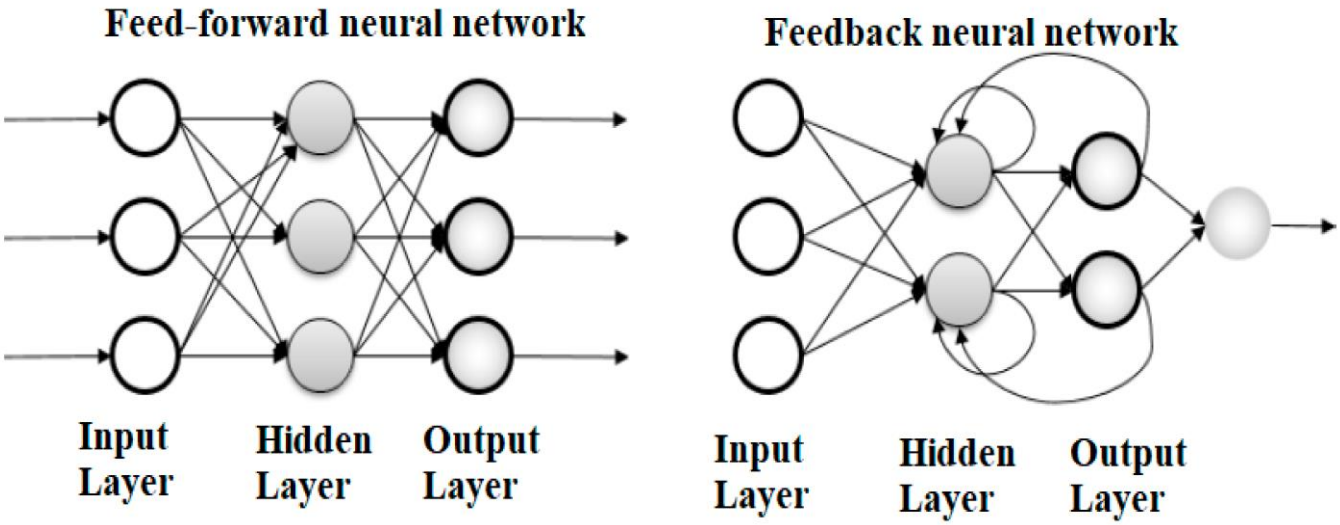


Slika 1.4: Prikaz slojeva neuronske mreže

U neuronskoj mreži svi neuroni utječu jedni na druge i stoga su svi povezani. Mreža može prepoznati i promatrati svaki aspekt skupa podataka koji je pri ruci i kako se različiti dijelovi podataka mogu ili ne moraju međusobno odnositi. Ovo je način na koji su neuronske mreže sposobne pronaći iznimno složene obrasce u golemim količinama podataka.

U neuronskoj mreži protok informacija odvija se na dva načina (slika 1.5):

- Feedforward mreže: U ovom modelu signali putuju samo u jednom smjeru, prema izlaznom sloju. Feedforward mreže imaju ulazni sloj i jedan izlazni sloj s nula ili više skrivenih slojeva. Naširoko se koriste u prepoznavanju uzoraka.
- Mreže s povratnom spregom: U ovom modelu interaktivne mreže koriste svoje unutarnje stanje (memoriju) za obradu slijeda ulaza. U njima signali mogu putovati u oba smjera kroz petlje (skrivene slojeve) u mreži. Obično se koriste u vremenskim serijama i sekvencijalnim zadacima.



Slika 1.5: Prikaz protoka informacija

Na slici 1.4, vanjski zeleno-plavi sloj je ulazni sloj. Neuron je osnovna jedinica neuronske mreže. Oni primaju podatke iz vanjskog izvora ili drugih čvorova. Svaki čvor je povezan s drugim čvorom iz sljedećeg sloja, a svaka takva veza ima određenu težinu. Težine se dodjeljuju neuronu na temelju njegove relativne važnosti u odnosu na druge ulaze.

Kada se sve vrijednosti čvorova iz zeleno-plavog sloja pomnože (zajedno s njihovom težinom) i sažmu, generira se vrijednost za prvi skriveni sloj. Na temelju sažete vrijednosti, plavi sloj ima unaprijed definiranu funkciju "aktivacije" koja određuje hoće li ovaj čvor biti "aktiviran" i koliko će biti "aktivan".

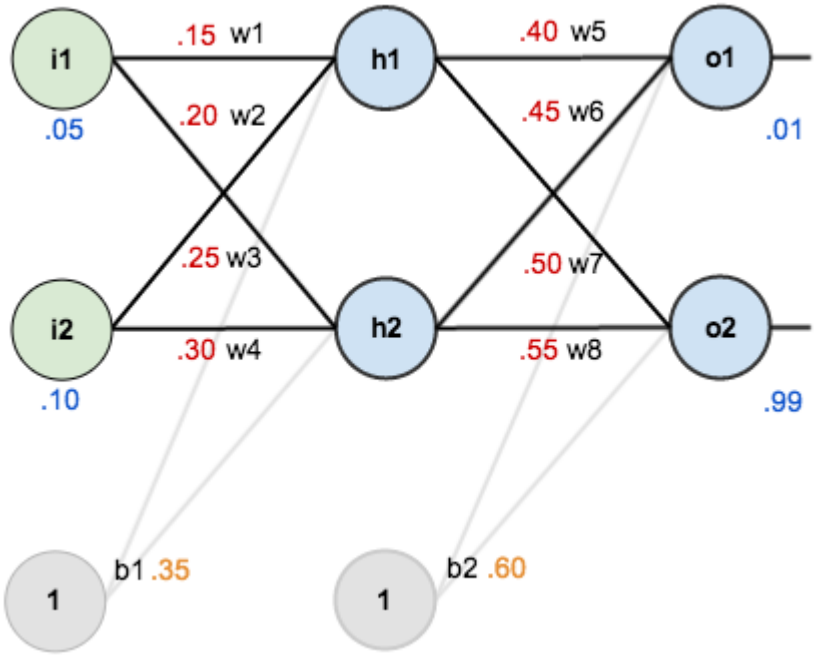
Sloj ili slojevi skriveni između ulaznog i izlaznog sloja poznati su kao skriveni sloj. Nazivaju se skrivenim slojem jer je uvijek skriven od vanjskog svijeta. Glavno računanje neuronske mreže odvija se u skrivenim slojevima. Dakle, skriveni sloj preuzima sve ulaze iz ulaznog sloja i izvodi potrebne izračune za generiranje rezultata. Taj se rezultat zatim prosljeđuje izlaznom sloju kako bi korisnik mogao vidjeti rezultat izračuna.

