STROJNO UČENJE V PYTHONU

Področje: Računalništvo in informatika

RAZISKOVALNA NALOGA

Martin Alojz Flisar,

Filip Hočevar Draškovič

8. razred

Mentor: Aleš Drinovec

2020/2021

Osnovna šola narodnega heroja Maksa Pečarja

KAZALO

KAZALO SLIK

KAZALO TABEL

1 POVZETEK

V zadnjem času je svet doživel ogromen napredek na področju strojnega učenja, malokdo pa dejansko ve, kako le-to deluje. Večina ljudi meša strojno učenje z umetno inteligenco, pa čeprav sta to dve dokaj različni področji. To raziskovalno nalogo sva ustvarila z namenom samostojnega učenja na tem področju, ki bo, kot vse kaže, v naslednjih letih vedno bolj priljubljeno.

V raziskovalni sva se ukvarjala s strojnim učenjem v programskem jeziku *Python*. Implementirala sva algoritem spodbujevalne nevronske mreže v igro "Kača" in preverila, kako natančno se lahko le-ta nauči igrati igro, oziroma koliko točk lahko nevronska mreža zbere z različnimi parametri in velikostmi mreže.

Ta raziskovalna naloga je razdeljena na dva dela. To sta teoretični del, kjer je navedena teorija strojnega učenja, in pa eksperimentalni oziroma praktični del, kjer sva poskušala doseči čim višje rezultate nevronske mreže pri igri Kače.
Model se je že po nekaj poskusih naučil zelo dobro igrati igro Kača. Nato sva poskusila uporabiti ta algoritem z različnimi parametri, ki si jih lahko ogledate v eksperimentalnem delu. Najina hipoteza je bila potrjena.

Pri strojnem učenju je vedno prisoten nek računalnik, ki algoritme izvaja. Glavna omejitev (ki trenutno tudi preprečuje stvaritev splošne umetne inteligence) je seveda računska zmožnost računalnika. Čeprav so nekatere zahtevnejše naloge (npr. DeepMind-ov AlphaZero) izvedene na zmogljivejših, včasih celo kvantnih računalnikih. Ta omejitev je bila prisotna tudi pri tej nalogi, vendar ni vplivala na poskuse in rezultate.

2 ZAHVALA

Zahvaljujeva se mentorju za pomoč in nasvete pri pisanju naloge in prof. Andreju Bauerju za preverjanje pravilnosti vsebine in splošno pomoč.

3 UVOD

Za strojno učenje sva se začela zanimati šele pred pol leta, ko sva na spletnem portalu YouTube našla video o igri Super Mario World. Ampak igre v tem primeru ni igral človek, temveč natančno streniran računalniški algoritem. Bilo je neverjetno, saj bi bilo s preprostim programiranjem zelo težko povedati programu, kdaj naj skoči in kdaj naj se premika levo ali desno. V igro je vštetih preveč spremenljivk, kot so oblika okolice, nasprotniki, nagrade, itd. Zato pa, kot sva kasneje ugotovila, obstaja veliko lažji in učinkovitejši način za rešitev takšnih problemov: strojno učenje. Eno glavnih in najbolj zanimivih področij strojnega učenja so nevronske mreže. Te oponašajo delovanje človeških možganov, zato so pri takšnih nalogah veliko uspešnejše. Algoritem namreč začne brez predhodnega znanja, podane pa ima informacije o poteku igre, možnost uporabe ukazov in cilj – po navadi je to čim večje število točk. Z vsako generacijo se izpopolnjuje, saj je ob opravljeni nalogi nagrajen, oziroma kaznovan, če naloge ne opravi. Čas učenja je odvisen od kompleksnosti nevronske mreže in zmogljivosti računalnika, nalogo pa lahko, ko je mreža naučena, opravi vsaj tako dobro ali pogosto celo boljše kot človek.

Slika 1. Nevronska mreža igra igro Super Mario World (SethBling, 2015)

Ta princip naju je zelo navdušil, čeprav je matematično kar zapleten, a na voljo je ogromno programskih knjižnic, ki ponujajo že izdelano nevronsko mrežo.
Splošni namen te raziskovalne naloge je ugotoviti, ali so algoritmi strojnega učenja dovolj dobri, da se naučijo igro Kača in jo premagati.
Najina hipoteza je:

Domnevamo, da se algoritmi strojnega učenja lahko naučijo igrati računalniško igro Kača brez začetnega znanja, tako da so po treningu zmožni zbirati točke (jesti hrano).

V tej raziskovalni nalogi bova:

- Implementirala algoritem nevronske mreže v igro Kače, da bi preverila, kako natančno se jo lahko nevronska mreža nauči igrati.

4 TEORETIČNI DEL

4. 1 STROJNO UČENJE – UVOD

Psi in mačke imajo veliko skupnih lastnosti. Oboji so kosmate živali podobne barve s štirimi nogami ter repom in ušesi. Kako lahko ljudje tako preprosto ločimo med njimi?
Ko smo še majhni, nam povejo, katere živali so psi in katere mačke. Kmalu ne potrebujemo več novih primerov, saj je naša zmožnost učenja dovolj dobra, da prepoznamo žival kot mačko oziroma psa, tudi če te živali še nismo videli. Dolgo je bila to najtežja naloga za računalnike. Lahko so si zapomnili ogromno podatkov, veliko več kot si jih je zmogel človek. Lahko so zmnožili dve velikanski števili v trenutku, ko bi si mi šele ustvarjali sliko teh števil v glavi. Ampak niso se bili zmožni učiti. To pa se je z odkritjem in napredkom strojnega učenja spremenilo.

4. 2 ZAKAJ POTREBUJEMO STROJNO UČENJE

Strojno učenje je potrebno, da računalniki sofisticirano opravljajo nalogo brez kakršnega koli posredovanja ljudi, na podlagi učenja in nenehnega pridobivanja izkušenj, da bi razumeli kompleksnost problema in potrebo po prilagodljivosti.
To zveni zelo zakomplicirano, zato bomo dodali primer in dodatno razlago. Naloge lahko delimo na tiste, ki jih opravljajo ljudje in tiste, ki jih opravljajo računalniki s pomočjo strojnega učenja in so za ljudi preveč zapleteni.

- Naloge, ki jih izvajajo človeška bitja: Ogromno je nalog, ki jih opravijo ljudje vsak dan, vendar je glavna skrb pri tem popolno izvajanje nalog in izvajanje po natančno določenem programu ali algoritmu. Primeri: kuhanje, vožnja, prepoznavanje govora...

- Naloge, ki presegajo človeške sposobnosti: Druga kategorija nalog vključuje naloge, ki jih lahko na učinkovit način opravi algoritem strojnega učenja. To je največkrat analiza velikih in zapletenih podatkovnih nizov, kot je vremenska napoved, e-trgovina, spletno iskanje, priporočanje oglasov itd. Zaradi velikih količin podatkov, ki jih človek ne more obdelati, je to zanj zapleteno in se mora zanesti na strojno učenje.

Vsak problem v strojnem učenju lahko postavimo v eno od spodaj naštetih petih kategorij. Odvisno od vrste problema je mogoče uporabiti ustrezen pristop strojnega učenja. (Alzubi, J., Nayyar, A., Kumar, A., 2018):

- Regresijski problem (1)
- Klasifikacijski problem (2)
- Težava z odkrivanjem nepravilnosti (3)
- Združevanje v skupine ali grupiranje (4)
- Spodbujevalni problem (5)

4. 3 STROJNO UČENJE

Učenje kot generični proces je pridobivanje novega ali spreminjanje obstoječega vedenja, vrednot, znanja, spretnosti, … To opredeljuje teorijo osebnega učenja, torej, kako se učijo ljudje. Stroji se zanašajo na podatke, kar je v nasprotju s tem, kar je naravno za ljudi: učenje iz izkušenj. Na osnovnem nivoju spada strojno učenje (angl. ML oz. Machine Learning) v kategorijo umetne inteligence, ki računalnikom omogoča razmišljanje in zmožnost samostojnega učenja. Gre predvsem za to, da računalniki spreminjajo svoja dejanja, da bi jih lahko izboljšali in posledično dosegli večjo natančnost. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

Obstaja ogromno definicij strojnega učenja, ampak uradno definicijo je skoval Arthur Samuel leta 1959, ko je strojno učenje opredelil kot »področje znanosti, ki omogoča računalnikom, da se učijo, ne da bi bili za to izrecno programirani.«

Pred kratkim je Tom Mitchell podal drugo definicijo, ki pa je tudi bolj natančna: »računalniški program, ki se uči iz izkušenj E, izvaja opravilo T in ima merilo uspešnosti P, kjer se njegova uspešnost pri opravilu T, merjeno s P, izboljšuje z izkušnjami E.« (Alzubi, J., Nayyar, A., Kumar, A., 2018).

Strojno učenje lahko delimo na **nadzorovano** (angl. supervised) in **nenadzorovano** (angl. unsupervised) učenje.

