SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 489

USPOREDBA METODA ZA UČENJE NEURONSKIH MREŽA ZA IGRU ZMIJA

Leo Goršić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 489

USPOREDBA METODA ZA UČENJE NEURONSKIH MREŽA ZA IGRU ZMIJA

Leo Goršić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 489

Pristupnik:	**Leo Goršić (0036523443)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Usporedba metoda za učenje neuronskih mreža za igru zmija**

Opis zadatka:

Proučiti umjetne neuronske mreže te različite optimizacijske metode bazirane na gradijentnom spustu i evolucijskim algoritmima koje se mogu primijeniti za optimizaciju njihovih parametara. Istražiti i predložiti prikladne arhitekture neuronske mreže te modele učenja za igranje igre zmija. Razviti programski okvir koji omogućuje učenje igraća za igru zmija različitim optimizacijskim metodama. Usporediti efikasnost igrača dobivenih različitim arhitekturama neuronskih mreža i algoritmima učenja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 10. lipnja 2022.

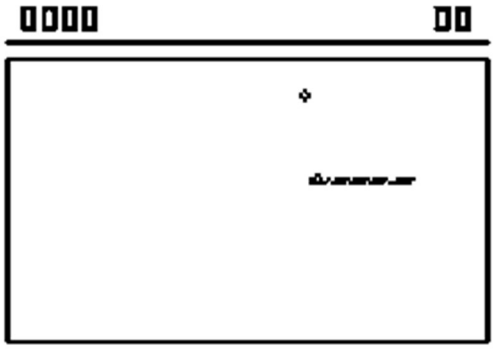
Zahvaljujem mentoru dr. sc. Marku Đuraseviću na pomoći i podršci tijekom izrade ovog rada

SADRŽAJ

1. UVOD

Igra Zmija popularna je arkadna videoigra. Datira još od 1970-ih godina, no veliku je popularnost stekla 2000-ih godina kada se standardno isporučivala uz mobilne telefone i računala. Na slici 1.1 i 1.2 mogu se vidjeti neke od implementacija. Igrač upravlja dugačkom tankom linijom koja predstavlja zmiju. Prostor po kojemu se zmija navigira sastoji se od ravnine omeđene zidovima i postavljene hrane. Cilj zmije je pojesti što više hrane te da se pri tome ne zabije u zidove ili u svoj rep. Svaki put kada zmija pojede hranu rep joj se produži i time manevriranje postaje sve teže. Igrač ne može zmiji zaustaviti kretanje niti ići unazad, a upravlja njome tako da bira smjer pomicanja njene glave (prve koordinate linije) koji može biti ravno, lijevo ili desno. Postoje razne implementacije ove igre, neke od njih čak i odstupaju od gore navedenih pravila, no ovaj rad razmatrati će igru opisanu po iznad navedenim pravilima.

Slika 1.1: Prikaz logotipa igre zmija na mobilnom telefonu nokia 3310 [13]



Slika 1.2: Prikaz ravnine po kojoj se zmija kreće kako je implementirano na mobilnom uređaju nokia 3310 [14]

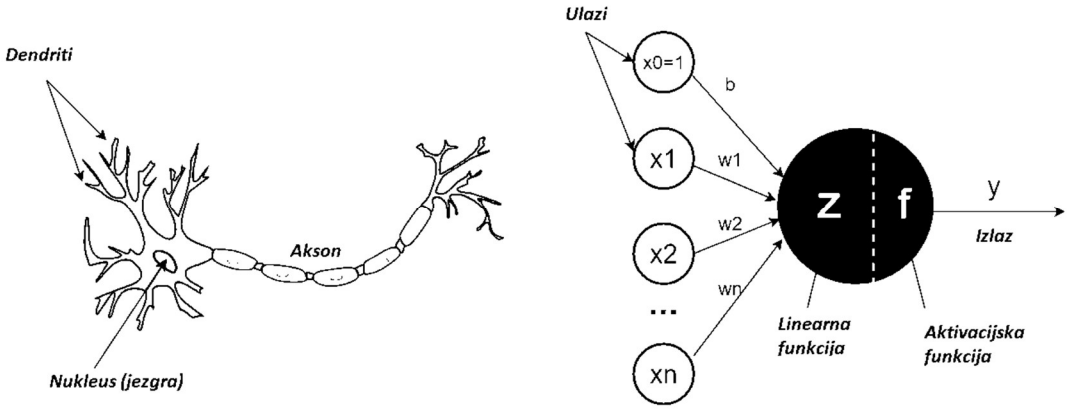
Ovakav jednostavni skup pravila čini ovu igru jako pogodnom za pokušaj implementacije agenta koji bi pomoću neuronske mreže upravljao zmijom. Obzirom na veliki skup različitih arhitektura neuronskih mreža kao i veliki skup različitih metoda za učenje neuronskih mreža u ovom radu razmatrat će se optimalni izbor arhitekture neuronske mreže, njenih parametara i metoda za učenje ovog problema. U idućem poglavlju rada dat će se uvod u neuronske mreže te kako mijenjanje parametara utječe na njihovo učenje. U trećem poglavlju dat ćemo osnovni pregled implementacije igre zmije kao poligon za učenje i testiranje različitih neuronskih mreža. U četvrtom poglavlju bit će prezentirani dobiveni rezultati, metodika učenja, testiranja i vrednovanja tih rezultata. Posljednje poglavlje iznijet će zaključak rada.

2. UMJETNA NEURONSKA MREŽA

„Neuronska mreža jest skup međusobno povezanih jednostavnih procesnih elemenata, *jedinica* ili *čvorova*, čija se funkcionalnost temelji na biološkom neuronu.“ [2]. Pri tome je važno naglasiti da postoje određena odstupanja umjetnog u odnosu na biološki neuron. Umjetni neuron bismo mogli tretirati kao pojednostavljenu analogiju biološkog neurona budući da nisu implementirani (ni modelirani) brojni fenomeni bioloških živčanih sustava. Moć umjetnih neuronskih mreža u odnosu na konvencionalne načine obrade podataka su prvenstveno sposobnost *učenja* odnosno obavljanje promjena nad samim sobom. Konvencionalna (simbolička) paradigma uvelike se oslanjala na postojanje algoritma i skupa pravila koji bi formalizirali zadani skup problema. Nažalost, gotovo svi svakodnevni problemi, primjerice raspoznavanje uzoraka, previše su teški za svođenje na skup znanih pravila ili algoritama. Stoga ćemo umjetne neuronske mreže koristiti prilikom obrade nejasnih ili manjkavih podataka. Osim toga neuronske mreže jako su prilagodljive te mogu raditi s velikim brojem ulaznih varijabli i parametara. Biološki neuroni u tijelu svoje stanice prikupljaju potencijale nekoliko tisuća susjednih neurona. Ako ukupni potencijal prijeđe određeni prag, neuron generira akcijski potencijal. Analogno tom ponašanju umjetni neuron će na ulazu dobiti numerički iznos signala koji množi sa težinskim faktorom (koji predstavlja jakost sinapse). Ako suma takvih signala prijeđe određeni prag neuron će dati izlazni signal [2].

Općenito izlazni signal neurona ovisit će o izlaznoj ili aktivacijskoj funkciji.

Slika 2.1 prikazuje na pojednostavljeni način analogiju umjetnog neurona sa biološkim.