1.4. Učenje neuronskih mreža

1.4.1. Backpropagation

Promatranje analogije može biti korisno u razumijevanju mehanizama neuronske mreže. Učenje u neuronskoj mreži usko je povezano s načinom na koji učimo u našim uobičajenim životima i aktivnostima - izvodimo radnju i ona je dobro napravljena ili nas trener ispravlja ili treniramo kako bismo razumjeli kako postati bolji u određenom zadatku. Slično tome, neuronske mreže zahtijevaju trenera kako bi opisali što je trebalo proizvesti kao odgovor na unos. Na temelju razlike između stvarne vrijednosti i vrijednosti koju je izračunala mreža, izračunava se vrijednost pogreške i šalje natrag kroz sustav. Za svaki sloj mreže, vrijednost pogreške analizira se i koristi za podešavanje praga i težine za sljedeći unos. Na taj način, pogreška postaje marginalno manja svakim izvođenjem kako mreža uči kako analizirati vrijednosti.

Gore opisani postupak poznat je kao backpropagation (slika 1.6) i kontinuirano se primjenjuje kroz mrežu dok se vrijednost pogreške ne dovede do minimuma. U ovom trenutku neuronska mreža više ne zahtijeva takav proces obuke i dopušteno joj je raditi bez prilagodbi. Mreža se tada konačno može primijeniti, koristeći prilagođene težine i pragove



kao smjernice.

Slika 1.6: Backpropagation

Kada neuronska mreža aktivno radi, ne dolazi do povratnog širenja jer ne postoji način za izravnu provjeru očekivanog odgovora. Umjesto toga, valjanost izlaznih naredbi ispravlja se tijekom nove sesije obuke ili se ostavlja kakva jest da mreža radi. Možda će trebati napraviti mnoge prilagodbe jer se mreža sastoji od velike količine varijabli koje moraju biti precizne kako bi umjetna neuronska mreža funkcionirala. Osnovni primjer takvog procesa može se ispitati učenjem neuronske mreže da pretvori tekst u govor.

Mreže poput mreži za prepoznavanje govora mogu biti održivi modeli za veliki niz matematičkih i statističkih problema, uključujući ali ne ograničavajući se na sintezu i prepoznavanje govora, prepoznavanje lica i predviđanje, nelinearno modeliranje sustava i klasifikaciju uzoraka.

Problem učenja predstavljamo u smislu minimizacije funkcije gubitka (f). Ovdje je "f" funkcija koja mjeri izvedbu neuronske mreže na danom skupu podataka. Općenito, indeks gubitka sastoji se od izraza pogreške i izraza za regulaciju. Dok pojam pogreške procjenjuje kako se neuronska mreža uklapa u skup podataka, pojam regulacije pomaže spriječiti problem prekomjernog prilagođavanja kontroliranjem efektivne složenosti neuronske mreže.

Funkcija gubitka ovisi o adaptivnim parametrima – težinama i pristranostima – neuronske mreže. Ti se parametri mogu grupirati u jedan n-dimenzionalni vektor težine.

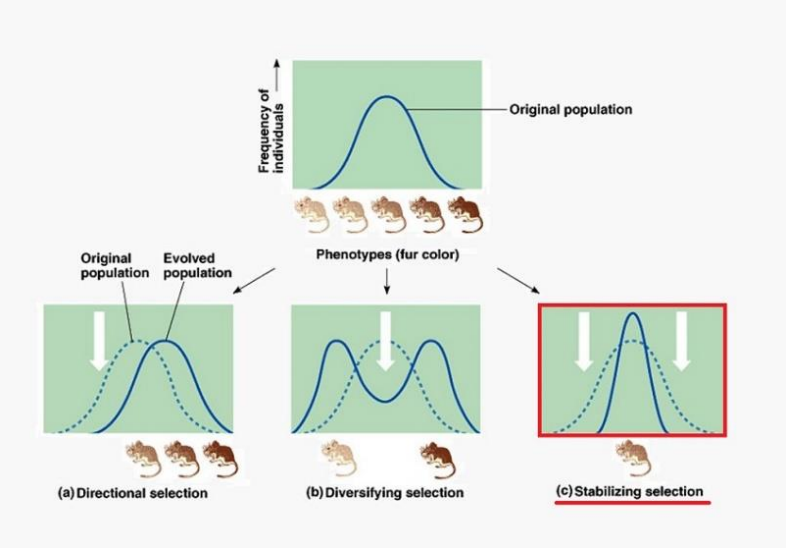
```
public void backPropagation(double[] dLdO) {
        double[] dLdx = new double[inLenght];
        double dOdz;
        double dzdw;
        double dLdw;
        double dzdx;
        for (int k = 0; k < inLenght; k++) {
            double dLdXSum = 0;
            for (int j = 0; j < outLenght; j++) {
                dOdz = derivationReLu(lastZ[j]);
                dzdw = lastX[k];
                dzdx = weight[k][j];
                dLdw = dLdO[j] * dOdz * dzdw;
                weight[k][j] -= dLdw * learningRate;
                dLdXSum += dLdO[j] * dOdz * dzdx;
```

```
            }
            dLdx[k] = dLdXSum;
        }
        if (prevLayer != null) {
            prevLayer.backPropagation(dLdx);
        }
    }
```

Kod 1.2: Kod za backpropagation

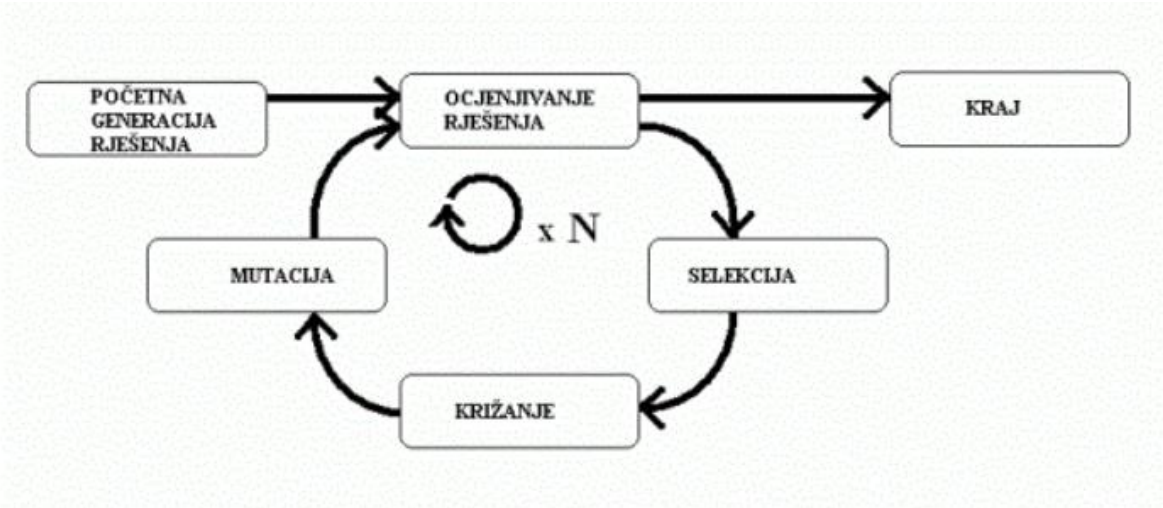
1.4.2. Genetski algoritam

Genetski algoritam način je rješavanja raznih problema poboljšavanja koji su ograničeni ili neograničeni na temelju prirodne selekcije, što je proces biološke evoluciju (slika 1.7). Genetski algoritam mnogo puta promjeni populaciju mogućih rješenja. U svim koracima genetski algoritam bira neke pojedince iz trenutne populacije da budu roditelji i koristi ih za stvaranje nove djece za buduću generaciju. S vremenom populacija "napreduje" do najučinkovitijeg rješenja. Genetski algoritmi mogu se koristiti za rješavanje raznih problema koji se ne rješavaju sa standardnim optimizacijskim algoritmima, primjerice, problemi s diskontinuiranim, nediferencijabilnim, stohastičkim ili visoko nelinearnim ciljevima. Genetski algoritmi sposobni su rješavati probleme mješovitog cjelobrojnog programiranja koji imaju komponente koje su ograničene na cijele brojeve.