Algoritem nadzorovanega učenja oblikuje funkcijo, ki poveže vhode z izhodi. Za to uporablja znane primere. Dober algoritem nadzorovanega učenja predvidi vrednost izhodov z znanjem vrednosti izhodov že videnih vhodov. Zmožnost opravljanja uspešnih predvidevanj vrednosti izhodov glede na še ne videne vhode, se imenuje **generalizacija**. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

Nadzorovano učenje še naprej delimo na regresijo in klasifikacijo. Pri **regresiji (1)** so izhodne spremenljivke zvezne, torej želimo poiskati takšno funkcijo, ki vhodnim spremenljivkam pripiše izhodne. Primer tega je funkcija, ki povezuje starost z višino. **Klasifikacija (2)** se ukvarja s problemi, kjer so izhodne spremenljivke diskretne, pri njej želimo vhodne podatke razvrstiti v diskretne kategorije brez številskih razmerij. Primer tega je že prej omenjeno razvrščanje slik psov in mačk. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

Problem zaznavanja nepravilnosti (3) - Sem spadajo problemi, pri katerih je potrebno, analizirati določen vzorec in zaznati spremembe oziroma nepravilnosti v vzorcu. Na primer, podjetja s kreditnimi karticami uporabljajo algoritme za odkrivanje nepravilnosti ali anomalij, da bi ugotovili odstopanje od običajnega vedenja transakcij njihovih klientov. Kadar pride do nenavadne transakcije, dobijo opozorilo. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

Pri nenadzorovanem učenju poskuša algoritem ugotoviti povezave med vhodi, ne da bi se pred tem učil iz podatkov. Sem spadajo problemi **združevanja v skupine oziroma grupiranja (4).** Algoritmi, ki rešujejo takšne probleme, se poskušajo naučiti struktur v podatkih in poskušajo narediti skupine na podlagi podobnosti v strukturi podatkov. Različni razredi ali skupine so nato označeni. Primera tega sta lahko predvidevanje možnih prijateljskih zvez iz profilov na socialnih omrežjih, ali združevanje člankov iz podobnih področij v skupine. Slab primer bi bil ugotavljanje posameznikovega spola iz njegovega imena. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

Nevronske mreže (ANN, angl. artificial neural networks) so dovolj fleksibilne, da jih lahko uporabljamo za reševanje večine problemov.
V tej raziskovalni nalogi se bova ukvarjala predvsem z algoritmom nevronske mreže in spodbujevalnim učenjem, ki ga bova razložila in opredelila spodaj.

4. 4 SPODBUJEVALNO UČENJE (5) – UVOD

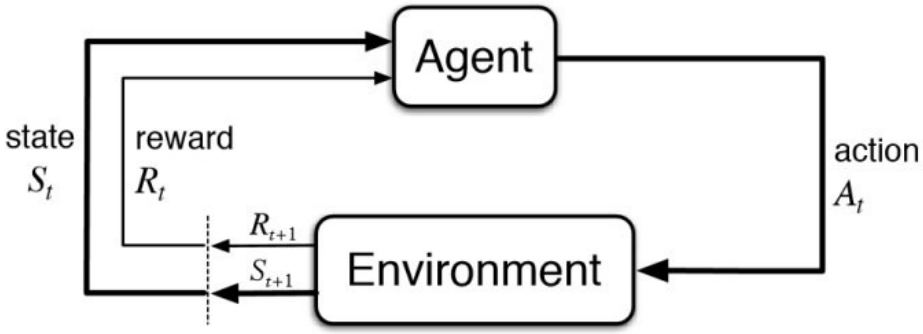
Spodbujevalno učenje (angl. reinforcement learning) spada pod nenadzorovano učenje. Postavimo ga poleg evolucijskega učenja, ki se prilagodi razmeram in je osnovano na bioloških organizmih in tako imenovanega skupinskega učenja, kjer je prisotnih več osebkov. Algoritem dobi po izvedenem dejanju odziv, ki mu pove, ali je izhod pravilen ali ne. Algoritem mora raziskati in izključiti različne možnosti, da dobi pravi izhod. Tako se izboljšuje z vsako generacijo. Lahko je prisotnih tudi več modelov, saj to pospeši proces učenja. Zato je nekakšna mešanica evolucijskega in skupinskega učenja. Algoritem se uči s poskušanjem in izpopolnjevanjem. Tako oponaša žive organizme, prav to pa ga naredi mističnega in zanimivega. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

4. 4. 1 SPODBUJEVALNO UČENJE

Ko se dojenček še igra, maha z rokami ali se ozira, pri tem nima učitelja, vendar pa ima neposredno povezavo s svojim okoljem. Raziskovanje okolja mu da veliko informacij o vzrokih in učinkih, o posledicah dejanj in o tem, kaj storiti, da dosežemo cilje. Skozi življenje prinašajo takšne interakcije znanje o našem okolju in nas samih. Ali se učimo voziti avto ali se pogovarjamo, v obeh primerih se zavedamo, kako to vpliva na naše okolje. Učenje s pomočjo interakcije je temeljna ideja teorije spodbujevalnega učenja. (Sutton, Richard S., Barto, Andrew G. 2018).

Pristop, ki ga bomo raziskovali, se imenuje spodbujevalno učenje in je veliko bolj osredotočeno na učenje iz interakcije za doseg ciljev, kot drugi pristopi k strojnemu učenju. Spodbujevalno učenje (RL, reinforcement learning) je vse večja podmnožica strojnega učenja, ki vključuje programske agente, ki poskušajo ukrepati v upanju, da bodo povečali dobljeno nagrado. Obstaja več različnih oblik povratnih informacij. V primerjavi z algoritmi nadzorovanega učenja, ki preslikajo funkcije od vhoda do izhoda, algoritmi spodbujevalnega učenja običajno ne vključujejo ciljnih izhodov (podani so le vhodi). Obstajajo 3 elementi osnovnega algoritma spodbujevalnega učenja: **agent** (ki se lahko odloči za **dejanja** v svojem trenutnem **stanju**), **okolje** (odziva se na dejanja in posreduje nov vhod **agentu**) in **nagrada** (spodbudni ali kumulativni mehanizem, ki ga vrne okolje).

Osnovna shema za algoritem spodbujevalnega učenja je podana spodaj (Sharma, Siddharth, 2020):



Slika 2. Algoritem spodbujevalnega učenja. (Sharma, Siddharth, 2020)

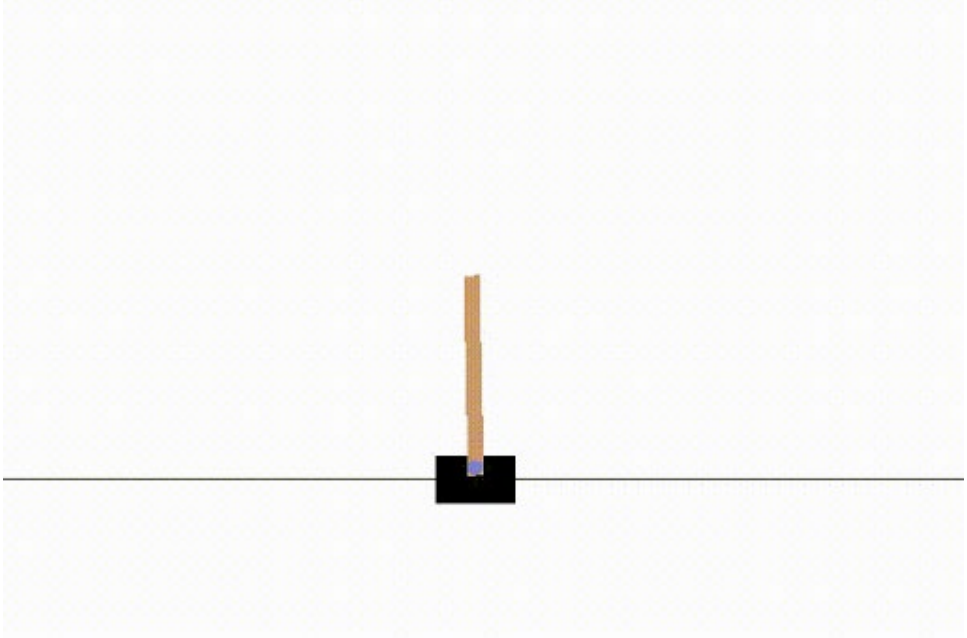
Splošni cilj večine algoritmov spodbujevalnega učenja je doseči ravnovesje med raziskovanjem (usposabljanje na novih podatkih) in izkoriščanjem (uporaba predhodno zajetih podatkov). Neposredni cilj je maksimirati nagrado z preizkusi, ki se izmenjujejo med zgoraj omenjenim izkoriščanjem in raziskovanjem. Pomembno je omeniti, da obstajajo tri vrste izvedb spodbujevalnega učenja: na podlagi politike, vrednosti in modela. Spodbujevalno učenje, ki temelji na politiki, vključuje pripravo politike ali deterministične/stohastične strategije za maksimiranje nagrade. Spodbujevalno učenje, ki temelji na vrednosti, poskuša maksimirati poljubno funkcijo vrednosti, V (s). Spodbujevalno učenje, ki temelji na modelu, se zanaša na ustvarjanje navideznega modela za določeno okolje in agent se uči izvajati poteze v okviru omejitev okolja. (Sharma, Siddharth, 2020).

4. 4. 2 DEFINICIJE

Akcija (A): Vse možne poteze agenta.
Stanje (-a): Trenutno stanje, ki ga je vrnilo okolje.
Nagrada (R): Takojšnja vrnitev, poslana nazaj iz okolja, da oceni zadnje dejanje.
Politika (π): strategija, ki jo agent uporablja za določitev naslednjega dejanja na podlagi trenutnega stanja.
Vrednost (V): Pričakovani dolgoročni donos v nasprotju s kratkoročno nagrado R.
Vπ(s) je opredeljen kot pričakovani dolgoročni donos trenutne politike stanja π.
Vrednost Q ali vrednost akcije (Q): Vrednost Q je podobna vrednosti V, le da zavzame dodaten parameter, trenutno dejanje a. Qπ (s, a) se nanaša na dolgoročno vrnitev trenutnega stanja s, ki deluje pod politiko π. (Sharma, Siddharth, 2020).