Slika 2.1: Prikaz analogije biološkog i umjetnog neurona [12]

2.1. AKTIVACIJSKA FUNKCIJA

Promatramo li stanje neurona kao težinsku sumu svojih *n* ulaza $x_1 + \cdots + x_n$ tada je izlaz *a* ovisan o funkciji *g*:

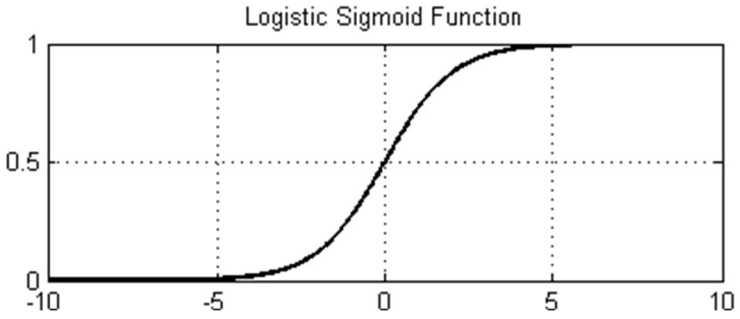
$$a = g(\omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n)$$

Funkciju *g* u ovom kontekstu nazivamo aktivacijskom ili prijenosnom funkcijom.
Aktivacijskih funkcija ima mnogo te ćemo se za potrebe ovog rada osvrnuti na:

- Sigmoidalnu aktivacijsku funkciju
- ReLU aktivacijsku funkciju

2.1.1. SIGMOIDALNA AKTIVACIJSKA FUNKCIJA

Linearne aktivacijske funkcije najjednostavnija su klasa aktivacijskih funkcija. Neuronske mreže sačinjene samo od neurona koji imaju linearnu aktivacijsku funkciju lagano se treniraju, no ne mogu naučiti složenija preslikavanja odnosno ona koja odstupaju od linearnog preslikavanja. Stoga se često preferiraju nelinearne aktivacijske funkcije [1]. Primjeri takvih aktivacijskih funkcija su sigmoida (logistička funkcija) i tangens hiperbolni. Sigmoidalna funkcija[1] (nazvana logističkom funkcijom zbog značaja u logističkoj regresiji) dugo je vremena omiljena aktivacijska funkcija. Devedesetih godina je bila čak *defaultna* opcija kod mnogih neuronskih mreža [7]. Funkcija je dobila ime po karakterističnom S obliku koji preslikava ulaz na vrijednosti u rasponu intervala [0 – 1]. Slika 2.2 prikazuje primjer sigmoidalne funkcije (logističku funkciju)



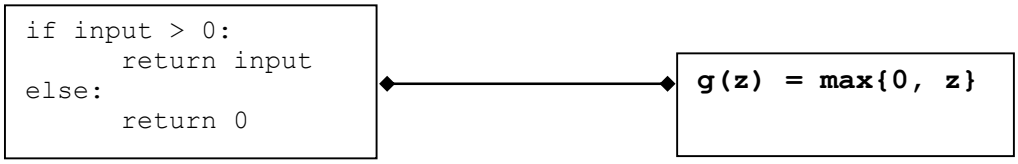
Slika 2.2: Prikaz logističke funkcije [11]

[1] Iako pojam sigmoidalna funkcija označava cijeli skup funkcija dalje u tekstu pod ovim pojmom podrazumijevam logističku funkciju.

Iako je pogodnija od linearnih aktivacijskih funkcija kod problema gdje je prisutno nelinearno preslikavanje, ove klase funkcija imaju određenih nedostataka. Prvi nedostatak je tzv. zasićenje. Ovo se očituje u tome da će se sve pozitivno velike vrijednosti preslikati u 1.0, a sve negativno velike vrijednosti u 0.0 za logističku funkciju ili u -1.0 za tangens hiperbolni. Drugo ograničenje jest što su ove funkcije osjetljive na promjene samo na relativno uskom intervalu oko sredine svoje domene. Pri tome je važno napomenuti da će se zasićenje i smanjena osjetljivost dogoditi neovisno o tome jesmo li neuronu dali korisne podatke ili ne. Jednom prenaučene neuronske mreže sa sigmoidalnim aktivacijskim funkcijama teško se prilagođavaju novim podatcima [7].

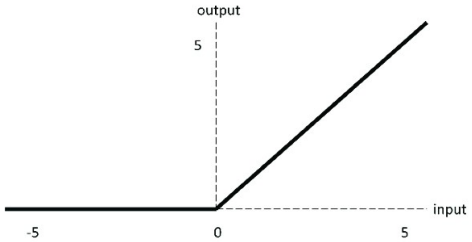
2.1.2. RELU AKTIVACIJSKA FUNKCIJA

Kako bismo koristili stohastički gradijentni spust i *backpropagation* algoritam htjeli bismo nelinearnu aktivacijsku funkciju koja bi nam omogućila učenja nelinearnog preslikavanja ali da istovremeno izgleda i ponaša se kao linearna. Rješenje ovog problema bila je ReL funkcija (kratica od *rectified linear function*). Ovo otkriće bilo je toliko značajno da nekoliko utjecajnih knjiga (npr. *Deep Learning*) ovo navode kao prekretnicu u razvoju dubokog učenja. Svaki neuron tj. jedinica koja implementira ReL zove se *rectified linear activation unit* odnosno skraćeno ReLU. ReL funkcija jednostavan je izračun gdje se za ulaz koji je veći od 0 vraća dani ulaz, a za ulaz manji od 0 vraća 0. Pseudo kod je prikazan na slici 2.3:



Slika 2.3: Prikaz ReL pseudokoda

Prednosti ReLU-a su prije svega linearno ponašanje, jednostavnost računanja (time i treniranja) i mogućnost jednostavnog reprezentiranja točne nule (npr. kod sigmoidalnih funkcija teže je ugoditi da se za input dobije točno 0) [7].



Slika 2.4: Prikaz ReL aktivacijske funkcije [10]

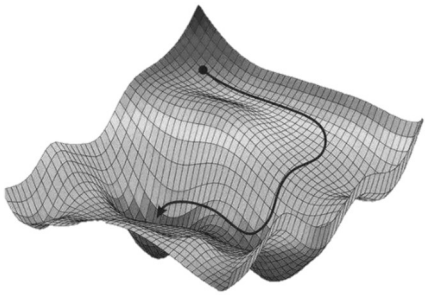
2.2. OPTIMIZATORI

Model učenja neuronske mreže podrazumijeva generaliziranje podataka pomoću algoritma te stvaranje predikcija [6]. U tu svrhu potreban nam je algoritam preslikavanja podataka s ulaza na izlaz iz naše neuronske mreže kao i optimizacijski algoritam. Svrha optimizacijskog algoritma bila bi ugađanje težina neuronske mreže na način da je dobiveni gubitak minimalan. Pri tome gubitak u najopćenitijem smislu definiramo kao odstupanje predikcije neuronske mreže od stvarne ciljne varijable za dani ulaz (pod uvjetom da nam je ona poznata). Smanjenjem gubitka povećavamo preciznost neuronske mreže.

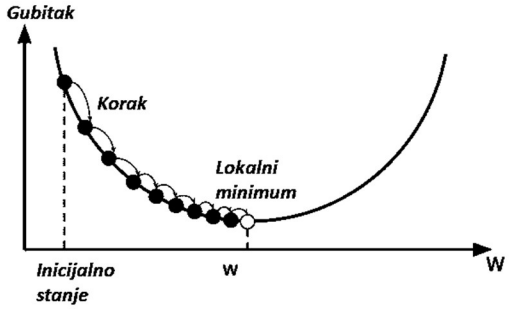
2.2.1. GRADIJENTNI SPUST

Gradijenti spust smatra se omiljenim razredom optimizatora. Ovaj optimizacijski algoritam koristi diferencijalni račun kako bi pronašao najveći gradijent u određenom trenutku. Gradijent (oznaka ∇) označava smjer najvećeg prirasta (strmine) neke veličine u prostoru. Pojednostavljeno rečeno algoritam gradijentnog spusta provodi sljedeće korake:

1. Počinje s nekim koeficijentima, odredi njihov gubitak i traži vrijednost gubitka u suprotnom smjeru gradijenta, koja je manja od sadašnje.
2. Pomiče se prema nižem gubitku i ažurira vrijednost koeficijenata.
3. Ponavlja korake 1 i 2 sve dok se ne postigne lokalni minimum.