Slika 1.7: Primjeri prirodne selekcije

Genetski algoritmi su obitelj računalnih algoritma koje mogu ponuditi alternativu povratnom širenju prilikom pronalaženja dobrog skupa težina u neuronskoj mreži. Genetski algoritam kodira potencijalno rješenje problema (fenotip) u podatkovnoj strukturi sličnoj kromosomu koja se naziva genotip ili genom. Tradicionalno, genetski algoritam stvara početnu populaciju (obično nasumičnih) genoma, koji se materijaliziraju kao fenotip i procijenjena na temelju neke funkcije dobrote (slika 1.8). Ti genomi koji predstavljaju bolja rješenja za problem koji je pri ruci dane su mogućnosti poboljšavanja ostalih lošijih genoma, stvarajući genome za sljedeću generaciju sličnije njima. Genomi također su podvrgnuti mutaciji kako bi se osigurala genetska raznolikost iz jedne populacije na sljedeću (analogno biološkoj mutaciji). Kao heuristika pretraživanja, genetski algoritmi imaju neke prednosti u odnosu na druge oblike traženja. Na primjer, dok metode gradijentnog spusta mogu dobiti zarobljeni u lokalnim minimumima na površini pogreške, genetski algoritmi to izbjegavaju zamku uzorkovanjem više točaka na površini pogreške. Nadalje, genetski algoritmi zahtijevaju vrlo malo apriornog znanja o domeni problema.



Slika 1.8: Shema genetskog algoritma

Kao što je ranije spomenuto, arhitektura neuronske mreže i algoritam učenja koji se koristi za treniranje modela važni su za dobivanje najboljeg rješenje. Neuroevolucija je metoda koja kombinira principe evolucije i neuronskih mreža kako bi se razvile optimalne arhitekture mreža za rješavanje složenih problema, koristeći genetske algoritme za optimizaciju. Ova tehnika omogućava evoluciju i prilagodbu neuronskih mreža putem selekcije, križanja i mutacije, kako bi se postigla što bolja performansa u zadatku.

1.4.3. Usporedba backpropagationa i genetskog algoritma

Backpropagation i genetski algoritam su dva različita pristupa u strojnom učenju. Backpropagation je popularan algoritam za učenje neuronskih mreža koji se temelji na gradijentnom spuštanju. Koristi se za prilagodbu težina mreže kako bi se minimizirala pogreška između izlaza mreže i stvarnih vrijednosti. Backpropagation se koristi za super vizirano učenje i ima sposobnost prilagođavanja mreže na temelju povratnih informacija o pogrešci.

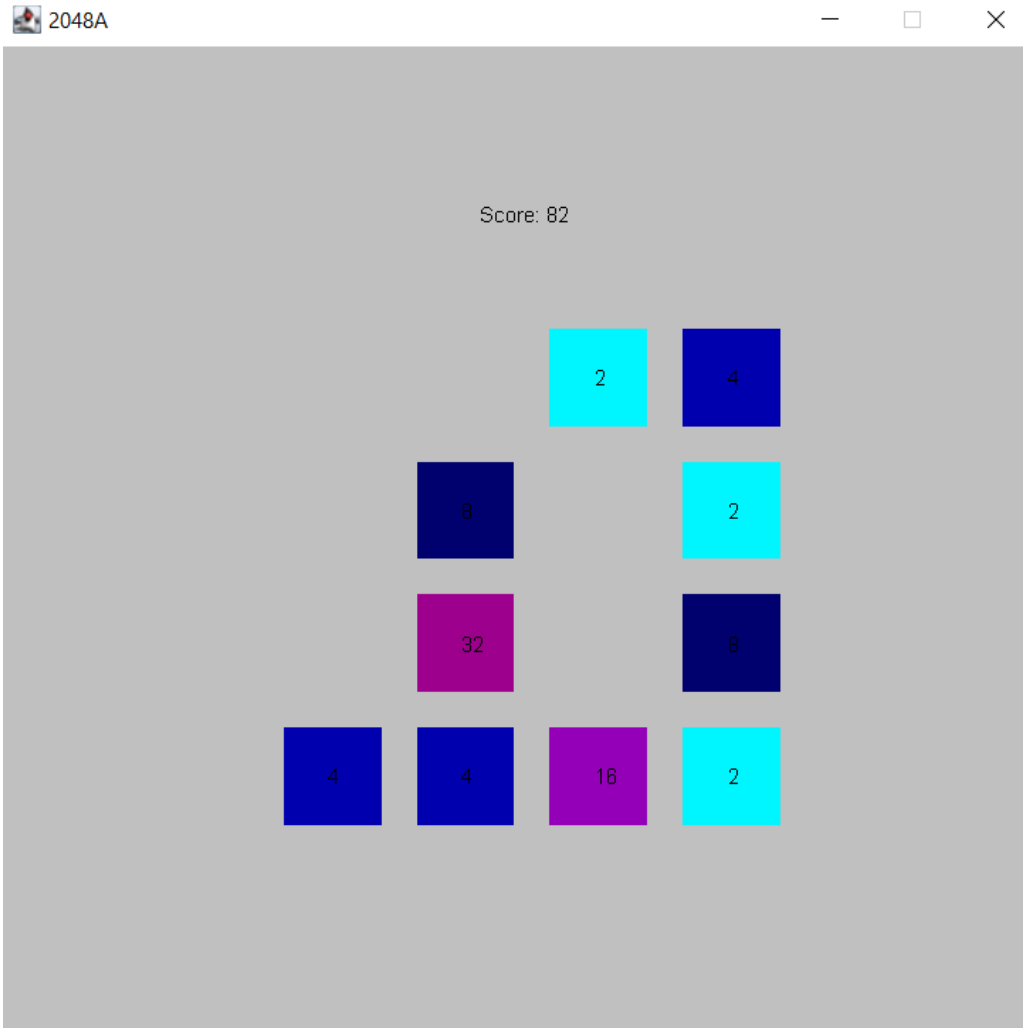
S druge strane, genetski algoritam je evolucijski algoritam inspiriran prirodnim procesima evolucije. Radi na principu selekcije, križanja i mutacije jedinki u populaciji kako bi se pronašla optimalna rješenja. Genetski algoritmi mogu se koristiti za rješavanje problema optimizacije u kojima se traže najbolje vrijednosti za određene parametre ili kombinaciju parametara.