4. 4. 3 PROBLEM VOZIČKA

Problem vozička je znan problem dinamike in teorije nadzora z nihalom, katerega težišče je nad točko vrtenja. To naravno ustvari nestabilen sistem in nihalo bo običajno ostalo obrnjeno navpično navzdol, ne da bi se uporabila sila nihanja ali dinamično krmiljenje. Voziček ima eno os svobode na svoji osi in sistem nima vertikalnega gibanja. Cilj večine sistemov vozičkov je učinkovito vzdrževanje ravnovesja z uporabo različnih sil na vrtilno točko in njeno os gibanja (vodoravna smer).



Slika 3. Problem vozička. (Sharma, Siddharth, 2020)

Pokazali bomo, kako se lahko s pomočjo spodbujevalnega učenja sistem učinkovito uravnoteži. (Sharma, Siddharth, 2020).

4. 4. 4 Q-UČENJE

Na podlagi možnih stanj problema z vozičkom vemo, da bo, če se pravilno odločimo, voziček ostal pokonci in uravnotežen. Tako lahko prepoznamo pare akcija-stanje, ki vodijo do višje nagrade v sistemu vozičkov. Vsak par lahko modeliramo v odvisnosti od verjetnosti nagrade:

$Reward = Q\ (s, a)$.

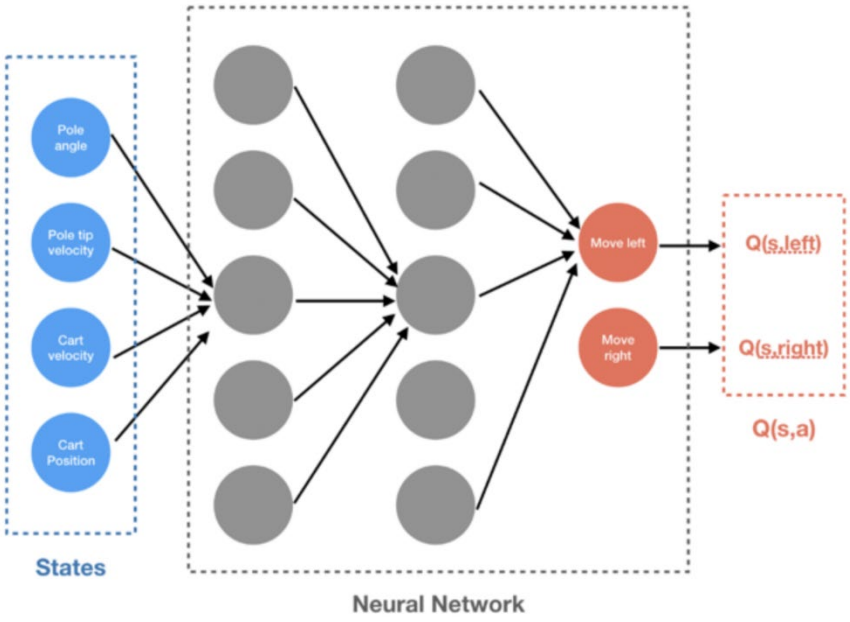
V tem primeru je nagrada znana kot Q-vrednost. Cilj Q-učenja algoritma spodbujevalnega učenja je najti to funkcijo $\boldsymbol{Q\ (s, a)}$, hkrati pa jo iterativno uporabiti za $\boldsymbol{s'}$ (prihodnje stanje). Z drugimi besedami, Q-funkcija predstavlja pričakovano skupno nagrado, ki jo lahko agent prejme z izvajanjem določenega dejanja $\boldsymbol{a}$. Začetno funkcijo Q-učenja lahko predstavimo kot (Sharma, Siddharth, 2020):

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

Ko dobimo neko nagrado $\boldsymbol{r}$ z akcijo $\boldsymbol{a}$, lahko pridemo do naslednjega stanja: $\boldsymbol{s'}$. Ko doseže naslednje stanje ($\boldsymbol{s'}$), agent izvede novo dejanje ($\boldsymbol{a'}$) glede na nagrado. Utež, ki jo želimo osredotočiti na naslednjo nagrado ($\boldsymbol{r'}$), je $\boldsymbol{\gamma}$. Tako enačbo posodobimo na naslednji način:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$$

Na podlagi analize elementov Q-učenja je razvidno, da je mogoče sistem vozičkov učinkovito modelirati z uporabo Q-učenja. Problem z vozičkom ima prostor stanja: **4** dimenzije zveznih vrednosti **($\boldsymbol{\theta}$, $\boldsymbol{l}$, $\boldsymbol{x}$, $\boldsymbol{x2}$)** in prostor delovanja: **2** ločeni vrednosti (**premik desno ali levo**). Vendar pa moramo pri tipičnem Q-učenju spremeniti svoje stanje za vsako manjšo spremembo kota ali položaja vozička, kar bi zahtevalo izjemne zmogljivosti shranjevanja pomnilnika. Poleg tega moramo za uporabo Q-učenja pri uravnoteženju sistema vozičkov približati funkcijo brez modela Q (s, a), kjer je vhod par **stanja ($\boldsymbol{s}$, $\boldsymbol{a}$),** rezultat pa pričakovana nagrada. Ta tehnika približevanja funkcije Q (s, a) je znana kot globoko omrežje Q (**DQN**) in je močnejša pri spreminjanju parametrov in pogostih spremembah stanja. Sledi istemu procesu kot Q-učenje, vendar za izračun Q (s, a) na podlagi usposobljene mreže vozlišč uporablja globoko nevronsko mrežo, o katerih bomo slišali v naslednjih poglavjih (Sharma, Siddharth, 2020):



Slika 4. Globoko Q omrežje za problem vozička. (Sharma, Siddharth, 2020)

4. 5 NEVRONSKE MREŽE – UVOD

Večina problemov strojnega učenja je linearnih, oziroma je za rešitev potrebno oblikovati linearno funkcijo s pomočjo regresije. Pri nekaterih problemih pa ni tako, zato moramo uporabiti zdaj že znani algoritem nevronske mreže, ki deluje kot ogromna funkcija s podanimi vhodnimi parametri in številom izhodov. Ker je sestavljena iz ogromno nevronov in uteži, je izjemno fleksibilna in lahko oblikuje kakršnokoli funkcijo, če le ima dovolj časa za učenje.

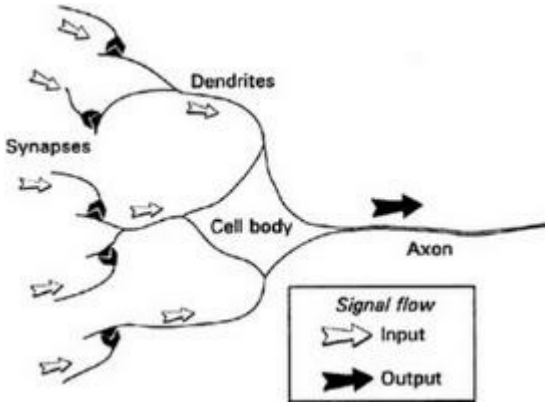
Nevronska mreža, tudi umetna nevronska mreža (angl. artificial neural network) je algoritem za obdelavo informacij, ki oponaša človeške možgane.

Na podlagi učnega vedenja lahko umetne nevronske mreže razvrstimo kot:
• **Nadzorovana nevronska mreža**
• **Nenadzorovana nevronska mreža**
• **Spodbujevalna nevronska mreža** - Kot ljudje, ki se odzivajo na okolje in se učijo iz napak, se spodbujevalna nevronska mreža na podlagi preteklih odločitev uči s kaznimi za napačne odločitve in nagradami za pravilne. Uteži na povezavah, ki izbirajo prave odločitve se okrepijo, medtem ko se tiste, ki sprejemajo napačne, oslabijo. (Alzubi, J., Nayyar, A., Kumar, A., 2018).

4. 5. 1 NEVRON

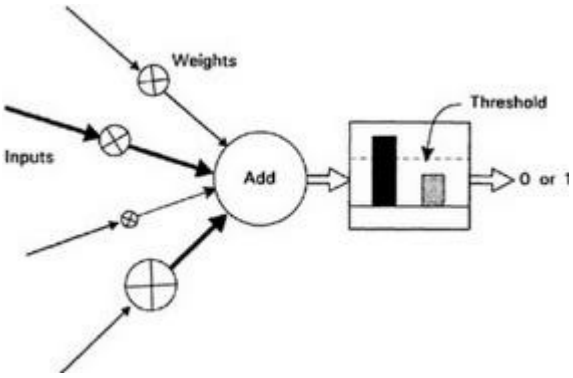
Nevron je najpreprostejša računska enota in glavni gradnik človeških možganov, in njihove interakcije omogočajo možganske procese, kot so učenje. Vsak nevron izračuna preprosto funkcijo. Medtem ko je dejanska dinamika nevronskega računanja zelo kompleksna, lahko na njih gledamo kot da **integrirajo** in se **sprožijo**. To pomeni, da nevron opravi računsko operacijo s svojimi vhodi in se potem sproži, če je ta vrednost višja od določene meje.

Človeški možgani so zgrajeni iz približno 10^11 (100 milijard) živčnih celic ali nevronov. Nevroni komunicirajo prek električnih signalov, ki so kratkotrajni impulzi v napetosti celične stene ali membrane. Med-nevronske povezave posredujejo elektrokemična stičišča, imenovana sinapse, ki se nahajajo na vejah celice, imenovanih dendriti. Vsak nevron ima običajno več tisoč povezav z drugimi nevroni in zato nenehno prejema množico dohodnih signalov, ki sčasoma dosežejo celico. Tu so integrirani ali povzeti skupaj na nek način in, grobo rečeno, če dobljeni signal preseže nek prag, potem se bo nevron "sprožil", ali kot odgovor ustvaril napetostni impulz. To se potem prenaša na druge nevrone prek razvejanega vlakna, znanega kot akson. (Gurney, Kevin, 2004).