Slika 2.5: Prikaz gradijentnog spusta po ravnini u prostoru [8]



Slika 2.6: Prikaz gradijentnog spusta po krivulji [9]

2.2.2. STOHASTIČKI GRADIJENTNI SPUST

Gradijentni spust kao algoritam optimizacije ima nedostatak velikog povećanja složenosti kod većih skupova podataka. Kako bi tome doskočili umjesto cijelog skupa podataka prilikom svake iteracije računanja uzimat ćemo manje podskupove koje ćemo odabirati na slučajan način. Zbog elementa nasumičnosti ovaj algoritam se zove *Stochastic Gradient Descent* (u tekstu dalje *SGD*). Obzirom kako ne radimo s cijelim skupom podataka često ćemo imati prisutne greške odnosno šum. Zbog toga SGD nerijetko treba više iteracija dok ne dosegne prikladni lokalni minimum. No unatoč povećanom broju iteracija računanje je više puta brže nego kod standardnog gradijentnog spusta [6].

2.2.3. SGD MOMENTUM

Kako bismo prevladali problem većeg šuma prilikom traženja gradijenta, uvodimo stohastički gradijentni silazak s algoritmom momenta. Ovaj algoritam pomaže u bržoj

konvergenciji funkcije gubitka. Stohastički gradijentni spust oscilira između smjerova gradijenta i u skladu s tim ažurira težine. Međutim, ukoliko dodamo trenutnoj iteraciji neke podatke iz prethodne iteracije možemo ubrzati konvergenciju. Zbog ovog „akceleriranog“ traženja gradijenta postoji opasnost da preskočimo lokalni minimum te prouzročimo još više oscilacija [6].

2.2.4. ADAGRAD (ADAPTIVE GRADIENT DESCENT)

Adaptivni gradijentni spust malo je drugačiji od ostalih algoritama baziranih na gradijentnom spustu. Dosadašnji algoritmi imali su parametar stope učenja – *learning rate* dalje u tekstu označen s η. Ovim hiperparametrom kontroliramo koliko će se naša neuronska mreža (naš model) promijeniti svaki puta kada se korigiraju težine. Mala vrijednost η tipično će uzrokovati duži proces učenja, dok velika vrijednost η povećava mogućnost prenaučenosti sub optimalnog skupa podataka. ADAGRAD algoritam prilikom svake iteracije mijenja i ugađa η te nema konstanto zadanu η. Računa ju pomoću formule:

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w(t-1)}$$

Gdje je $\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$,

α_t različita stopa učenja prilikom svake iteracije,

ϵ mali pozitivni broj kako bi spriječio dijeljenje s nulom.

w_t predstavlja težinu u iteraciji t

ADAGRAD-ovo automatsko korigiranje η reflektira činjenicu da u stvarnom skupu podataka imamo rijetka i gusta pojavljivanja određenih uzoraka. Sukladno tome htjeli bismo da model ne prenauči krive uzorke, već da učenje prilagodi zastupljenosti pojedinih uzoraka. ADAGRAD-ov nedostatak jest što za velike skupove podataka η može postati jako malen broj i time značajno usporiti ili čak zaustaviti proces učenja [6].

2.2.5. RMSPROP (ROOT MEAN SQUARE PROPAGATION)

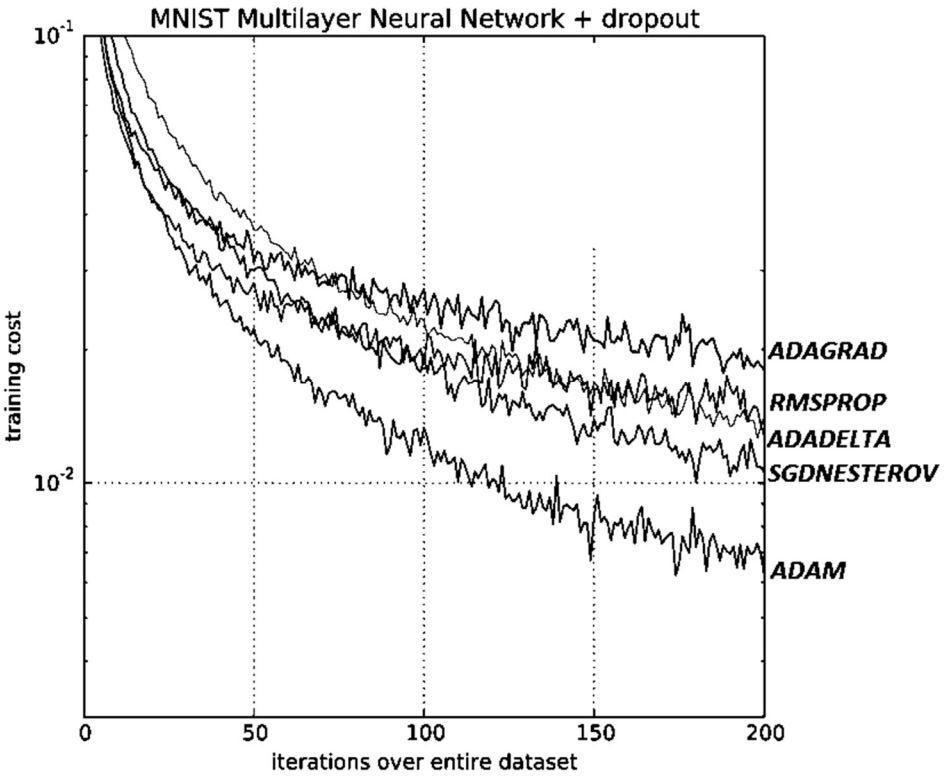
RmsProp smatramo poboljšanjem nad ADAGRAD algoritmom. RmsProp rješava problem monotono padajuće stope učenja η te se fokusira na ubrzanje optimizacijskog procesa kroz smanjenje broja iteracija da se dosegne lokalni minimum.

Algoritam uzima prosjek kvadrata gradijenata za svaku težinu i dijeli s korijenom kvadrata sredine. Jednostavnije rečeno, ako postoji parametar zbog kojeg funkcija gubitka jako oscilira, želimo penalizirati ažuriranje ovog parametra. Ovaj algoritam ima nekoliko prednosti u usporedbi s ranijim verzijama SGD-a. Algoritam brzo konvergira u minimum te zahtijeva manje ugađanje od ostalih varijanti algoritma gradijentnog spusta [6].