Backpropagation se fokusira na lokalnu optimizaciju, tj. prilagođavanje parametara mreže kako bi se smanjila pogreška. Genetski algoritmi, s druge strane, provode globalno pretraživanje prostora rješenja putem evolucijskih operacija. Oni su pogodni za probleme s više opcija i za traženje dobre aproksimacije optimalnog rješenja.

Važno je napomenuti da su backpropagation i genetski algoritmi komplementarni te se često koriste zajedno u optimizaciji neuronskih mreža. Genetski algoritmi mogu se primijeniti za pretraživanje arhitekture mreže, odabir hiperparametara ili optimizaciju težina, dok backpropagation služi za treniranje samih težina mreže na temelju zadanih ulaza i očekivanih izlaza.

2. Programsko ostvarenje

2.1. Simulacijsko okruženje



Slika 2.1: Simulacijsko okruženje

Simulacijsko okruženje koje je prikazano na slici 2.1 sastoji se od pločica koje popunjavaju polje 4x4. Nove pločice nastaju nakon svakog poteza. Prikazuju se na nasumičnom praznom polju. Kada napravimo potez jedine mogućnosti za nastanak su pločice s brojevima 2 i 4 s postotkom nastajanja 70% naspram 30%. Igru igramo sve dok se sva polja nisu popunila.

Mogući potezi u okruženju:

- gore
- dolje
- lijevo
- desno.

Odabirom poteza sve pločice pomiču se u tom smjeru sve dok ne dođu do neke druge pločice različite vrijednosti ili ruba polja 4x4. Tada ostaju na tom polju i čekaju dok sve pločice ne miruju na svom novom mjestu.

Kada odaberemo jedno od mogućih poteza ako se pločice istog broja (primjerice dvije dvojke) dotaknu one postaju nova pločica koja nosi vrijednost zbroja te dvije pločice (dvije dvojke daju četvorku). Ako potez traje, a nastala je nova pločica koja je jednaka kao pločica pored nje u tom smjeru ona će se nastaviti povezivati i s tom pločicom te će se na kraju poteza povezati 3 pločice u jednu.

Iznad 4x4 polja nalazi se ukupni rezultat koji se dobiva tako što se zbroje sve vrijednosti pločica. Cilj u igri je osvojiti što veći ukupni rezultat. Njegova vrijednost najvažnija je informacija pri učenju neuronskih mreža. To je glavni indikator jesmo li igru dobro odigrali ili ne.

Svaka pločica drugog broja različite je boje kako bi se jednostavnije razlikovale i jednostavnije znalo koje se dvije pločice mogu spajati a koje ne.

Igricu je moguće pokrenuti i bez neuronske mreže te ju igrati kao običan igrač. Potezi su intuitivni: strelica gore (gore), strelica dolje (dolje), strelica lijevo (lijevo) i strelica desno (desno). Strelice se moraju držati samo jedna u jednome trenutku sve dok se sve pločice ne zaustave na svojem novom mjestu. Ako bi držali više strelica u isto vrijeme doći će do neželjenog i nepredviđenog kretanja koje može izazvati razne probleme koji mogu spriječiti daljnje igranje igre na način na koji je osmišljena.

```
            while (kontrola == 1) {
                            kontrola = 0;
                            int rbrX = r.nextInt(0, 4);
                            int rbrY = r.nextInt(0, 4);
                            for (int i=0;i<handler.object.size(); i++) {
                                GameObject tempObject =
            handler.object.get(i);
                                if (tempObject.x == 150 + rbrX * 75 &&
            tempObject.y == 150 + rbrY * 75) {
                                    kontrola = 1;
                                }
                            }
                            if (kontrola == 0) {
                                int posto = r.nextInt(1, 11);
                                int plbr;
                                if (posto > 7) {
                                    plbr = 4;
                                } else {
                                    plbr = 2;
                                }
                                String ime = "player " +
            String.valueOf(plbr);
                                if (pomaknuto == 1) {
                                    handler.addObject(new Player(150 +
            rbrX * 75, 150 + rbrY * 75, ID.getOperationName(ime),
            handler));
                                }
}
```

Kod 2.1: Dodavanje pločica

Kada nastaje nova pločica nakon odigranog poteza program odabire nasumično polje te provjerava je li je već zauzeto ili je moguće staviti novu pločicu. Ako prije poteza nije niti došlo do pomaka tada odmah znamo da su sva polja popunjena te završavamo s programom. Nakon što se provjeri je li odabrano polje već zauzeto, program će ponovno birati polje sve dok ne odabere slobodno polje. Nakon što odabere slobodno polje, ta pločica se dodaje u odgovarajući "handler" koji je zadužen za prikaz te pločice na našem sučelju.

2.2. Tehnologije

Za izradu rada koristi se IntelliJ. IntelliJ IDEA je integrirano razvojno okruženje (IDE) za programiranje. IntelliJIdea nudi bogat skup značajki, uključujući inteligentno automatsko dovršavanje koda, refaktoriranje koda, različite alate za testiranje i profiliranje, integraciju s sustavima za upravljanje verzijama, podršku za razvoj web aplikacija i mobilnih aplikacija, te mnoge druge korisne značajke. Aplikacija podržava različite programski jezike:

- Java
- Kotlin
- JavaScript
- TypeScript
- SQL
- druge.

2.2.1. Java

Programski jezik koji se koristi za izradu rada je Java. Java je objektno orijentirani programski jezik. Java je dizajnirana tako da bude jednostavna, sigurna i platformski neovisna. Programi napisani u Javi se prevode u bytecode koji se može izvoditi na bilo kojoj platformi koja podržava virtualnu mašinu Java (JVM).