Slika 5. Bistvene komponente nevrona. (Gurney, Kevin, 2004)

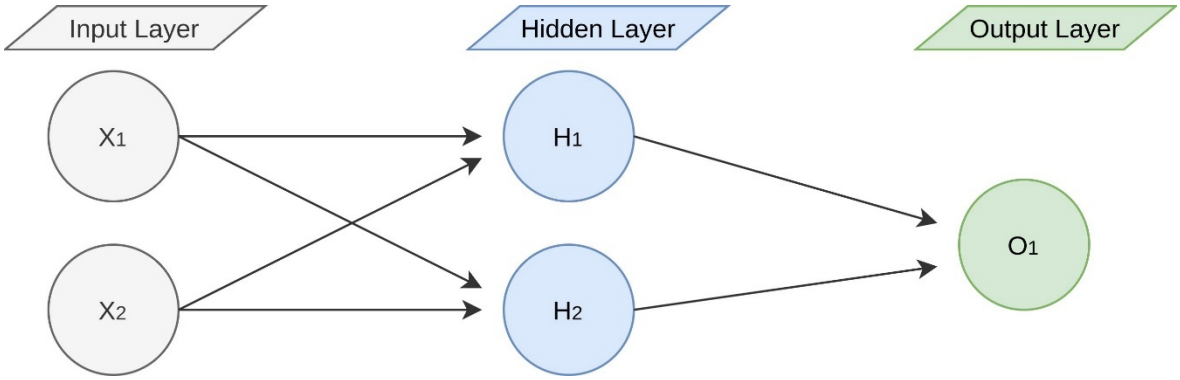
Umetni ekvivalenti bioloških nevronov so **vozlišča** ali enote in prototipni primer je prikazan na sliki. Sinapse so modelirane z eno številko ali **utežjo**, tako da se vsak vhod (vhodni signal) pomnoži z utežjo pred pošiljanjem naprej vozlišče. Tukaj so vsi prihajajoči signali združeni v eno vrednost z enostavnim aritmetičnim seštevanjem. V tipu vozlišča, prikazanem na sliki (Slika 6) je potem uporabljena tako imenovana **aktivacijska funkcija**, ki primerja vrednost s **pragom**; če je vrednost preseže prag, vozlišče proizvede visoko vrednoteno izhodno vrednost (običajno "1"), sicer izpiše nič. Na sliki ta proces predstavljen.



Slika 6. Preprost umetni nevron. (Gurney, Kevin 2004).

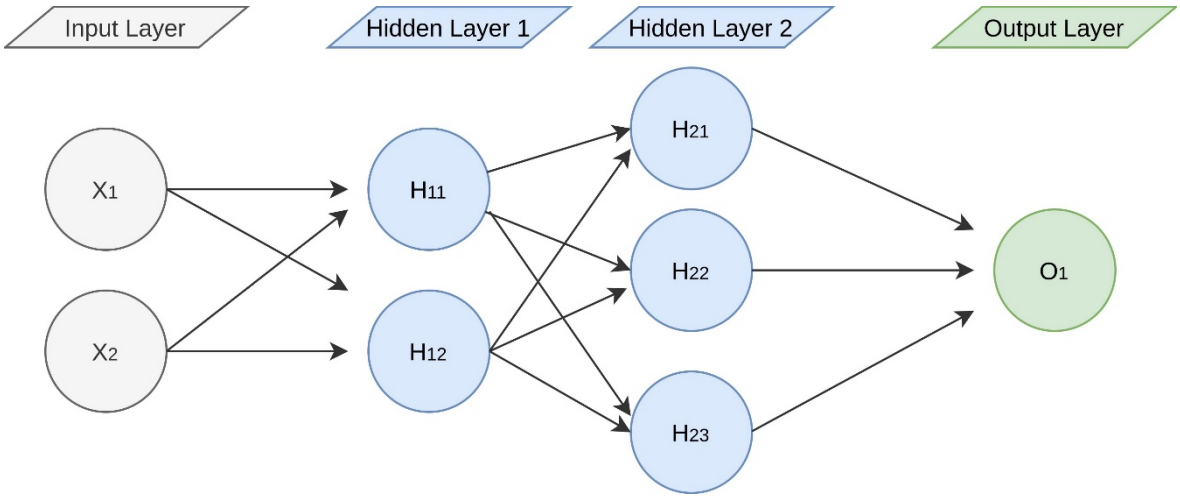
4. 5. 2 VLOGA NEVRONA V NEVRONSKI MREŽI

Nevroni so bistveni sestavni deli nevronske mreže. Natančneje, nevronske mreže nastanejo s povezovanjem enega nevrona z vsakim drugim nevronom, ampak za razliko od človeškega nevronskega sistema so nevroni v nevronskih mrežah med seboj povezani v **plasti** ali **nivoje**. Čas obdelave vseh povezav bi bil izjemno velik, če bi bil vsak nevron povezan z vsakim drugim nevronom. Da bi zmanjšali čas obdelave in prihranili računalniško moč, se v nevronski mreži uporabljajo nivoji. Spodnja slika predstavlja osnovno strukturo nevronske mreže s tremi nivoji. (Cansiz, Sergen, 2020).



Slika 7. Osnovana struktura nevronske mreže s tremi nivoji. (Cansiz, Sergen, 2020)

Kot je vidno, so na zgornji sliki 3 plasti: vhodni nivo, skriti nivo in izhodni nivo. Te tri plasti so glavne plasti nevronske mreže. Nevroni v vhodni plasti predstavljajo vsako spremenljivko v naboru podatkov. Nevron v izhodni plasti predstavlja končno napovedano vrednost po prehodu vhodnih vrednosti v vsak nevron v skritih nivojih. Čeprav je samo en vhodni in en izhodni nivo, je število skritih nivojev poljubno. Zato je delovanje nevronskih mrež odvisno od števila nivojev in števila nevronov v posameznem nivoju. Napovedovalne zmogljivosti omrežja s 3 skritimi nivoji s po 3 nevroni v vsakem nivoju in omrežjem z enim nevronom enega nivoja so lahko drugačne. Torej, kako izgleda omrežje, če dodamo še skriti nivo s tremi nevroni? Spodnja slika prikazuje mrežo z dvema skritima nivojema. V omrežje lahko dodamo več skritih nivojev in nevronov. Za to ni nobene omejitve. Vendar ne smemo pozabiti, da dodajanje več nivojev in nevronov ne pomeni, da bo naša nevronska mreža boljše napovedovala. Prav tako se bo, če povečamo število skritih plasti in nevronov, povečal čas treninga zaradi izračunov v vsakem nevronu. Najti moramo čim boljšo mrežno strukturo za naše omrežje. (Cansiz, Sergen, 2020).

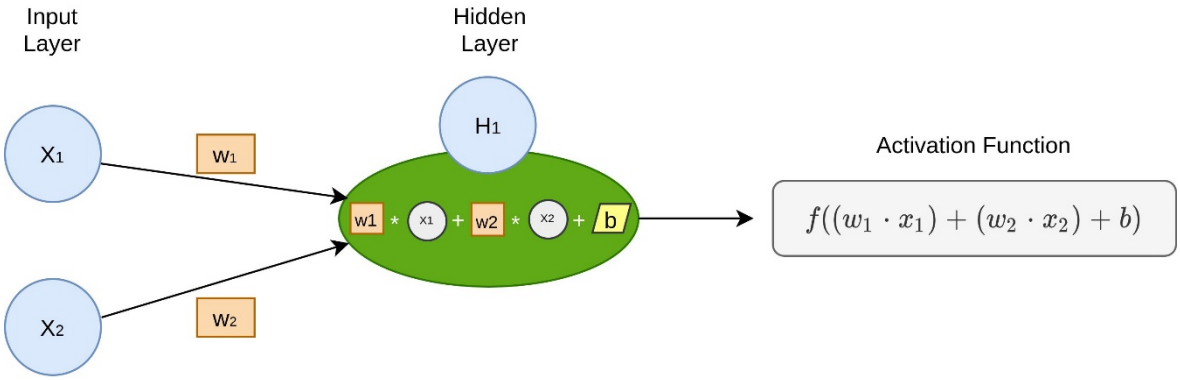


Slika 8. Nevronska mreža z dvema skritima nivojema. (Cansiz, Sergen, 2020).

4. 5. 3 HRANJENJE NEVRONOV

Nevronske mreže delujejo s ponovitvami oziroma iteracijami in vsaka ponovitev izboljša (upamo) model, da ta doseže najboljšo napoved. Hranjenje nevronov oziroma posredovanje je torej glavni proces, ki trenira našo mrežo. Ta proces se včasih imenuje tudi "podajanje naprej" (angl. Feed forward), kar pomeni prejemanje podatkov iz prejšnjih povezanih nevronov, izvajanje izračunov s temi podatki (4.5.1 Nevron) in pošiljanje rezultata naslednjim povezanim nevronom. Ko je končni izračun opravljen in prenesen v izhodne nevrone, se iz nabora podatkov vzame še eno opazovanje/podatek in gre spet v svet nevronov in postopek se ponovi. Ta postopek se nadaljuje, dokler se napoved našega omrežja ne približa dejanski napovedani vrednosti. Tu je najpomembneje, kakšni izračuni se izvajajo v nevronih. Ne smemo pozabiti, da bomo s pomočjo tega omrežja napovedovali. Zato je potrebno nevrone »stehtati« oziroma oceniti, da bomo lahko najbolje napovedali. Zdaj pa si poglejmo ta postopek poglobljeno. (Cansiz, Sergen, 2020).