2.2.6. ADAM (ADAPTIVE MOMENT ESTIMATION)

ADAGRAD optimizator uvažavao je različitu učestalost pojavljivanja pojedinih uzoraka i time prilagođavao stopu učenja. RmsProp s druge strane je također ugađao stopu učenja no imao je problem što ona monotono pada. Adam optimizator danas je iznimno popularan jer kombinira prednosti ADAGRAD i RmsProp optimizatora. Umjesto da ugađa η prema prvom momentu (sredini), on ga ugađa prema vrijednosti drugog momenta (varijance). Točnije, algoritam izračunava prosjek pomaka gradijenta i kvadriranog gradijenta, a parametri beta1 i beta2 kontroliraju stope opadanja tih prosjeka. Tvorci Adam-a su istaknuli slijedeće prednosti ovog algoritma [5]:

1. Jednostavna implementacija
2. Računski učinkovit (prikaz usporedbe performansi na slici 3.3)
3. Mali memorijski zahtjevi
4. Prikladan za probleme koji su opsežni u količini podataka ili u parametrima
5. Prikladan za podatke u kojima su prisutni podaci sa šumom ili rijetki događaji
6. Hiperparametri su intuitivni te obično zahtijevaju malo podešavanja (ugađanja)



Slika 2.7: Prikaz performansi Adam-a u usporedbi sa ostalim optimizatorima [15]

2.3. KRITERIJ IZRAČUNA GUBITKA

Spomenimo još kriterij za izračun gubitka. Ova funkcija uzima rezultat predikcije koju je dala neuronska mreža te stvarnu vrijednost za dane inpute. Kriterij gubitka važan nam je za optimalno učenje neuronske mreže. Ovih je funkcija jako puno te ćemo se u ovom radu osvrnuti na dvije: L1Loss i MSE.

2.3.1. L1LOSS

Ova funkcija mjeri apsolutnu grešku prema formuli:

$$loss(x, y) = \sum |x - y|$$

Ovo je najjednostavniji kriterij izračuna gubitka te je prikladan za jednostavne modele. Konvencija je da pogreška ove vrste je prihvatljiva ako je ona manja ili jednaka 5% zbog toga što je u praksi jako nerealno očekivati da će predikcija našeg modela uvijek davati savršeno poklapanje sa stvarnim rezultatima [3].

2.3.2. MSELOSS

MSE kriteriji računa srednju kvadratnu grešku prema formuli:

$$loss = \frac{\sum(x - y)^2}{n}$$

Kriterij srednje kvadratne pogreške često je korišten kriterij kod regresijskih problema koji nisu previše dimenzionalni. Funkcija ima lijepo svojstvo da naglašava velike pogreške (kvadrat velikog broja je još veći broj) a istovremeno oprašta male greške (kvadrat malog broja je još manji broj). To je jako pogodno jer će rijetko naš model biti apsolutno precizan te ćemo favorizirati manja odstupanja naspram velikim [3].

3. OPIS PROBLEMA

Obzirom kako je problem optimalnog izbora arhitekture i parametara neuronske mreže za pojedini problem i dalje otvoreno pitanje, potrebno je sistematičnom promjenom parametara i praćenjem performansi eksperimentalno doći do optimalnog izbora. Pri tome važno je naglasiti da zaključak o optimalnom izboru vrijedi samo za ovaj specifični slučaj, bez ikakve garancije optimalnosti za ostale probleme.

Kao poligon za treniranje i eksploatiranje neuronske mreže napravljena je igra zmije u programskom jeziku Python uz pomoć biblioteka Pygame i Pytorch.

3.1. KRATKI OPIS IMPLEMENTACIJE IGRE

Projekt je arhitekturno inspiriran MVC stilom. Pri tome je organizacija sljedeća:

- *Pogled*: Prikaz ploče i svih elemenata na njoj, dojava o stanju ploče u nekom trenutku igre
- *Model*: Model neuronske mreže sa svim hiperparametrima i algoritmima potrebnim za učenje.
- *Nadglednik*: Pomoću stanja na ploči (koju mu dojavi *Pogled*) računa i poziva model te pomoću izlaza neuronske mreže odigra potez

Pogled uz metode koje su zadužene za *„rendering“* ploče i svih elemenata na njoj ima i API koji nudi informacije o poziciji hrane i mogućim kolizijama[2]. Pozicija hrane zadana je koordinatama na ploči dok je informacija o kolizijama dana kao boolova zastavica. (*True* znači da se dogodila kolizija te da je igra završena, *False* znači da kolizije nema te da se igra nastavlja).

Model pomoću biblioteke Pytorch definira razred koji modelira neuronsku mrežu te joj postavlja parametre (broj slojeva, aktivacijske funkcije, kriteriji gubitka, vrstu optimizatora…). U tom razredu implementiraju se i metode za učenje. Vrsta učenja je nadzirano učenje gdje se za ispravan potez uzima onaj koji bi generirao deterministički algoritam za isti skup ulaznih podataka.

Nadglednik „igra“ igru. Pojednostavljeno on izvodi sljedeći pseudo kod prikazan na slici 3.1:

```
ponovi:
      trenutno_stanje <- dohvati_stanje()
      determ_potez <- dohvati_determ_potez(trenutno_stanje)
      ai_potez <- dohvati_ai_potez(trenutno_stanje)
      kraj <- odigraj_potez(ai_potez)
      uči_iz_poteza(trenutno_stanje, determ_potez)
      ako je kraj:
            broj_igre++
            ako je rezultat > max_rezultat:
                  max_rezultat = rezultat
```

Slika 3.1: Prikaz pojednostavljenog pseudokoda *nadglednika*

3.2 DETERMINISTIČKI ALGORITAM

Zbog potrebe izračuna gubitka prilikom nadziranog učenja bilo je potrebno napraviti deterministički algoritam koji bi postizao solidne rezultate i bio „učitelj“ neuronskoj mreži. Ovaj algoritam zmiju navigira po najkraćem putu do hrane. Iako je pohlepan on pazi na

[2] Kolizijom se smatra trenutak “zabijanja” zmije u svoj rep ili u zidove koji omeđuju ravninu na kojoj se zmija kreće. Preciznije rečeno to je preklapanje koordinate zmije sa koordinatom prepreke

opasnosti te zmiju navigira samo u onom smjeru gdje nema neposredne opasnosti[3]. Algoritam postiže zadovoljavajuće rezultate obzirom da ne radi nikakvu vrstu pretraživanja. Prilikom odabira poteza algoritam koristi pojednostavljeni pseudo kod prikazan na slici 3.2:

```
Funkcija sljedeci_potez:
      koordinata_glave <- dohvati_koordinate_glave()
      koordinata_hrane <- dohvati_koordinate_hrane()
      koordinate_skretanja[] <- izracunaj_skretanja()
      udaljenosti[] <- izracunaj_udaljenosti(koordinate_skretanja)
      opasnosti[] <- izracunaj_opasnosti(koordinate_skretanja)
      sigurni_potezi[] <- izracunaj_sigurne_poteze(opasnosti)
      najkraci <- sigurni_potezi[0]
      za svaki potez iz sigurni_potezi:
            ako je udaljenosti[potez] < najkraci:
                  najkraci = potez

      vrati najkraci
```

Slika 3.2: Prikaz pojednostavljenog pseudokoda deterministčkog algoritma

Zbog prirode najkraćeg puta zmija kretanjem stvara karakterističan stepeničasti uzorak kretanja koji proizlazi iz nemogućnosti idealnog dijagonalnog kretanja.

3.3. METODIKA TESTIRANJA MREŽA I VREDNOVANJA REZULTATA

Kako bi se na ispravan i konzistentan način moglo uspoređivati različite varijante neuronskih mreža potreban je usklađeni kriterij za vrednovanje rezultata.