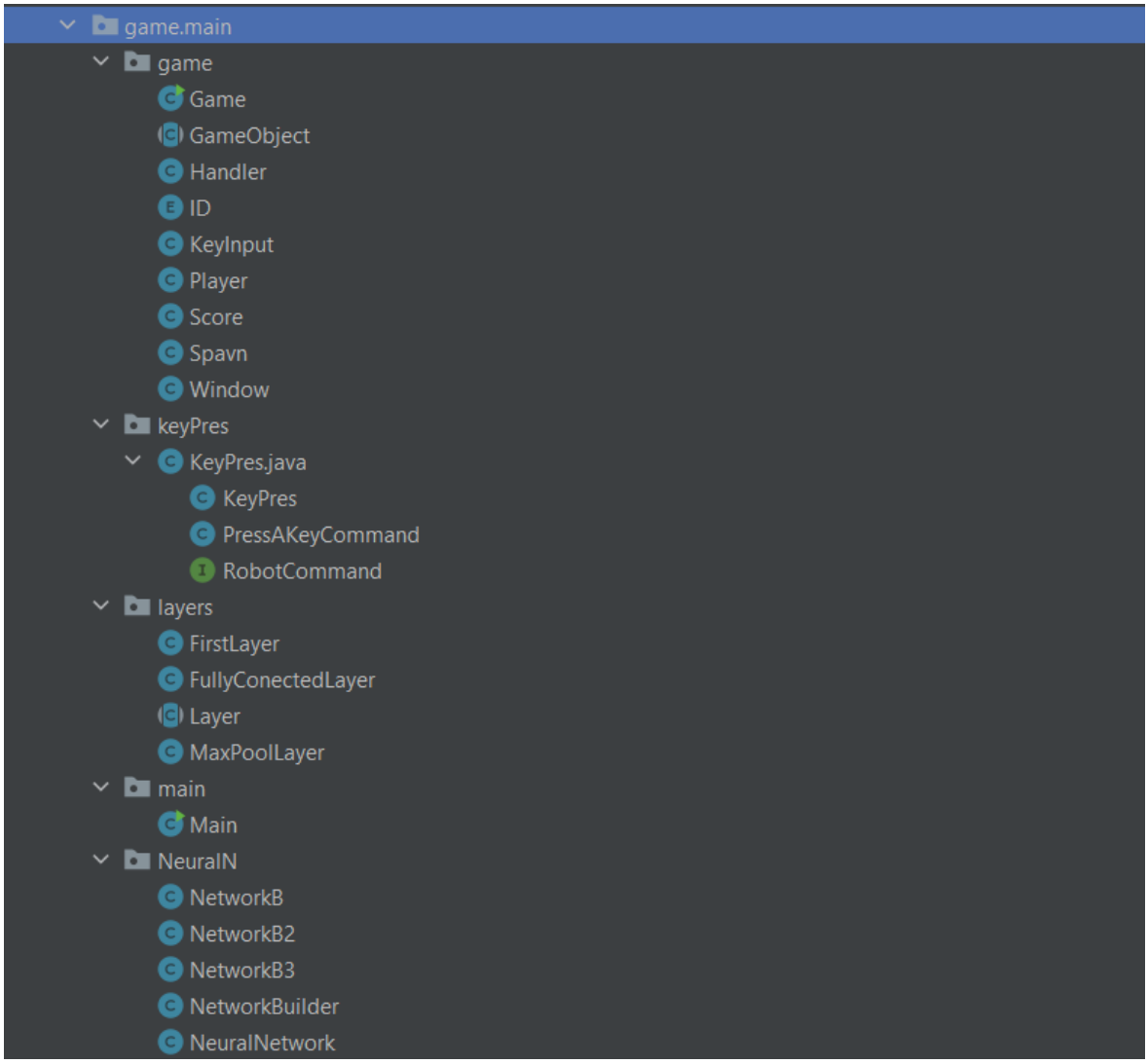
Java ima široku primjenu u razvoju web i mobilnih aplikacija, igara, financijskih aplikacija i druge programske opreme. Javina sintaksa je jednostavna i lako se uči, a bogata biblioteka klasa i okvira olakšava programiranje. Java je također poznata po svojoj sigurnosti i stabilnosti. Javina virtualna mašina ima ugrađene mehanizme za upravljanje memorijom i kontrolu iznimki što olakšava rad programera i smanjuje mogućnost grešaka u kodu.

Java je popularan programski jezik koji se koristi u različitim aplikacijama, od weba do mobilnih aplikacija i financijskih sustava, zbog svoje jednostavnosti, sigurnosti i platformski neovisne prirode. Iz ovih razloga ja sam odlučio napraviti svoju neuronsku mrežu iz nule u Javi.

2.2.2. Struktura

Kod je podijeljen u više razreda koji su povezani kako bi bila bolja preglednost koda i snalaženje u njemu (slika 2.2). Neki od bitnijih razreda su:

- Games – pokreće igricu
- Player – funkcije pločica
- KeyPres – samostalno odigravanje poteza
- FullyConectedLayer – sloj u neuronskoj mreži
- Main – pokreće neuronsku mrežu i igricu
- NeuralNetwork – spaja slojeve i radi neuronsku mrežu
- NetworkB3 – upravlja s genetskim algoritmom i učenjem neuronske mreže



Slika 2.2: Prikaz svih razreda

2.3. Razred „Games“

Razred *Games* poziva se za pokretanje simulacijskog okruženja. Kada njega pokrenemo igru možemo igrati kao običan igrač bez ostalih algoritama. On omogućuje konstantno pokretanje novih igara i njihovo prikazivanje. Pozivanjem ovog razreda pozivamo i sve ostale razrede potrebne igranje igrice kao što su *Handler*, *Spavner*, *Score* i *Window*.

2.4. Razred „Player“

Razred *Player* omogućuje sposobnosti spajanja i promjene pločica. Neke od funkcija koje to rade su:

- *collision()*
- *vratiVrstu(ID id)*
- *kojiGdje(GameObject prvi, GameObject drugi).*

Funkcija *collision()* prati pomicanje pločica te gleda ako se dvije pločice diraju. Ako su one iste vrijednosti stvaramo novu duplo veću pločicu na mjestu spoja i uklanjamo dvije manje od koje je nastala. Ako se pločice koje nisu iste vrijednosti diraju moramo ih razdvojit nakon provjere jednakosti na njihovu namijenjenu poziciju.

Funkcija *vratiVrstu(ID id)* vraća informaciju o ID-u pločice koje mogu biti primjerice player 2, player 3, player 256… ID označava vrijednost pločice.

Funkcija *kojiGdje(GameObject prvi, GameObject drugi)* korištena je u funkciji *collision()* pri odluci kada su pločice različite vrijednosti i moramo ih razdvojiti nakon dodira. Tada funkcijom odlučujemo poziciju svake pločice.

2.5. Razred „KeyPres“

Ovaj razred omogućuje samostalno pritiskanje pokreta:

- Gore
- Dolje
- Lijevo
- Desno.

Izrađen je tako što ne samo da se tipke pritisnu one se drže određeno vrijeme koje je potrebno da se pločice na rubovima stignu pomaknuti s jednog kraja polja do drugog. To je ostvareno uz pomoć *while* petlje koja provjerava jesu li sve pločice na svojim mjestima te drži tipku sve dok se one kreću.

```
Robot robot = new Robot();
while (isContinue) {
     robot.keyPress(KeyEvent.VK_UP);
}
robot.keyRelease(KeyEvent.VK_UP);
```

Kod 2.2: Prikaz petlje za potez gore

2.6. Razred „FullyConectedLayer“

Potpuno povezani sloj (Fully connected layer) je osnovni tip sloja koji se koristi u neuronskim mrežama. Ovaj sloj sastoji se od skupa neurona koji su povezani sa svim neuronima u prethodnom sloju, te svaki neuron prima ulaze od svih neurona u prethodnom sloju i daje izlaze svim neuronima u sljedećem sloju.