Predpostavimo, da imamo nevronsko mrežo z 1 vhodno plastjo z 2 vhodnima nevronoma, 1 skrito plastjo z 2 nevronoma in 1 izhodno plastjo z 1 nevronom. Torej, najprej lahko pregledamo proces hranjenja nevrona v skriti plasti.



Slika 9. Proces hranjenja nevronov v skritih nivojih. (Cansiz, Sergen, 2020).

Na zgornji sliki sta v vhodni plasti dva nevrona. To pomeni, da imamo v svojem naboru podatkov dve spremenljivki, ki jih bomo uporabili za ocenjevanje. Kot lahko vidite, ima vsaka povezava med vhodnimi nevroni in nevronom v skriti plasti utežne vrednosti »**W**«. Te vrednosti predstavljajo električne impulze, ki jih posamezen nevron posreduje vsakemu drugemu nevronu, na katerega je povezan. V prvem koraku je mogoče vrednosti "**W**" in "**b**" generirati naključno med **0** in **1**. Med ponovitvami (koraki) se te vrednosti posodobijo, da dosežejo najbolje predvideno vrednost. Te ponovitve bomo videli v naslednjih razdelkih. Nevron **H1** napajajo nevroni **X1** in **X2**, zato ima težo **W1** in **W2** za **X1** in **X2**. Če vidite del izračuna, se vrednost vsakega nevrona, ki prihaja iz vhodne plasti, pomnoži z njegovo težo in vsoto seštejemo. Vendar pa obstaja še ena vrednost, ki je predstavljena kot "**b**". Ta vrednost je tako imenovani **bias** nevrona in to vrednost seštejemo s pomnoženimi utežmi. Po operaciji seštevanja rezultat preide do aktivacijske funkcije. Aktivacijska funkcija naredi nekakšno operacijo transformacije.
Dobljena vrednost iz aktivacijske funkcije bo končna vrednost tega nevrona v trenutnem koraku, ki pa more biti **0** ali **1,** kar nam pove, ali se nevron **sproži** (1) ali **ne** (0). (Cansiz, Sergen, 2020).

Bolj natančno je ta proces in matematika povezana z njim razložena spodaj.

Značaj akcijskega potenciala nevrona "vse ali nič" je lahko označen z uporabo dvo-vrednostnega signala. Takšni signali se pogosto imenujejo **binarni** ali '**Boolean**' in običajno zavzamejo vrednosti "**0**" in "**1**". Če imamo torej vozlišče, ki prejema ***n*** vhodnih signalov $x_1, x_2, ..., x_n$, lahko ta dobi le vrednosti "0" ali "1". Sinapse so modelirane s preprostim množenjem dohodnega signala z vrednostjo uteži. Zato imamo ***n*** uteži $w_1, w_2, ..., w_n$ ki tvorijo ***n*** produktov $w_1x_1, w_2x_2, ..., w_nx_n$. Vsak produkt je lahko negativen ali pozitiven, odvisno od vrednosti uteži. Potem so te vrednosti združene v procesu, ki naj bi posnemal dogajanje v aksonu v biološkem nevronu. To je storjeno tako, da so vrednosti preprosto seštete v aktivacijsko vrednost ***a***, kot je prikazano spodaj. (Gurney, Kevin, 2004):

$$a = w_1x_1 + w_2x_2 + ... + w_nx_n$$

Za posnemanje ustvarjanja akcijskih potencialov potrebujemo mejno vrednost **θ** (grško črko teta), tako da, če aktivacija preseže (ali je enaka) to vrednost **θ**, vozlišče odda "1" (akcijski potencial) in če je manj kot **θ**, potem odda "0". Rezultat je označen s simbolom **y**. Ta relacija se včasih imenuje stopničasta funkcija (step function). Ta vrsta umetnega nevrona je znana kot logična enota praga (TLU). (Gurney, Kevin, 2004).

Funkcionalnost TLU je bolj priročno predstaviti s simboli in ne v grafični obliki. V prejšnji enačbi že imamo zapisan ta obrazec, a ga lahko zapišemo bolj kompaktno, kot je prikazano spodaj. Zapis je tukaj pomemben. Majhna števila, ki se uporabljajo za označevanje vhodov in uteži se imenujejo indeksi. S simboličnim pisanjem indeksa (prej kot številčno) se lahko na količine sklicujemo splošno, tako da na primer x_i označuje generični ali ***i***-ti vhod, kjer se domneva, da je ***i*** lahko katero koli celo število med **1** in **n**. Podobno velja za uteži w_i. Z uporabo teh idej je možno predstaviti prejšnjo enačbo v obliki

$$a = \sum_{i=1}^{n} w_i x_i$$

kjer ∑ (velika grška črka sigma) pomeni vsoto. Zgornje in spodnje število glede na ∑ označujeta zgornjo in spodnjo mejo seštevanja in nam povesta, da indeks ***i*** teče od **1** do **n**. Včasih so omejitve izpuščene, ker so že bile definirane drugje in indeks seštevanja (v tem primeru i) preprosto zapišemo pod ∑. (Gurney, Kevin, 2004).

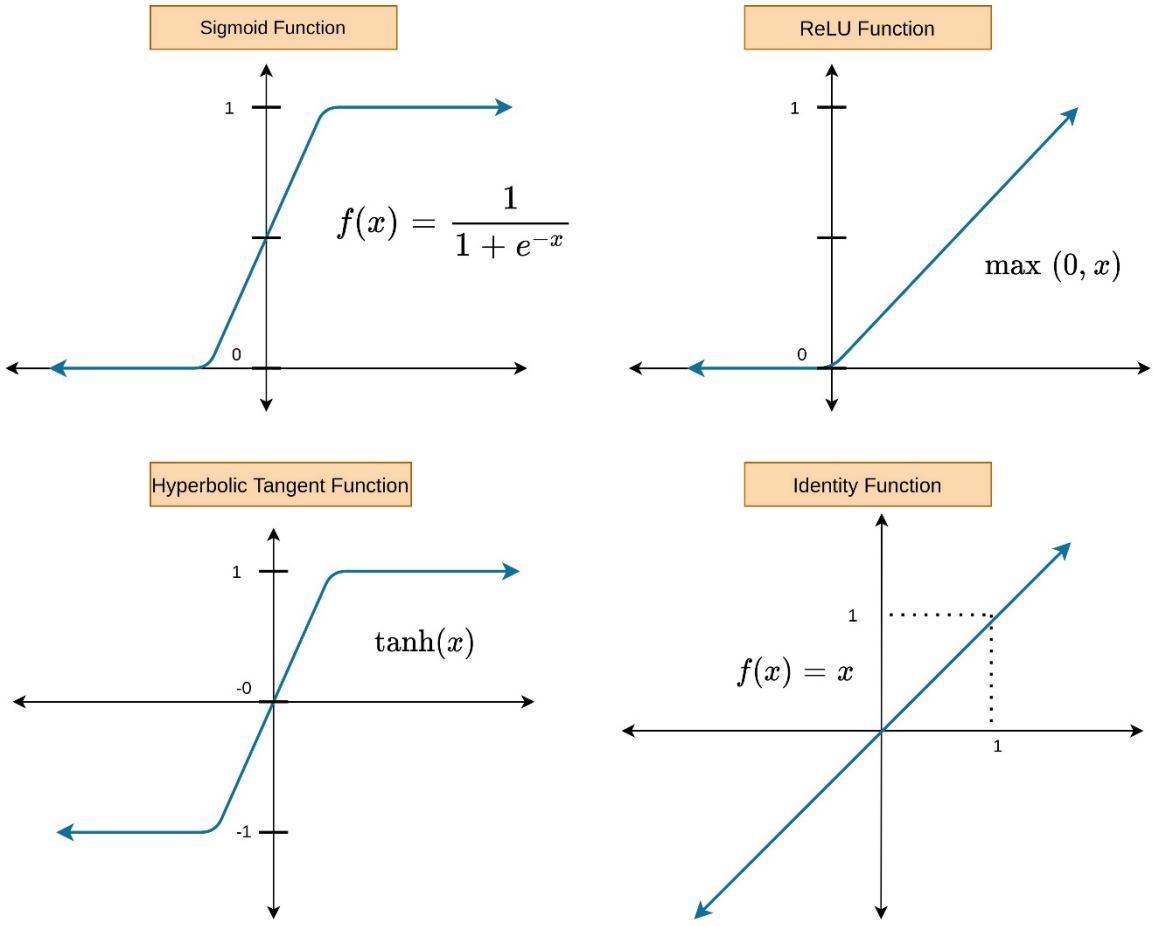
Izhodno vrednost **y** pa lahko pridobimo z

$$y = \begin{Bmatrix} 1, a \geq \theta \\ 0, a < \theta \end{Bmatrix}$$

Seveda pa je to zelo preprost primer, ki ni vedno učinkovit, zato uporabimo aktivacijsko funkcijo primerno za naš problem.