Praksom se pokazalo da u prvih 10 minuta učenja neuronska mreža nauči veliku većinu ponašanja deterministčkog algoritma. Daljnje produljenje vremena treniranja uglavnom ne donosi značajnije poboljšanje. Sukladno tome svakoj varijanti neuronske mreže dalo se točno 10 minuta treniranja. U tom periodu rezultat bi se ignorirao. Nakon isteka tog vremena

[3] Ako je glava zmije sa svih strana okružena opasnostima, algoritam će zmiju navigirati ravno. To je jedini teoretski slučaj kada se zmija može ubiti.

rezultat ulazi u vrednovanje te se prikuplja uzorak od 100 igara. Pri početku treniranja neuronske mreže može se uključiti opcija da prvih nekoliko igara igra deterministički algoritam. Tada neuronska mreža uči te se u kontekstu navigiranja njene odluke ignoriraju (no i dalje se uzimaju u obzir prilikom učenja za npr. računanje gubitka). Pokazalo se da sa neuronska mreža kada vidi nekoliko kompletno odigranih igara puno uspješnije nauči željeno ponašanje. Prilikom prikupljanja uzorka svi rezultati zapisuju se u CSV datoteku koja sadrži slijedeće podatke:

- Game Number: redni broj igre
- Score: ostvareni rezultat (broj pojedene hrane) u toj igri
- Max Score: najveći dosadašnji rezultat
- Deterministic Flag: zastavica koja je podignuta na 1 ukoliko tu igru je odigrao deterministički algoritam, 0 ako je igru odigrala neuronska mreža. Treba napomenuti da iz praktičnih razloga ako je ova zastavica postavljena na 1 Max Score neće se ažurirati.

3.4. ULAZNI I IZLAZNI SLOJEVI

Ulazni i izlazni slojevi neuronske mreže su konstantnog oblika za sve varijante neuronskih mreža koje se isprobavaju u ovom radu. Ulazni sloj jest jedanaest-dimenzionalni vektor koji se sastoji od boolovih vrijednosti:

- Opasnost ispred
- Opasnost zdesna
- Opasnost slijeva
- Smjer kretanja lijevo
- Smjer kretanja desno
- Smjer kretanja gore
- Smjer kretanja dolje
- Lokacija hrane po koordinati x manja od koordinate glave
- Lokacija hrane po koordinati x veća od koordinate glave
- Lokacija hrane po koordinati y manja od koordinate glave
- Lokacija hrane po koordinati y veća od koordinate glave

Izlaz modela sastoji se od 3 neurona koji daju trodimenzionalni boolov vektor poteza:

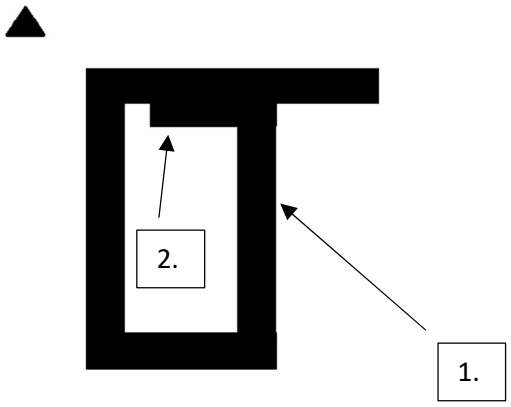
[1,0,0] → idi ravno (ne mijenjaj smjer)
[0,1,0] → idi desno
[0,0,1] → idi lijevo

Pri tome izlazni sloj neurona generira brojeve koji nisu ni centrirani ni normirani (dakle dolaze iz proizvoljnog intervala). Vektor poteza generira se na način da se na mjesto najvećeg broja u vektoru kojeg generira izlazni sloj upisuje broj 1, a na ostala mjesta 0. Na primjer izlaz iz modela oblika [3.2,15.2,1] rezultirati će potezom [0,1,0] to jest skretanjem udesno.

4. PRIKAZ REZULTATA TESTIRANJA NEURONSKIH MREŽA

Kako je već objašnjeno u prethodnom potpoglavlju ispitano je nekoliko varijanti neuronskih mreža. Kako bi se dobila još jasnija predodžba o rezultatima prikupljen je uzorak veličine 100 igara čistog deterministčkog agenta.

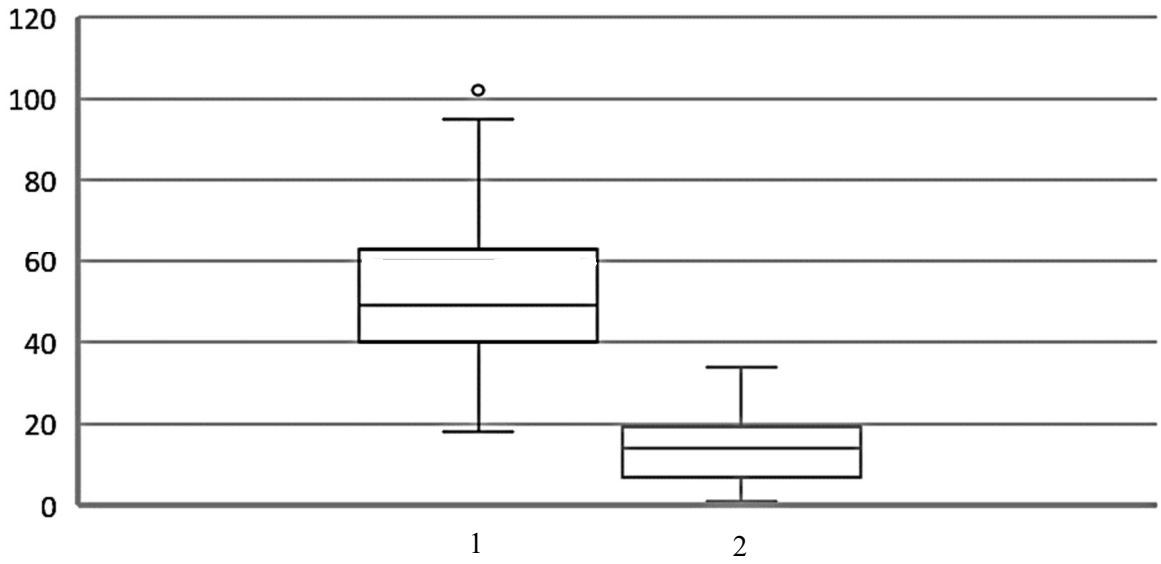
Deterministički algoritam u prosjeku ostvario je rezultat od 52.12 pojedene hrane s varijancom od 290.77 (standardna devijacija 17.052). Promatrajući igru deterministčkog agenta jedini slučaj kada zmija umire jest tzv. *boxing.* To je situacija kada se zmija zatvori (okruži) u samu sebe te nikako ne može izaći van. Time je algoritam suočen sa situacijom da će u jednom trenutku ostati bez sigurnih poteza gdje povlači potez [1,0,0] (igra ravno) i gubi igru. Nažalost ne postoji trivijalan način rješavanja ovog problema bez upotrebe neke vrste iscrpne pretrage. Primjer situacije *boxinga* dan je na slici 4.1.



Slika 3.1: Skica prikazuje situaciju boxinga. Tada deterministički algoritam gubi igru.

Neka trokut predstavlja hranu u danom trenutku te je oblik zmije kako je prikazano na slici 4.1. Zmija u trenutku 1. kreće se prema hrani no nailazi na svoj rep. Tada će deterministički algoritam birati između dva sigurna puta: skretanje lijevo ili skretanje desno. Odabrat će onaj koji ga dovodi bliže hrani (skretanje lijevo) (trenutak 2.). Time je uzrokovan *boxing* jer će se zmija zarobiti u sebi. Iako postoje metode kako se ovi slučajevi mogu smanjiti (npr. da se prilikom skretanja provjeri udaljenost do najbliže prepreke) niti jedna metoda ne otklanja ovaj problem kompletno. Ovaj je problem ključan jer neuronska mreža uči ponašanje od determinističkog algoritma te će također „naslijediti“ problem *boxinga*.