Svaki neuron u potpuno povezanom sloju množi težine s ulazima, zbroji ih i primjenjuje aktivacijsku funkciju na rezultat kako bi se generirao izlaz. Težine u sloju prilagođavaju se tijekom treninga kako bi se minimizirala pogreška mreže.

Potpuno povezani sloj često se koristi kao zadnji sloj u neuronskim mrežama za klasifikaciju, gdje svaki neuron u sloju predstavlja jednu klasu koju neuronska mreža može prepoznati. Ovaj sloj također se može koristiti kao skriveni sloj u neuronskim mrežama, gdje služi kao detektor značajki koje se koriste za prepoznavanje uzoraka u podacima.

Funkcija fullyConnectedForwardPass() u fully connected layer-u neuronske mreže izvodi prosljeđivanje podataka (engl. *forward pass*) kroz sloj. Funkcija prima ulaze koji su izlazi prethodnog sloja i parametre sloja koji se koriste za izračun izlaza.

Funkcija počinje s matričnim množenjem ulaza i težina sloja, pri čemu se svaki ulaz množi sa svojom težinom kako bi se generirao vektor skalarnih umnožaka. Zatim se vektor skalarnih umnožaka zbroji s vektorom iznosa pristranosti kako bi se dobili neto ulazi neurona.

Nakon toga primjenjuje se aktivacijska funkcija na neto ulaze kako bi se dobili izlazi neurona. Ovisno o tipu problema, može se koristiti različite aktivacijske funkcije, poput ReLU, sigmoidne funkcije, ili softmax funkcije. U radu je korištena ReLu funkciju.

Funkcija fullyConnectedForwardPass() vraća izlazni vektor neurona koji se šalje u sljedeći sloj neuronske mreže kao ulaz. Ovaj postupak ponavlja se za svaki primjer u skupu podataka koji se obrađuje.

```
public double[] fullyConnectedForwardPass(double[] input) {
        lastX = input;
        double[] z = new double[outLenght];
        double[] out = new double[outLenght];

        for (int i = 0; i < inLenght; i++) {
            for (int j = 0; j < outLenght; j++) {
                z[j] += input[i] * weight[i][j];
            }
        }

        lastZ = z;
        for (int i = 0; i < inLenght; i++) {
            for (int j = 0; j < outLenght; j++) {
                out[j] = reLu(z[j]);
            }
        }
        return out;
    }
```

Kod 2.3: FullyConnectedForwardPass()

2.7. Razred „Main“

Razred *Main* pokreće cijeli program pozivajući druge razrede. Poziva razrede koje uče i pokreću neuronsku mrežu skupa s igricom.

2.8. Razred „NeuralNetwork“

U razredu *NeuralNetwork* povezujemo slojeve u jednu veliku listu koja čini neuronsku mrežu. Funkcija koja povezuje slojeve naziva se *linkLayers()*.

```
private void linkLayers() {
        if (layers.size() <= 1) {
            return;
        }
        for (int i = 0; i < layers.size(); i++) {
            if (i == 0) {
                layers.get(i).setNextLayer(layers.get(i +
1));
            } else if (i == layers.size() - 1) {
                layers.get(i).setPrevLayer(layers.get(i -
1));
            } else {
                layers.get(i).setNextLayer(layers.get(i +
1));
                layers.get(i).setPrevLayer(layers.get(i -
1));
            }
        }
    }
```

Kod 2.4: Povezivanje slojeva u neuronsku mrežu

Također u ovom razredu imamo funkciju *train(double[] input, double[] netaknut, int g)* koja trenira neuronsku mrežu po backpropagation algoritmu tako što pretpostavlja najbolji potez i uspoređuje ga s trenutno odigranim potezom.

2.9. Razred „NetworkB3“

Glavna uloga razreda NetworkB3 je ta da uči neuronsku mrežu uz pomoć genetskog algoritma. To radi tako što prvo odredimo količinu neuronskih mreža koje želimo koristiti za učenje i koliko puta ćemo provesti algoritam. Svaki put kada algoritam krene uzima jednu po jednu neuronsku mrežu i kombinira ju s nekom drugom nasumično odabranom mrežom na način ga provjeri koja od mreža je bila uspješnija i njene težine uzima više u obzir pri spajanju. Težine bolje mreže množimo s 1.2, a težine lošije mreže s 0.8 i onda se radi njihov prosjek. Genetski algoritam se provodi određeni broj puta te u svakom koraku proizvodi sve kvalitetnije mreže.

Ulazi u neuronsku mrežu su (ima ih 16):

1. Broj najvećih pločica gornjeg poteza
2. Suma praznih polja gornjeg poteza
3. Najveća pločica gornjeg poteza
4. Broj pločica vrijednosti 2 u odnosu na gornju stranu praznih polja
5. Broj najvećih pločica donjeg poteza
6. Suma praznih polja donjeg poteza
7. Najveća pločica donjeg poteza
8. Broj pločica vrijednosti 2 u odnosu na donju stranu praznih polja
9. Broj najvećih pločica lijevog poteza
10. Suma praznih polja lijevog poteza
11. Najveća pločica lijevog poteza
12. Broj pločica vrijednosti 2 u odnosu na lijevu stranu praznih polja
13. Broj najvećih pločica desnog poteza
14. Suma praznih polja desnog poteza
15. Najveća pločica desnog poteza
16. Broj pločica vrijednosti 2 u odnosu na desnu stranu praznih polja

Predviđanje položaja pločica idućeg poteza dobivamo uz pomoć matrica koje prikazuju gdje će se pločice nalaziti.

Prvi, peti, deveti i trinaesti ulaz prebrojavaju koliko je najvećih vrijednosti pločica. To radi tako što nađemo najveću vrijednost i prebrojim koliko ima takvih istih pločica.

Drugi, šesti, deseti i četrnaesti ulaz prebrojavaju polja u matrici jednaka nuli.

Treći, sedmi, jedanaesti i petnaesti ulaz traži najveću vrijednost pločica.

Četvrti, osmi, dvanaesti i šesnaesti ulaz pomažu nam oko nove nasumične pločice koja će 70% biti vrijednosti 2. Provjeravamo u kojim sve praznim poljima je mogućnost da se nova pločica može spojiti u potezu nakon toga.

U ovom razredu također provjeravamo je li moguć potez koji je neuronska mreža odlučila napraviti, jer ako potez ne pomiče niti jednu pločicu on nije dozvoljen. Tada moramo odlučiti jedan od ostalih poteza koji je moguć.