4. 5. 4 AKTIVACIJSKA FUNKCIJA

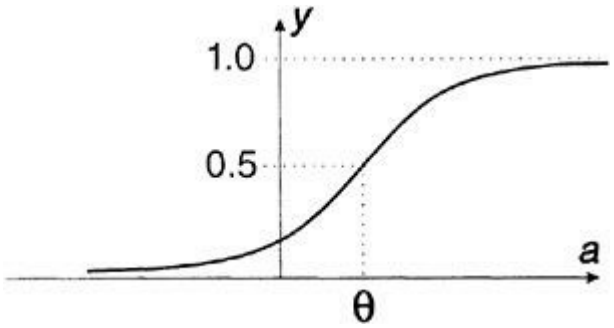
Obstajajo različne aktivacijske funkcije, ki jih lahko izberemo glede na vrednost, ki jo želimo predvideti. Predpostavimo, da želimo predvideti **2 razreda**, označena kot **0** in **1**. Torej, kar potrebujemo, je vrednost verjetnosti med 0 in 1, da se lahko odločimo za predvideni razred. Če je predvidena vrednost manjša od 0,5 (kar pomeni, da je blizu razredu 0), lahko rečemo, da je napovedana kot 0. Če je večja od 0,5, lahko rečemo, da je predvidena kot 1. Brez aktivacijske funkcije je mogoče dobiti kakršnekoli vrednosti, ki nevronu ne pomenijo nič. Nevron razume samo vrednosti med 1 in 0. Zato uporabimo aktivacijsko funkcijo, ena od teh je "Sigmoidna funkcija". Druge pogosto uporabljene aktivacijske funkcije so predstavljene na spodnji sliki. (Cansiz, Sergen, 2020).



Slika 10. Štiri najpogostejše aktivacijske funkcije. (Cansiz, Sergen, 2020).

Če pogledate zgornjo sliko, obstajajo tudi druge aktivacijske funkcije. Na primer hiperbolično tangentno funkcijo lahko uporabimo, kadar potrebujemo izhodno vrednost med -1 in 1. Funkcijo ReLU lahko uporabimo, če izhod ne sme biti manjši od 0, v tem primeru pa funkcija vrne izhod, kakršen je bil vhod. Obstaja pa tudi »Identiteta«, ki vrednost pusti takšno, kot je. Zadnji dve uporabljamo pri problemih regresije, prvi dve pa pri klasifikaciji. Glede na spremenljivko, ki jo bomo predvideli, je treba izbrati eno od teh funkcij. (Cansiz, Sergen, 2020).

Ker rabimo pri večini primerov strojnega učenja verjetnost, da določen nabor lastnosti pripada določenemu razredu (klasifikacija), uporabimo **sigmoidno funkcijo**. Aktivacijska vrednost **a** je lahko ogromna, zato rabimo funkcijo, ki nam vrne vrednost med 0 in 1.
Na sliki je graf **sigmoidne funkcije σ(x).**



Slika 11. Sigmoidna funkcija. (Gurney, Kevin, 2004).

Kot lahko vidimo, je graf simetričen prek vrednosti 0.5 na y-osi. Večja (ali manjša) kot je vhodna vrednost **x**, bolj se bo **y** približeval **1** (ali 0, če se **x** veča v negativni smeri). (Gurney, Kevin, 2004).

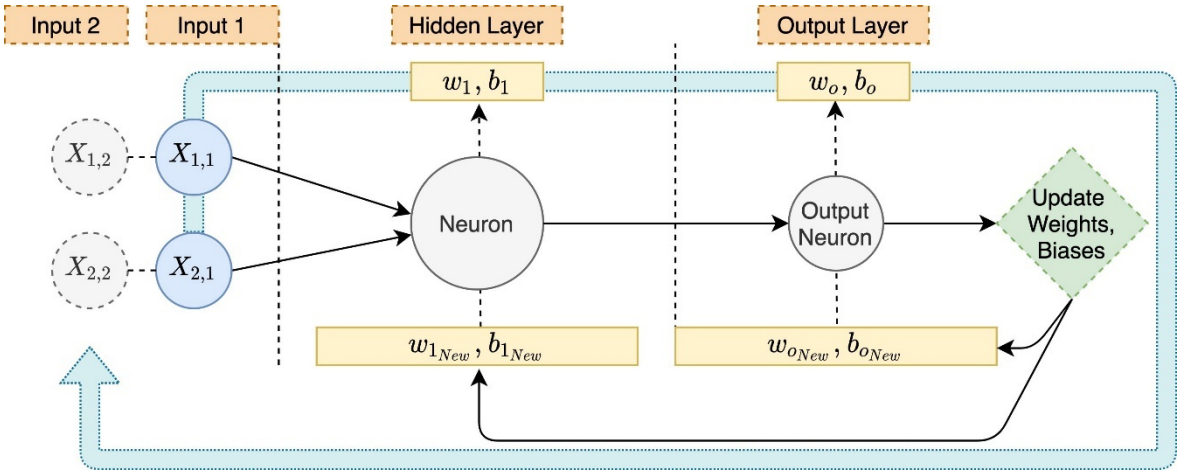
Sigmoidna funkcija je podana z enačbo

$$y = \sigma(a) \equiv \frac{1}{1 + e^{-(a-\theta)/\rho}}$$

kjer je **y** izhodna vrednost nevrona, **a** aktivacijska vrednost, **θ** pa mejna vrednost, ki je navadno 0,5 (če bo aktivacijska vrednost večja, bo izhodna vrednost 1, če ne 0).
V tem primeru je **e** (približno 2,7183…) matematična konstanta, ki ima tako kot **π** neskončno decimalnih mest. Količina **ρ** (grška črka rho) določa obliko funkcije. Velika vrednost naredi krivuljo bolj plosko, medtem ko majhne vrednosti krivuljo delajo bolj strmo. V mnogih besedilih je ta parameter izpuščen, tako da mu je implicitno dodeljena vrednost 1. (Gurney, Kevin, 2004).

4. 5. 5 NADGRAJEVANJE NEVRONOV IN KAKO SE UČIJO NEVRONSKE MREŽE

To je najverjetneje najbolj kritičen del teorije pri tej raziskovalni nalogi. Ta odsek smo poimenovali »Nadgrajevanje nevronov…«, ker se po vsaki ponovitvi v mreži spremenijo vrednosti uteži nevronov. Torej jih nadgradimo na višjo raven z namenom boljše napovedi. Kot je bilo že omenjeno, je treba uteži in biase nevronov posodobiti, tako da se upošteva razlika med napovedano in dejansko vrednostjo po vsaki končani ponovitvi. Ta postopek se imenuje povratno širjenje ali **vzvratno razširjanje** (angl. Back-propagation), algoritem, ki ga bomo uporabili za to, pa je **gradientni spust** (angl. Gradient descent). Po vsaki iteraciji je potrebno opraviti tudi povratno širjenje. S pomočjo tega algoritma se nevronske mreže učijo. Naslednja slika predstavlja postopek povratnega širjenja. (Cansiz, Sergen, 2020):



Slika 12. Povratno širjenje. (Cansiz, Sergen, 2020)

Da bi posodobili uteži in biase, moramo najprej najti razliko med **dejansko** in **predvideno vrednostjo**. Dve najpogostejši funkciji, ki se za to uporabljata, sta **napaka povprečnega kvadrata** in **navzkrižna entropija**. V nekaterih primerih se uporablja tudi **vsota napak na kvadrat**. Te funkcije so tako imenovane **stroškovne funkcije**. Medtem ko je pri regresijskih problemih bolj učinkovita napaka povprečnega kvadrata (**MSE, Mean Squared Error**), je za klasifikacijske probleme večinoma uporabljena navzkrižna entropija. Spodnja enačba prikazuje funkcijo napake povprečnega kvadrata:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_{true} - Y_{prediction})^2$$

Kot je razvidno iz zgornje slike, MSE deluje tako, da ugotovi vse razlike napovedanih in dejanskih vrednosti, jih kvadrira (za večjo učinkovitost) in sešteje, na koncu pa deli s številom vseh vrednosti in vrne povprečno napako oziroma **strošek**. (Cansiz, Sergen, 2020).

Ko imamo vrednost **izgube** moramo narediti še naslednje. Ugotoviti moramo, kako bomo spremenili uteži in biase, tako da bomo upoštevali njihove učinke na izgubo (strošek). Zato potrebujemo metodo, ki to lahko stori. Ta metoda se v matematiki imenuje parcialni odvod. Parcialni odvodi **stroškovne funkcije L** (MSE) z utežmi in biasi nam povejo, kako uteži in biasi vplivajo na **strošek**! Izvesti moramo naslednje operacije:

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial w_5}, \frac{\partial L}{\partial w_6}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$$

Zgornja slika prikazuje parcialne odvode za vsako utež in bias (v tem primeru 6 uteži in 3 biasi) glede na **stroškovno funkcijo L**. (Cansiz, Sergen, 2020).

Ta del je matematično dokaj zahteven. Nadaljevali bomo z odvodom za **W1** (**dL / dW1**). Da bi našli ta parcialni odvod, lahko uporabimo spodnjo formulo. **L** je funkcija izgube, **H1** predstavlja vrednost **H1** nevrona po ocenjevanju, **predvidevanje Y (**$Y_{prediction}$**)** predstavlja rezultat **O1** nevrona in **W1** je vrednost uteži, ki je povezana na **H1** nevron.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial Y_{prediction}} \cdot \frac{\partial Y_{prediction}}{\partial H_1} \cdot \frac{\partial H_1}{\partial w_1}$$

Želimo ugotoviti, kako se spremeni utež **W1**, glede na spremembo v stroškovni funkciji **L.**
Da bi rešili zgornjo enačbo, moramo nadaljevati s formulami za $Y_{Prediction}$, **H1** in **L**. Po tem lahko dobimo parcialne odvode vseh. Če pogledamo prvo formulo za vrednost $Y_{Prediction}$, lahko vidimo, da gre za končni izračun predvidene vrednosti v nevronu **O1**. (Cansiz, Sergen, 2020).

$$Y_{prediction} = O_1 = (w_5 \cdot H_1) + (w_6 \cdot H_2) + b_3$$

In formula za vrednost **H1** nevrona, je:

$$H_1 = (w_1 \cdot x_1) + (w_2 \cdot x_2) + b_1$$

(Povezave med nevroni na tej in zgornji enačbi so izbrane poljubno.)
Zadnja formula, ki jo potrebujemo, je formula za odvod stroškovne funkcije L (MSE). Ne smemo pozabiti, da v vsakem koraku izvedemo samo 1 opazovanje. Torej bo vrednost "n" enaka 1 (Cansiz, Sergen, 2020):

$$MSE = L = \sum_{i=1}^{1} (Y_{true_i} - Y_{prediction_i})^2$$

$$L = (Y_{true} - Y_{prediction})^2$$

$$L = (Y_{true})^2 - 2 \cdot Y_{true} \cdot Y_{prediction} + (Y_{prediction})^2$$

(Kjer je Y_{True} dejanska vrednost in $Y_{prediction}$ predvidena vrednost nevrona.)