Prva neuronska mreža čije ćemo rezultate usporediti sastoji se od 3 sloja. Prvi (ulazni) sloj je veličine 11 neurona. Slijedi prvi skriveni sloj veličine 256 neurona te drugi skriveni sloj veličine 40 neurona. Na kraju je izlazni sloj veličine 3 neurona. Za sve neurone koristi se ReLU aktivacijska funkcija (zglobnica) kao optimizator koristi se ADAM sa kriterijem L1Loss odnosno apsolutna greška. Uzorak je skupljen kako je objašnjeno ranije u radu te su dobiveni sljedeći rezultati: Prosjek bodova koje je neuronska mreža ostvarila je 14.23 s varijancom od 76.69 (standardna devijacija je 8.76). Neuronska mreža ostvarila je najveći rezultat od 34 pojedene hrane. Usporedbu ovog modela neuronske mreže i determinističkog algoritma zorno možemo usporediti pomoću slijedećeg box plota (sl. 4.2).



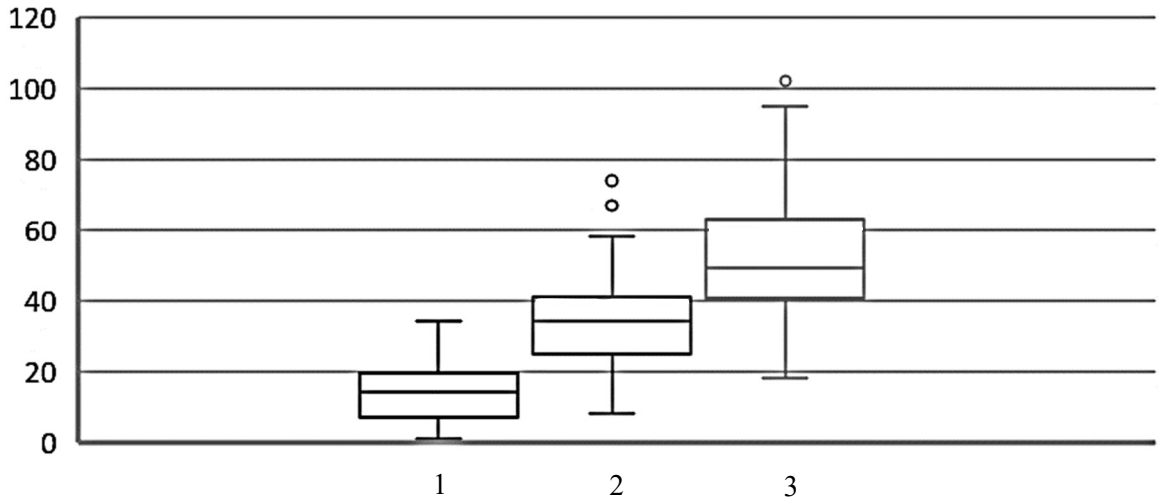
Slika 4.2: Prikaz box plot dijagrama rezultata:
1: deterministički algoritam
2: neuronska mreža

Kao što je vidljivo na dijagramu neuronska mreža konzistentno postiže lošije rezultate od determinističkog algoritma. Treći kvartil jedva doseže najlošije rezultate determinističkog algoritma. Proučavajući igru koju igra neuronska mreža često se događaju situacije da se zmija zabije sama u sebe ili u rubne zidove. Zmija se jako dobro navigira prema hrani te poput determinističkog algoritma stvara karakterističan stepeničasti uzorak kretanja ali slabo nauči ponašanje da razmatra samo one poteze koji su sigurni. Pojavila se teorija da ako moć odlučivanja neuronske mreže ograničimo samo na skup sigurnih[4] poteza trebali bismo dobiti rezultate koji su jednako dobri onima koje postiže deterministički algoritam.

Slijedeći tu ideju modificiran je kôd *controllera* na način da neuronska mreža izabire samo iz skupa sigurnih poteza. Dobiveni rezultati prikazani su u nastavku:

Dobivena je sredina od 34.61 pojedene hrane s varijancom od 179.1 (standardna devijacija 13.38). Usporedimo li dobiveni rezultat sa rezultatom determinističkog algoritma dobivamo slijedeći box plot:

[4] Pod pojmom sigurni potezi mislim na one poteze koji neće neposredno dovesti do zabijanja zmije to jest do završetka igre

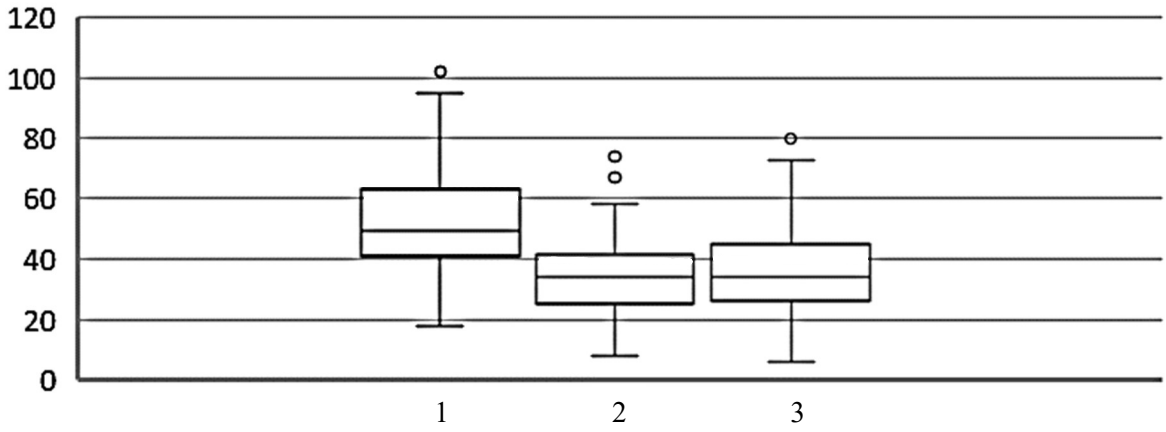


Slika 4.3: Prikaz box plot dijagrama rezultata:
1: neuronska mreža
2: potpomognuta neuronska mreža
3: deterministički algoritam

Iako sam uspio postići značajno poboljšanje u odnosu na „golu“ neuronsku mrežu, i dalje ne možemo reći da su dobiveni rezultati jednako dobri kao oni koje postiže deterministički algoritam. Promatrajući igru ove neuronske mreže (na koju ću se dalje u tekstu referencirati pod pojmom potpomognuta neuronska mreža) može se primijetiti da se igra ne gubi na trivijalne načine već isključivo *boxingom* stoga ovakva razlika u rezultatima pomalo iznenađuje.

Iduća varijanta neuronske mreže čije ćemo rezultate ovdje iznijeti jest ponovno potpomognuta neuronska mreža međutim kao optimizator umjesto Adam-a koristi se RmsProp. Aktivacijska funkcija i dalje ostaje ReLU kao i funkcija za izračun gubitka (L1Loss ili apsolutna greška). Mreži je također dan jednak period treniranja (10 minuta) te je dobiven slijedeći rezultat:

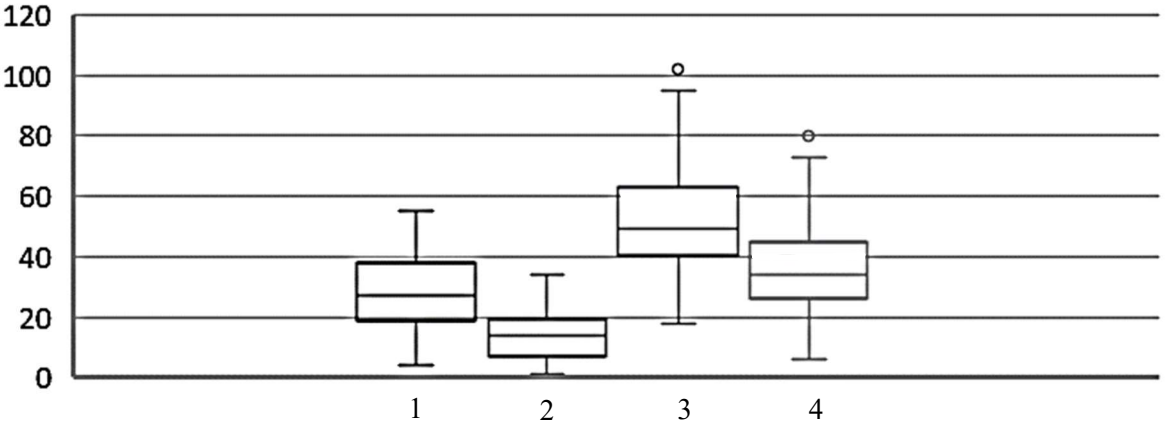
Ova neuronska mreža postiže sredinu od 36.01 pojedene hrane s varijancom od 225.89 (standardna devijacija iznosi 15.030). Box plot dijagram prikazan je na slici 4.4