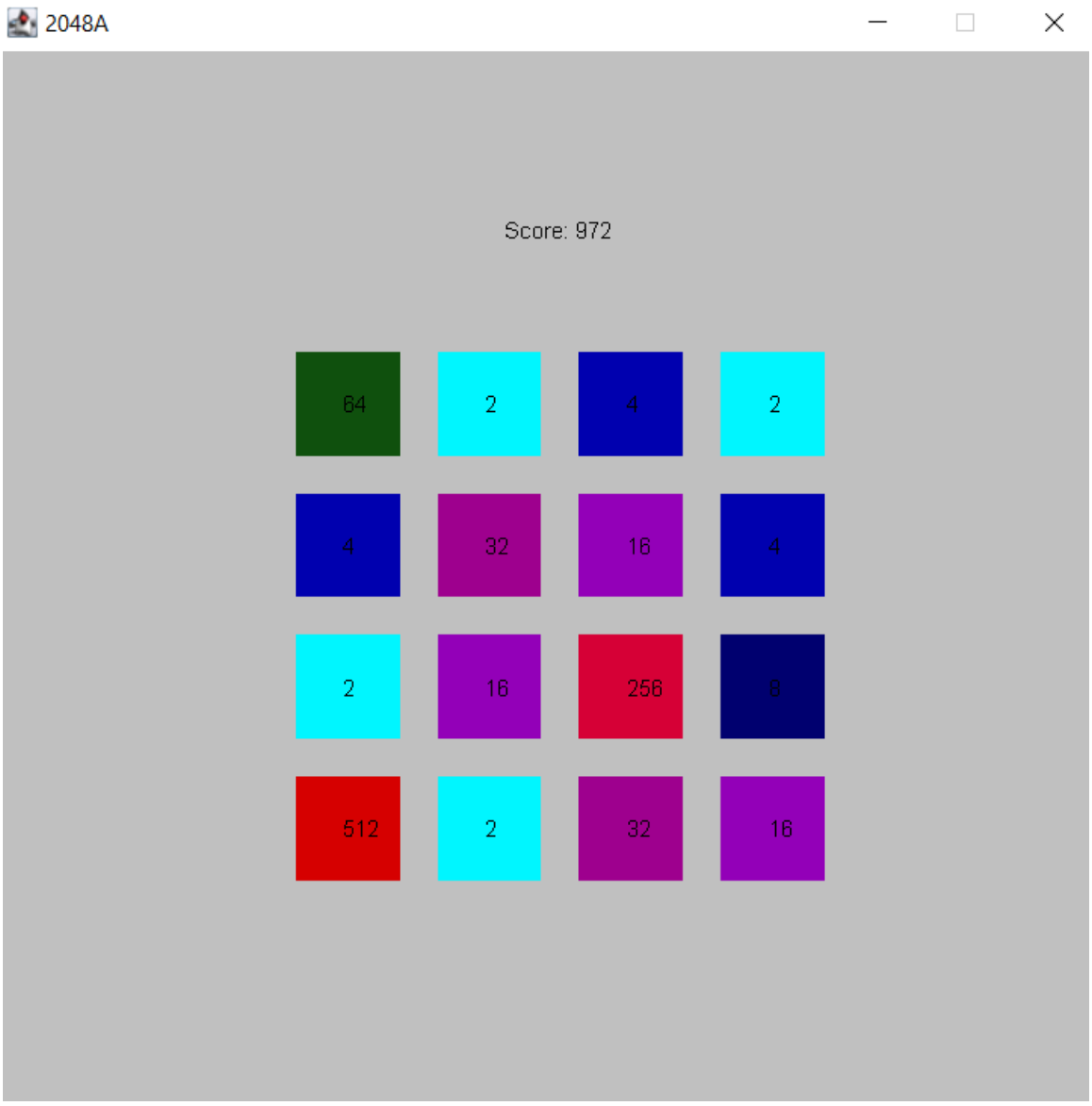
Nakon svake odigrane igre neuronske mreže zapisujemo u .txt datoteku kako bi poslije ponovno mogli pokrenuti neke od najbolji neuronskih mreža bez cijelog postupka genetskog algoritma.

Postupak pokretanja neke od najboljih neuronskih mreža izvodi se funkcijom *startNeuronska()* tako što u .txt datoteci red prije željene neuronske mreže stavimo znak #.

3. Rezultati

3.1. Parametri

Kriterij koji pratimo za provjeru uspješnosti neuronske mreže ukupni je rezultat pločica na kraju igre. On je dovoljan pokazatelj kvalitete rada neuronske mreže. Najbolji uspjeh igre bez neuronske mreže bio je 528, a naučena neuronska mreža došla je i preko 1200 što je velika razlika i prikaz uspješnosti te mreže (slika 3.1).



Slika 3.1: Prikaz uspjeha jedne uspješnije neuronske mreže

3.2. Skup za testiranje backpropagation algoritma

Algoritam backpropagation nije bio uspješan jer ne može odrediti najbolji idući potez. Trebali bi odrediti samo jedan ispravan potez nakon svakog poteza što nije moguće jer često se događa zbog slučajnosti postavljanja pločica da je više od jednog poteza najbolji. Tada se algoritam zbuni i odabire jednu od mogućnosti što na kraju nije bio ispravan potez te krene učiti krivo. Zbog velike ovisnosti o nasumično postavljenim pločicama ovaj algoritam nije prelazio preko ukupnog rezultata 300 što je naspram genetskog algoritma puno lošije.

3.3. Skup za testiranje genetskog algoritma

Skup za testiranje izvodi se na ne toliko velikoj količini primjera jer bi inače jako dugo trajalo. Korišteno je 5 mreža koje kombiniraju kroz genetski algoritam 10 puta. Učenje mreže traje duže jer svaki puta kada pokrenemo igricu mora postojati grafičko sučelje koje sprječava efikasnost i brzinu. I zbog male količine početnih mreža rezultati učenja mogu se jako razlikovati. Neki pokušaji krenu učiti već nakon prve iteracije algoritma jako dobro, dok nekima treba i do 5 iteracija algoritma da se počnu poboljšavati u pravom smjeru.

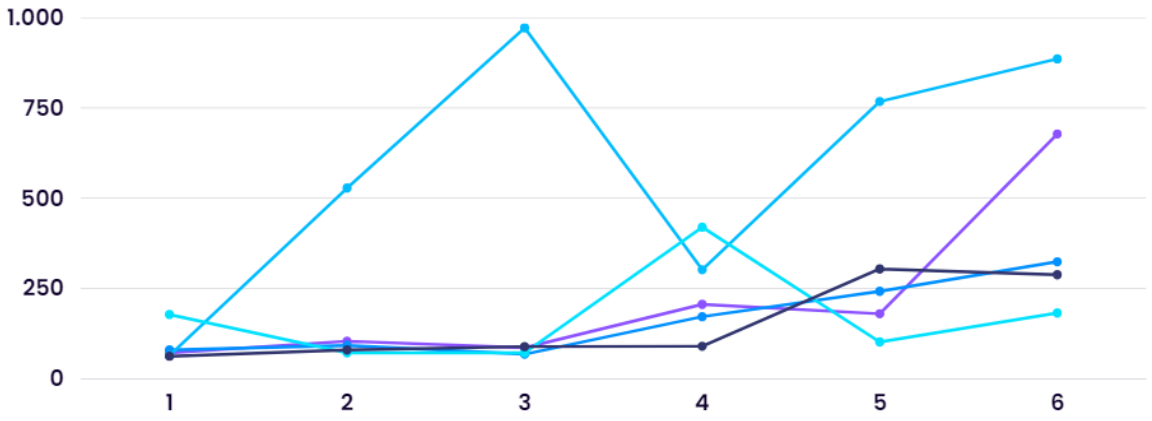
Najboljih 5 mreža koje sam dobio došle su do rezultata:

- 1244
- 1144
- 1008
- 972
- 964

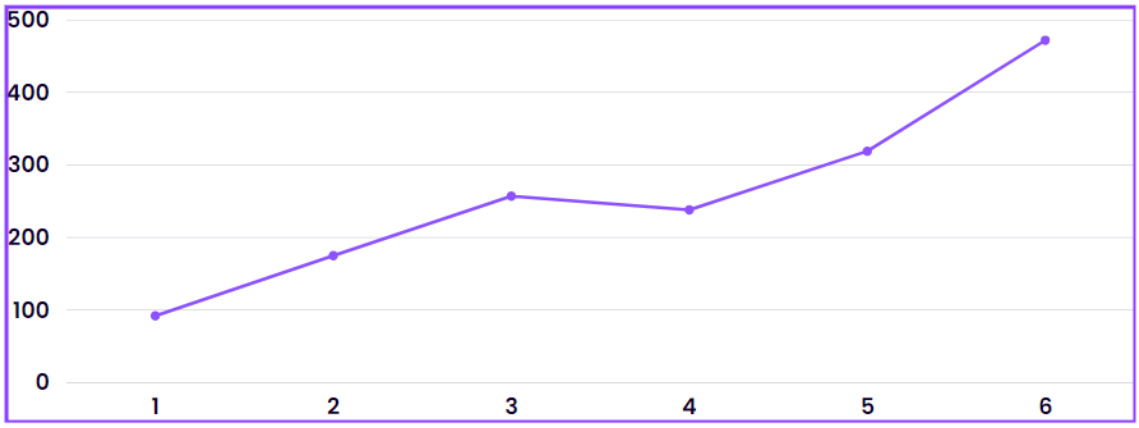
U tablici 3.1 i slikama 3.2 i 3.3 prikazano je jedno pokretanje algoritma genetskog učenja. U kojem možemo vidjeti napredak i poboljšanje neuronske mreže. Ovaj primjer dobro prikazuje kako nalaženjem jedne uspješnije neuronske mreže s vremenom poboljšava sve ostale te s vremenom i sebe učini stabilnijom i povjerljivijom.

	1.ite	2.ite	3.ite	4.ite	5.ite	6.ite
1.NM	72	104	86	206	180	678
2.NM	80	92	68	172	242	324
3.NM	64	528	972	302	768	886
4.NM	178	72	72	420	102	182
5.NM	62	80	90	90	304	288
Prosjek	91	175.2	257.4	238	319.2	472

Tablica 3.1: Tablica prikaza učenja neuronske mreže



Slika 3.2: Odnos svih neuronskih mreža



Slika 3.3: Prikaz prosjeka tokom algoritma

Zaključak

Ovaj rad o neuronskim mrežama treniranim na genetskom algoritmu je pokazao da genetski algoritam može biti uspješna metoda za treniranje neuronskih mreža u situacijama kada je teško pronaći optimalne vrijednosti parametara mreže korištenjem drugih metoda.

Genetski algoritam pokazao se kao metoda koja se dobro nosi sa složenim problemima, gdje je potrebno pronaći najbolju kombinaciju parametara mreže, a osim toga, omogućuje eksploraciju velikog prostora rješenja i potencijalno pronalaženje rješenja koja bi bila teško dostupna klasičnim metodama optimizacije.

Međutim, performanse neuronskih mreža treniranih genetskim algoritmom ovise o kvaliteti funkcije procjene koja se koristi za ocjenu kvalitete rješenja. Preciznija i adekvatnija funkcija procjene može rezultirati boljim performansama mreže. Također, genetski algoritam može biti spor u odnosu na druge metode treniranja neuronskih mreža, posebno u situacijama kada je skup podataka velik ili kada se koriste složene mreže.

Jedna od prednosti genetskog algoritma je mogućnost križanja, koja može biti korisna za kombiniranje dobrih rješenja iz različitih jedinki u populaciji, što može dovesti do poboljšanja performansi neuronskih mreža.

U zaključku, neuronske mreže trenirane genetskim algoritmom predstavljaju zanimljivu i moćnu metodu u području umjetne inteligencije, koja ima veliki potencijal za primjenu u različitim područjima, ali zahtijeva daljnje istraživanje i razvoj kako bi se postigle optimalne performanse.

Literatura

[1] Andries P. Engelbrecht, *Computational Nntelligence*. 2. izdanje.

[2] Neural Network https://www.ibm.com/topics/neural-networks

[3] Charu C. Aggarwal, Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edition

[4] David E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning 1st Edition

[5] Neural Network, https://www.techtarget.com/searchenterpriseai/definition/neural-network

[6] Golub M., Genetski Algoritam

Sažetak

Igranje igre 2048 korištenjem neuronskih mreža

Rad se fokusira na korištenje genetskog algoritma za treniranje neuronske mreže za igranje popularne igre 2048. Genetski algoritam koristi prirodnu selekciju, križanje i mutaciju kako bi generirao nove generacije mreža koje su sve bolje u igranju igre. Kroz iterativni proces, mreže uče strategije i pravila igre koje su potrebne za postizanje visokih bodova. Nakon što je mreža trenirana, testira se njezina sposobnost igranja igre i ocjenjuje se njezin uspjeh u usporedbi s drugim metodama igranja. Rezultati projekta pokazuju da genetski algoritam može biti uspješan u treniranju neuronske mreže za igranje 2048 igre.

Ključne riječi: neuronske mreže, genetski algoritam, strojno učenje, igra 2048

Summary

Playing the game 2048 using neural networks

The project focuses on using a genetic algorithm to train a neural network to play the popular game 2048. The genetic algorithm employs natural selection, crossover, and mutation to generate new generations of networks that get progressively better at playing the game. Through an iterative process, the networks learn strategies and rules of the game that are necessary to achieve high scores. Once the network is trained, its ability to play the game is tested, and its performance is evaluated in comparison to other playing methods. The project's results demonstrate that the genetic algorithm can be successful in training a neural network to play the 2048 game.

Keywords: neural networks, genetic algorithm, machine learning, 2048 game

Privitak

Instalacija programske podrške

1. Sadržaj se kopira na vlastito računalo te se pokreće iz razreda Main
2. Pokretanje običene igre pokreće se iz razreda Game

Upute za korištenje programske podrške

Dostupne su kratice na tipkovnici:

- Escape – izlazak iz programa
- Pritisak na prozor programa te pritisak 1 na tipkovnici
- 1 – pokretanje simulacije
- Strelica gore – potez prema gore
- Strelica dolje – potez prema dolje
- Strelica lijevo – potez prema lijevo
- Strelica desno – potez prema desno