Začnemo lahko s prvim parcialnim odvodom, ki je »**dL / dY$_{Prediction}$«**

$$\frac{\partial(L)}{\partial(Y_{prediction})} = \frac{\partial\left((Y_{true})^2 - 2 \cdot Y_{true} \cdot Y_{prediction} + (Y_{prediction})^2\right)}{\partial(Y_{prediction})}$$

$$= -2Y_{true} + 2Y_{prediction}$$

$$= -2(Y_{true} - Y_{prediction})$$

$$\frac{\partial(L)}{\partial(Y_{prediction})} = -2(Y_{true} - Y_{prediction})$$

Osnovna ideja te operacije je ugotoviti, kako **Y$_{prediction}$** vpliva na **stroškovno funkcijo L**. Naslednji delni odvod je **"dY$_{prediction}$ / dH1"**. Običajno je **H1** funkcija, ki vrne vrednost po izvedbi aktivacijske funkcije v nevronu.

$$Y_{prediction} = O_1 = (w_5 \cdot H_1) + (w_6 \cdot H_2) + b_3$$

$$\frac{\partial(Y_{prediction})}{\partial(H_1)} = w_5$$

Zadnji odvod je **„dH1 / dW1“:**

$$H_1 = (w_1 \cdot x_1) + (w_2 \cdot x_2) + b_1$$

$$\frac{\partial H_1}{\partial w_1} = x_1$$

In ko sestavimo vse koščke:

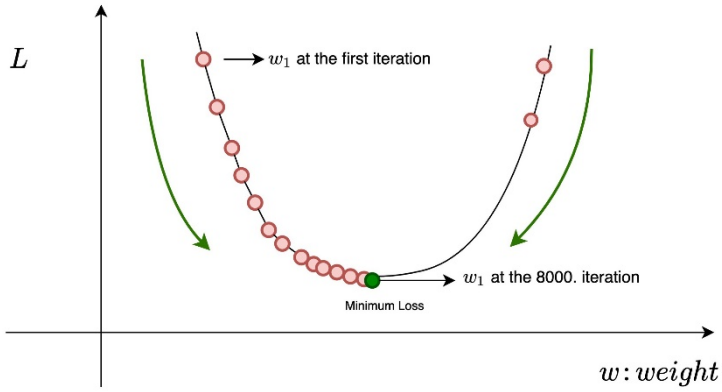
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial Y_{prediction}} \cdot \frac{\partial Y_{prediction}}{\partial H_1} \cdot \frac{\partial H_1}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = -2(Y_{true} - Y_{prediction}).\, w_5 \cdot x_1$$

Ko imamo parcialne odvode, jih moramo pomnožiti s **stopnjo učenja**. Po tem bi morali ta rezultat odšteti od originalnih uteži. **Stopnja učenja** pove, kako se bodo uteži spreminjale. Spremembe vrednosti uteži pri visoki stopnji učenja bodo veliko večje kot pri nizki stopnji učenja. Zato moramo določiti optimalno stopnjo učenja za naše omrežje. (Cansiz, Sergen, 2020).

Vemo kako vrednost **W1** vpliva na **strošek**. Odločiti se moramo, kako bomo spremenili vrednost **W1**, da bomo izgubo čim bolj zmanjšali. Zato bomo uporabili že prej omenjeni algoritem gradientnega spusta, bolj natančno **Stochastic Gradient Descent (SGD)**. Pravzaprav je bila operacija parcialnih odvodov, ki smo jo opravili prej, del gradientnega spusta.

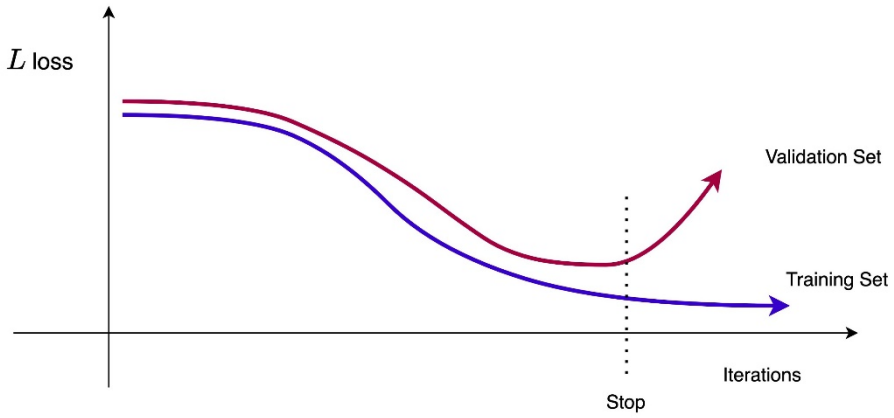
V SGD vzamemo posamezne vzorce iz nabora podatkov, jih prenesemo v mrežo in ugotovimo, kako bomo spreminjali uteži in biase, da bomo izgubo čim bolj zmanjšali. Ta postopek poteka v vsaki iteraciji, dokler ne dosežemo najnižjega stroška. To pomeni, da če čez nekaj časa ne moremo več zmanjšati stroška, lahko prenehamo s treningom. Na spodnji sliki je to prikazano. (Cansiz, Sergen, 2020).



Slika 13. Gradientni spust. (Cansiz, Sergen, 2020).

4. 5. 6 PROBLEMI VISOKE VARIANCE

Obstaja tudi še ena težava, za katero moramo poskrbeti. To je težava visoke variance. Do te težave pride, ko se naš model nevronske mreže odlično obnese na primerih za trening, ampak ne deluje dobro z novimi podatki. To pomeni, da lahko naša mreža doseže minimalno izgubo po tisočih ponovitvah, vendar ne deluje dobro na preizkusnem naboru, ki ga ločimo od nabora za učenje. V takih primerih lahko iz naših podatkov določimo nabor za preverjanje in nadzorujemo tudi strošek za ta nabor. Če se izguba pri naboru za preverjanje še povečuje, medtem ko se izguba pri tistem za učenje zmanjšuje, lahko trening končamo. (Cansiz, Sergen, 2020).



Slika 14. Visoka varianca (Cansiz, Sergen, 2020).

5 EKSPERIMENTALNI DEL

5. 1 METODOLOGIJA

V eksperimentalnem delu bova poskusila dokazati najino hipotezo, ki je:
Domnevamo, da se algoritmi strojnega učenja lahko naučijo igrati računalniško igro Kača brez začetnega znanja, tako da so po učenju zmožni zbirati točke (jesti hrano).
Prav tako bova izmerila, kako natančno (koliko točk lahko zbere) se lahko nauči algoritem spodbujevalne nevronske mreže igrati arkadno igro Kača.

Programe sva napisala v programskem jeziku Python, ki je eden najpreprostejših programskih jezikov (zato pa tudi ni tako hiter kot C ali Java). Python ima veliko knjižnic za strojno učenje, kot so *Tensorflow*, *Neat* in pa *PyTorch*, ki sva ga uporabila pri tej nalogi. Program je bil originalno napisan v Pythonu 3.8.6.

Ker sva uporabila algoritem spodbujevalne nevronske mreže (Globoka Q mreža), sva uporabila glavne principe spodbujevalnega učenja. Torej, agent, ki operira v nekem okolju in za izvedeno akcijo, ki se ravna po politiki glede na stanje, ki ga vrne okolje, prejme povratno informacijo (nagrado, ki je lahko tudi negativna).
Ne bova navedla dejanske kode (ki bo na USB-ključku), ampak bova program razložila z besedami.

5. 2 IGRA KAČA

Kača je ena prvih videoiger na svetu in je svetovno znana. Nastala je leta 1976. Prvotno je bila to arkadna igra, ki se jo je upravljalo z dvema gumboma, ki sta kači povedali, ali naj se obrne levo ali desno.
Čez čas je postala glavna zabava na telefonih Nokie. Glavni cilj je zbrati čim večje število točk, ki so pridobljene s pobiranjem hrane, ki je generirana na naključni poziciji v mreži.

Dimenzije mreže so se na različnih telefonih in igralnih avtomatih razlikovale. Igra se konča, ko se kača zaleti v rob ali svoj rep (kača se podaljša, ko poje hrano).
Na začetku sva naredila program, ki je omogočal osebi, ki ga uporablja, igrati igro Kača in zbirati točke. To je bilo dokaj preprosto, ker sva morala samo definirati spremenljivke in razred "Kača", kjer sva definirala tudi neke osnovno funkcije (začetne spremenljivke, premikanje, ustvari hrano, preveri, če je hrana zaužita ...). Kasneje sva ta program povezala še z drugimi.

5. 3 IMPLEMENTACIJA AGENTA

Potem sva implementirala agenta, ki je omenjen v poglavju o spodbujevalnem učenju. Kot vemo, mora agent delovati v nekem okolju (igra) in za svoja dejanja dobivati povratne informacija (nagrade ali kazni).