Slika 4.4: Prikaz box plot dijagrama rezultata:
1: deterministički algoritam
2: Potpomognuta neuronska mreža, Adam optimizator
3: Potpomognuta neuronska mreža, RmsProp optimizator

Iako je dobiveni rezultat mrvicu bolji od ekvivalentne neuronske mreže sa Adam optimizatorom razlika se ne može smatrati statistički značajnom.

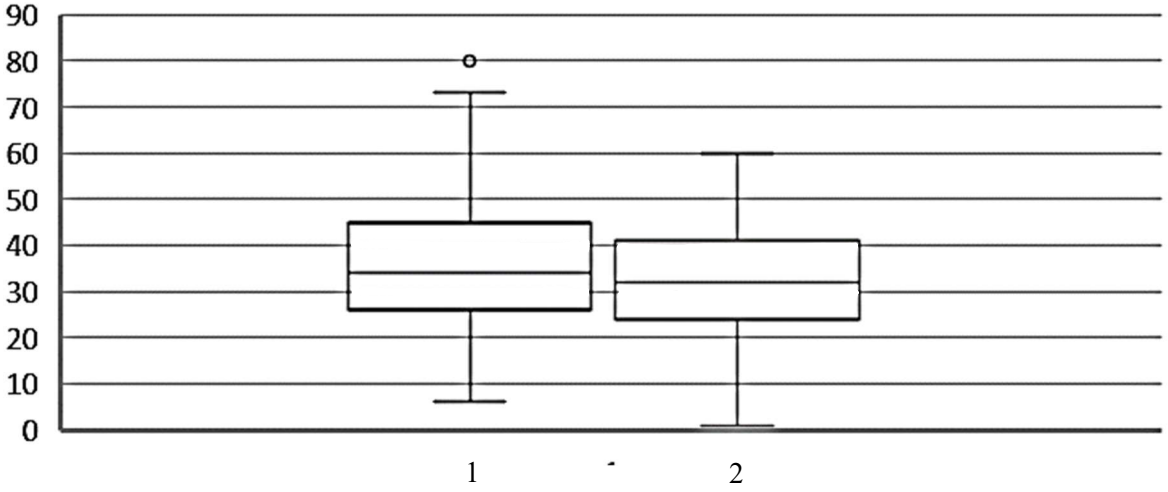
Sluteći kako je RmsProp optimizator pogodan uklonjena mu je restrikcija gdje neuronska mreža može povući samo siguran potez te mu je promijenjen kriterij izračuna greške sa L1Loss (apsolutna pogreška) na MSE (srednja kvadratna pogreška). Također je u drugom skrivenom sloju postavljena sigmoidalna prijenosna funkcija. Drugi skriveni sloj je povećan za 50 neurona te arhitektura izgleda 11x256x90x3. Uz ovakvu arhitekturu ostvaren je prosjek od 28.9 pojedene hrane sa varijancom od 153.13 (standardna devijacija iznosi 12.37). Vizualna usporedba sa ostalim rezultatima vidljiva je na slici 4.5.



Slika 4.5: Box plot dijagram rezultata neuronskih mreža:
1: RmsProp optimizator, arhitektura 11x256x90x3, MSE kriterij gubitka
2: Adam optimizator, arhitektura 11x256x40x3, L1 kriterij gubitka
3: Deterministički algoritam (nema neuronske mreže)
4: Potpomognuta neuronska mreža, RmsProp optimizator, arhitektura 11x256x40x3, L1 kriterij gubitka

Uklanjanjem drugog skrivenog sloja te postavljanjem arhitekture mreže na 11x256x3 s ReLU aktivacijskom funkcijom te RmsProp optimizatorom (MSE evaluacija greške) dobivaju se zanimljivi rezultati:

Postignuta je sredina od 32.32 sa varijancom od 173.6 (standardna devijacija iznosi 13.17). Taj rezultat komparabilan je sa rezultatom potpomognute neuronske mreže. Odnos ta dva rezultata jasno je vidljiv na slici 4.6.



Slika 4.6: Box plot prikaz rezultata:
1: Potpomognuta neuronska mreža, RmsProp optimizator, arhitektura 11x256x40x3 ,L1Loss kriterij gubitka
2: Obična neuronska mreža, arhitektura 11x256x3, RmsProp optimizator, MSE kriterij gubitka

Vizualno gledajući dijagram čini nam se kako su rezultati čak na razini potpomognute neuronske mreže. Ako uspijemo statističkim testiranjem dokazati značajnu razliku u sredini između rezultata ove neuronske mreže i neuronske mreže s početka (11x256x40x3, ReLU, Adam, L1Loss) možemo reći da smo pronašli značajno poboljšanje u odnosu na inicijalnu postavku.

Koristit ćemo jednostrani T test za testiranje srednje vrijednosti uz poznate varijance sa razinom značajnosti $\alpha = 0.01, v = 198\ df$. Test statistika je:

$$t = \frac{\bar{x}_1 - \bar{x}_2 - d_0}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Postavimo hipoteze:

H_0: *Sredina rezultata neuronske mreže arhitekture* $11x256x3$ *jednaka je sredini rezultata neuronske mreže arhitekture* $11x256x40x3$

H_1: *Sredina rezultata neuronske mreže arhitekture* $11x256x3$ *veća je od sredine neuronske mreže arhitekture* $11x256x40x3$

Izračun:

$$\bar{x}_1 = 14.24, \qquad s_1 = 8.76, \qquad n_1 = 100$$

$$\bar{x}_2 = 32.31, \qquad s_2 = 13.17, \qquad n_2 = 100$$

$$s_p = \sqrt{\frac{99 * 76.7 + 99 * 173.60}{100 + 100 - 2}} = 11.18705$$

$$t = \frac{32.31 - 14.24}{11.18705 * \sqrt{2 * \frac{1}{100}}} = 11.4215$$

Kritična vrijednost t distribucije za $\alpha = 0.01\ , df = 198$ je 2.576. Budući da dobiveni rezultat ulazi u kritično područje možemo odbaciti nultu hipotezu u korist alternativne na razini značajnosti 1%.

Preostaje provjeriti postoji li statistički značajna razlika rezultata ove neuronske mreže i rezultata potpomognute neuronske mreže sa RmsProp optimizatorom arhitekture 11x256x40x3 (L1loss izračun gubitka) obzirom da nam se čini kako bi ovi rezultati mogli biti jednaki.

Ako uspijemo potvrditi jednakost sredina ovo malo „potonuće“ desnog dijagrama mogli bismo objasniti lošijim uzorkom.

Koristit ćemo dvostrani T test za testiranje srednje vrijednosti uz poznate varijance sa razinom značajnosti $\alpha = 0.05, v = 198\ df$. Test statistika je:

$$t = \frac{\bar{x}_1 - \bar{x}_2 - d_0}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Postavimo hipoteze:

H_0: *Sredina rezultata neuronske mreže arhitekture* $11x256x3$ *jednaka je sredini neuronske mreže arhitekture* $11x256x40x3$

H_1: *Sredina rezultata neuronske mreže arhitekture* $11x256x3$ *različita je od sredine rezultata potpomognute neuronske mreže arhitekture* $11x256x40x3$

Izračun:

$$\bar{x}_1 = 36.01, \qquad s_1 = 15.03, \qquad n_1 = 100$$

$$\bar{x}_2 = 32.31, \qquad s_2 = 13.17, \qquad n_2 = 100$$

$$s_p = \sqrt{\frac{99 * 225.89 + 99 * 173.60}{100 + 100 - 2}} = 14.13312$$

$$t = \frac{32.31 - 36}{14.13312 * \sqrt{2 * \frac{1}{100}}} = -1.84618$$

Kritična vrijednost t distribucije za $\alpha = 0.025\,, df = 198$ je 1.972. Budući da dobiveni rezultat ne ulazi u kritično područje ne možemo odbaciti nultu hipotezu u korist alternativne na razini značajnosti 5%. Iz toga možemo tvrditi da razlika u sredinama između ovih mreža nije statistički značajna.

Iako nismo uspjeli dostići rezultate koje ostvaruje deterministički algoritam, rezultat ovog i prethodnog testa govori nam da smo pažljivim odabirom parametara ipak napravili određeni napredak. Uspješno smo se približili rezultatima koje ostvaruje neuronska mreža potpomognuta determinističkim algoritmom. Taj je rezultat ujedno i najbliži rezultatima „čistog“ determinističkog algoritma.

5. ZAKLJUČAK

Prilikom modeliranja neuronske mreže nužno je razmišljati o optimalnosti parametara. Nažalost, ne postoji jednostavna metoda pronalaska apsolutno optimalnog skupa parametara te često moramo posegnuti za metodom isprobavanja. Cilj ovog rada je prikazati kako pomoću razumijevanja problema možemo suziti izbor mogućih parametara, te potom ih isprobati i vrednovati njihov rezultat.

Pokazalo se da za problem razmatran u ovom radu RmsProp optimizator postiže nešto bolje rezultate od Adam optimizatora. Najveću razliku u brzini učenja (a time i razliku i u postignutim rezultatima) čini odabir broja slojeva i neurona mreže. Povećanje broja slojeva neuronske mreže iznad jednog skrivenog sloja značajno pogoršava performanse mreže. U slučaju mreža sa dva skrivena sloja, mreže koje su imale veći broj neurona u drugom skrivenom sloju postizale su konzistentno lošije rezultate. Zbog nedovoljnog broja napravljenih testiranja ne možemo tvrditi, no možemo naslućivati, kako MSE kriterij za izračun pogreške daje bolje rezultate u odnosu na kriterij apsolutne pogreške. ReLU aktivacijska funkcija također postiže značajno bolje rezultate u odnosu na sigmoidalnu funkciju. To možemo objasniti mogućom potrebom za dužim treniranjem mreža koje koriste sigmoidalne aktivacijske funkcije [7]. Pri razmatranju modela trebamo svakako uzeti u obzir i složenost. Uvijek ćemo htjeti dati prednost jednostavnijem modelu (u vidu broja slojeva i neurona) naspram složenijem modelu ako složeniji model ne daje *značajno* bolje rezultate u odnosu na jednostavniji model [16].

Rezultate potpomognute[5] neuronske mreže moguće je dostići odabirom optimalnih parametara međutim neuronske mreže bez obzira na parametre i vrijeme učenja ne postižu rezultate deterministčkog algoritma. Bolje učenje ponašanja deterministčkog algoritma mogli bismo dobiti restrukturiranjem ulaznih podataka tako da neuronskoj mreži damo više podataka o trenutnom stanju. Neuronskoj mreži mogli bismo davati udaljenosti umjesto boolovih vrijednosti ili koristiti konvolucijsku neuronsku mrežu sa ulaznim podatcima piksela (točaka) na ploči.

[5] Neuronska mreža kod koje je implementirano ograničenje odluke samo na skup sigurnih poteza

LITERATURA

[1] Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT press.

[2] Bašić, B.D., Čupić, M. and Šnajder, J., 2008. Umjetne neuronske mreže. *Zagreb: Fakultet elektrotehnike i računarstva*.

[3] Pratyaksha Jha, A Brief Overview of Loss Functions in Pytorch (2019, veljača). Poveznica: https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7 (pristupljeno 10. svibnja 2022.)

[4] Jason Brownlee, Understand the Impact of Learning Rate on Neural Network Performance (2019., siječanj). Poveznica: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (pristupljeno 11. Svibnja 2022.)

[5] Jason Brownlee, Gentle Introduction to the Adam Optimization Algorithm for Deep Learning (2017., srpanj). Poveznica: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (pristupljeno 8. Svibnja 2022.)

[6] Ayush Gupta, A Comprehensive Guide on Deep Learning Optimizers (2017., rujan). Poveznica: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/ (prisupljeno 13. Svibnja 2022.)

[7] Jason Brownlee, A Gentle Introduction to the Rectified Linear Unit (ReLU) (2019., siječanj). Poveznica: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (pristupljeno 13. Svibnja 2022.)

[8] https://easyai.tech/en/ai-definition/gradient-descent/ (pristupljeno 30. Svibnja 2022.)

[9] https://github-wiki-see.page/m/SoojungHong/MachineLearning/wiki/Gradient-Descent (pristupljeno 30. Svibnja 2022.)

[10] https://www.researchgate.net/figure/ReLU-activation-function_fig7_333411007 (pristupljeno 30. Svibnja 2022.)

[11] https://www.researchgate.net/figure/Logistic-sigmoid-function-Maps-real-numbers-to-the-interval-between-0-and-1_fig6_234049070 (pristupljeno 30. Svibnja 2022.)

[12] https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc (pristupljeno 30. Svibnja 2022.)

[13] https://www.geeklingo.net/nokia-brings-snake-back-messenger/ (pristupljeno 30. Svibnja 2022.)

[14] https://dribbble.com/tags/nokia_snake_game_design (pristupljeno 30. Svibnja 2022.)

[15] https://www.geeksforgeeks.org/intuition-of-adam-optimizer/ (pristupljeno 30. Svibnja 2022.)

[16] Hrvatska enciklopedija – Occamova britva https://www.enciklopedija.hr/natuknica.aspx?ID=70091 (pristupljeno 30.Svibnja 2022.)

SAŽETAK

Igra „Zmija“ jednostavna je igra pogodna za implementaciju agenta koji bi putem umjetne neuronske mreže igrao igru. Obzirom kako ne postoji općenita formula za izbor optimalnih parametara umjetnih neuronskih mreža potrebno je metodom isprobavanja pronaći optimalne parametre. U radu predstavljena je metodika isprobavanja i vrednovanja različitih varijanti neuronskih mreža. Sve varijante neuronskih mreža su testirane te su rezultati prikazani box plot dijagramima. Na kraju je dana kratka diskusija i moguća objašnjenja rezultata.

Ključne riječi: Igra zmija, parametri neuronske mreže, optimizacija parametara

ABSTRACT

The game "Snake" is a simple game suitable for the implementation of an agent who would play the game with help form artificial neural network. Since there is no general formula for selecting the optimal parameters of artificial neural networks, it is necessary to them by testing. This paper presents a methodology for testing and evaluating different variants of neural networks. All neural network variants were tested and the results are shown in box plot diagrams. At the end, a short discussion and possible explanations of the results are given.

Keywords: Snake game, neural network parameters, parameter optimization