Na podlagi le teh se lahko izboljšuje in uči iz lastnih napak oziroma izboljšuje svojo politiko (strategijo). Njegove možnosti izboljšave so neskončne. Tudi ljudje se lahko iz napak naučimo prav toliko kot iz dosežkov. (Michael Jordan je v enem od intervjujev rekel, da se je naučil več iz zgrešenih metov, kot iz zadetih.) Agent upravlja kačo prek modela, kateremu so podani parametri, kot so smer, pozicija hrane, … Od modela dobiva tudi povratne informacije: ali je agent dobil točko, ali se je zaletel, … Nato agent s to povratno informacijo posodobi svoj način igranja.

V najinem primeru agent svoje informacije (rekorde, število odigranih iger, ...) zapisuje v besedilni dokument in pa datoteko *tensors.pth*, kamor shrani vrednosti uteži in biasov nevronske mreže.

5. 4 RAZMERJE MED RAZISKOVANJEM IN IZKORIŠČANJEM

Pri spodbujevalnem učenju moramo ustvariti razmerje med raziskovanjem in izkoriščanjem, kar pomeni, da bo model v določenih primerih izbral naključno potezo (raziskovanje) in se s tem morebitno izboljšal (če pa ne, pa bo tako ali tako dobil negativno nagrado). Seveda pa ne more brezglavo tavati in izbirati naključnih potez, saj s tem ne bi izkoristil svojega znanja. To razmerje torej pomeni, v koliko primerih se bo model odločil za raziskovanje in v koliko za izkoriščanje znanja.
V najinem programu sva prvotno izbrala algoritem:

1. *epsilon = 80 - število iger*
2. *če je naključno število med 0 in 200* ***manjše*** *od epsilona, izberi naključno potezo.*

Torej model po 80 igrah ne bo več raziskoval, saj bo epsilon negativen, število med 0 in 200 pa ne more biti negativno. To se je na začetku zdelo v redu, kasneje pa sva epsilon spremenila na:

epsilon = 300 - število iger.

5. 5 MODEL SPODBUJEVALNE NEVRONSKE MREŽE

V našem primeru ta agent uporablja nek "model", ki mu da informacijo o tem, kaj naj naredi iz informacije o trenutnem stanju. Agent je v bistvu kot nekakšen posrednik, za njim pa se skriva stric iz ozadja - nevronska mreža. Postopek deluje tako, da agent prejme stanje iz okolja, ga posreduje modelu, ta pa izračuna naslednjo potezo in jo vrne agentu, ki jo nato izvede, okolje pa se spremeni. Ta postopek se ponavlja, model pa izboljšuje, saj poskuša doseči čim večje število točk.

Prvo sva "uvozila" vse knjižnice, ki jih bova uporabljala. To so bili že prej omenjeni "PyTorch" in njegove pod-knjižnice. Ustvarila sva dva razreda: LinearQNet (uporablja se pri skoraj vsakem takšnem programu), ki uporabi knjižnico "Pytorch", ki omogoči ustvarjanje nevronskih mrež. Notri sva definirala *__init__* (Initalizacija. Podane so začetne spremenljivke razreda). Nato še *forward* (Hranjenje nevronov. Uporabila sva aktivacijsko funkcijo *ReLU*). In pa funkcijo za shranjevanje modela v *.pth* datoteko.

Nato sva naredila še razred *QTrainer,* s pomočjo katerega se lahko model uči. S pomočjo PyTorch-ove funkcije sva lahko uporabila različno število parametrov.

5. 6 PRVI POSKUSI

Parametri modela so zmeraj 0 ali 1 (0 pomeni ne, 1 pa da). Lahko rečemo, da je "0" negativna, "1" pa pozitivna vrednost.
Prvo sva sprogramirala preprost model, ki prejme 12 vhodnih parametrov (12 nevronov v vhodnem nivoju):

1. nevarnost naravnost (ali je pred njo rob oziroma njen rep),
2. nevarnost levo (ali je levo od nje rob oziroma njen rep).
3. nevarnost desno (ali je desno od nje rob oziroma njen rep),
4. štiri parametri za smer, od katerih ima vedno samo en pozitivno vrednost (tista smer, kamor je kača obrnjena),
5. štiri parametri za relativno pozicijo hrane glede na kačo, od katerih sta vedno dva pozitivna (npr. hrana_spodaj in hrana_desno), razen če je kača v isti vrstici ali stolpcu kot hrana (takrat je pozitiven samo eden).

Pozicija kače je pozicija njene glave oziroma prvega člena.
Že samo s temi parametri se je kača odrezala solidno in podrla vse najine osebne rekorde. Nisva pričakovala tako hitrega napredka.
Potem sva poskusila dodati še parametre, ki preverjajo, ali obstaja nevarnost v oddaljenosti dveh kock. Ker to ni izboljšalo napredka, sva poskusila drugo metodo.

5. 6. 1 NOVA METODA

Odločila sva se, da modelu ne bova podala binarne vrednosti za preverjanje nevarnosti v bližini, ampak bova merila **razdaljo do nevarnosti v vsaki smeri.** To se žal ni obrestovalo, saj se model zaradi neznanega razloga ni učil. Prav tako pa to ponudi tudi prostor za naslednje raziskave. Na tem mestu sva zaključila raziskovalno nalogo.

5. 7 REZULTATI

Spodaj so predstavljeni rezultati. Podala sva število zbranih točk in povprečno število zbranih točk, ki pove, kolikokrat je model uspel s kačo pobrati/pojesti hrano.

Tabela 1. Rezultati modela nevronske mreže (lasten arhiv).

Poskusi: ↓ / Igranih iger: →	100	300
6x6 mreža, pregleduje 1 polje stran:	Rekord: 20 Povprečen rezultat: 5.76	Rekord: 21 Povprečen rezultat: 9.27
10x10 mreža, pregleduje 1 polje stran:	Rekord: 27 Povprečen rezultat: 7.69	Rekord: 37 Povprečen rezultat: 13.12
Mreže s pregledovanjem 2 polji stran:	Tudi s pregledovanjem za dve polji se ni izšlo, saj se je kača nenehno vrtela v krogih in ni napredovala.	/

6 ZAKLJUČEK, SKLEPI

Ko sva se odločila za izdelavo raziskovalne naloge, nisva dobro vedela, za kaj pravzaprav gre. Kasneje sva ugotovila, da je izdelovanje raziskovalne naloge zelo zabavno, a včasih utrujajoče. V glavnem sva se zabavala. V raziskavi sva ugotovila, da se model nevronske mreže lahko dokaj dobro naučiti igrati igro Kača in se z generacijami izboljšuje.

Raziskava je bila delno uspešna, saj sva naredila program, ki upravlja kačo in doseže nek rezultat, ni nama pa uspelo narediti takega, ki bi igro dokončal, kar je dobra motivacija za naprej. Poskusila sva tudi drug način, kjer sva modelu podala razdaljo do nevarnosti v vsaki smeri, vendar je bilo to zelo neučinkovito. Vprašanje je kako model sprogramirati tako, da bo model igro dokončal.

Hipoteza je bila potrjena, saj se je algoritem strojnega učenja naučil igrati računalniško igro Kača brez začetnega znanja tako, da je zmožen zbirati točke (jesti hrano).

7 PRILOGE

[1] SANDERSON, Grant. Deep Learning. Dostopno na spletnem naslovu; <https://www.3blue1brown.com/neural-networks>.

[2] Priložena povezava s programski kodo in komentarji:
<https://github.com/FlippySleepy/Raziskovalna-naloga>.

8 LITERATURA IN VIRI

[1] ALZUBI, Jafar. NAYYAR, Anand. AKSHI, Kumar. 2018. *Machine Learning from Theory to Algorithms: An Overview* [online]. IOP Publishing, Bristol [Citirano 1. mar. 2021; 19:45]. Dostopno na spletnem naslovu: <https://doi.org/10.1088/1742-6596/1142/1/012012>.

[2] CANSIZ, Sergen. 2020. The Math Behind Training of a Neural Network [online].[Citirano 1. mar. 2021; 19:46]. Dostopno na spletnem naslovu: <https://towardsdatascience.com/adventure-of-the-neurons-theory-behind-the-neural-networks-5d19c594ca16>.

[3] GURNEY, Kevin. 2004. An introduction to neural networks [online]. Taylor & Francis e-Library, London. [Citirano 1. mar. 2021; 19:46]. Dostopno na spletnem naslovu: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjepc7qg4rvAhUY7qQKHfd7AsYQFjACegQIAhAD&url=https%3A%2F%2Fwww.inf.ed.ac.uk%2Fteaching%2Fcourses%2Fnlu%2Fassets%2Freading%2FGurney_et_al.pdf&usg=AOvVaw2jooKWeAUL1IzU5kyM4RMZ>.

[4] SIDDHARTH, Sharma. 2020. The Ultimate Beginner's Guide to Reinforcement Learning [online]. [Citirano 1. mar. 2021; 19:47]. Dostopno na spletnem naslovu: <https://towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec>.

[5] SUTTON, Richard S., in BARTO, Andrew G. 2018. Reinforcement Learning, second edition: An introduction. Cambridge, Massachusetts, ZDA: MIT Press. ISBN 9780262352703.

[6] SethBling (YouTube uporabniško ime, ime neznano). 2015. MarI/O - Machine Learning for Video Games [online]. [Citirano 3. mar. 2021; 19:20]. Dostopno na spletnem naslovu: <https://www.youtube.com/watch?v=qv6UVOQ0F44>.