Nikola Jovanović

Autonomno računarsko igranje igara inspirisano *Deep Learning* principima

U ovom radu demonstrirana je mogućnost korišćenja jednostavne metode mašinskog učenja za igranje jednostavnih igara. U tu svrhu implementirane su igre iks-oks i igra trkanja sa konzole Brick Game, kao primeri logičke i arkadne igre. Deep learning principi podrazumevaju univerzalnost algoritama, odnosno njihovu sposobnost da generalno prepoznaju zakonitosti u podacima. U kontekstu ovog rada to znači da algoritmi ne zavise od odabira konkretne igre, niti poznaju pravila i ciljeve igre. Na raspolaganju su informacije o trenutnom stanju ekrana (niz piksela), kao i mogućnost izvršenja nekakvih akcija, ali ne i informacije o povezanosti tih akcija sa samom igrom. Igre su tako dizajnirane da boduju uspešnost algoritma i da tu informaciju prosleđuju zajedno sa trenutnim stanjem ekrana. Algoritmi koji su dizajnirani da demonstriraju objašnjeni princip su se dobro pokazali u učenju igara na kojima su testirani, čemu je doprinela i mala složenost igara. Za složenije igre potrebno je koristiti složenije metode, jer korišćeni algoritmi ne mogu biti korišćeni za proaktivno igranje potrebno kod, na primer, „pucačkih" igrica.

Uvod

Ideja ovog rada je da demonstrira primenu dubokog mašinskog učenja (Ariel *et al*. 2005; Kaelbing *et al*. 1997), kao jednog od oblika veštačke inteligencije, koja je sve češće tema naučno-popularnih članaka i jedna je od oblasti računarstva koje se trenutno najviše razvijaju. Veliku primenu ima u učenju računara da igra video-igre, a najnaprednija rešenja sposobna su da nadmaše ljudskog eksperta (Mnih *et al*. 2015) u velikom broju igara (Tesauro 1994). Inspiracija za ovaj projekat je rad „Playing Atari with Deep Reinforcement Learning" (Mnih *et al*. 2013), čiji su autori uspešno dizajnirali algoritam koji je u stanju da sa velikim uspehom nauči da igra nekoliko Atari igara na kojima je testiran.

Ovaj rad prikazuje dva algoritma sa različitim pristupima i njihovu primenu na igranje konkretnih igara. Jedan algoritam je demonstriran na

Nikola Jovanović (1996) Novi Beograd, Antifašističke borbe 2/14, učenik 4. razreda Devete gimnazije „Mihailo Petrović Alas"

MENTOR:

Dragan Toroman, Istraživačka stanica Petnica

društvenoj igri iks-oks, a drugi na dve verzije igre trkanja (Racing) sa konzole Brick Game. Osnovna ideja je da program koji uči da igra igru nema informacije o funkcionisanju konkretne igre. Program raspolaže trenutnim stanjem na ekranu. Ono je dato u obliku matrice piksela, ali u opštem slučaju može predstavljati bilo kakav niz vrednosti, koje program bez prethodnog iskustva ne može da interpretira. Važno je razumeti da, kada program koji uči da igra igru trkanja dobije sliku ekrana na kojoj se nalaze igračev, kao i protivnički automobili, on na njoj ne prepoznaje nikakve automobile, već celu sliku posmatra isključivo kao niz brojeva. Druga informacija kojom raspolaže je signal da je upravo dobio ili izgubio poen ili život. Takođe, ima informaciju o tome koje komande su mu trenutno na raspolaganju (koji tasteri na tastaturi mogu da utiču na stanje igre). Kombinovanjem ovih signala i stanja ekrana, program počinje da uči u kojim stanjima ekrana mora odigrati koji potez kako bi osvojio poene ili sačuvao život.

Opis korišćenih igara i algoritama

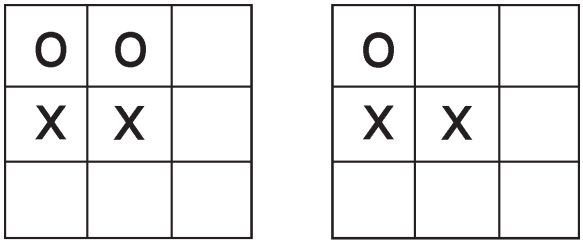
Oba korišćena algoritma su zasnovana na istoj ideji. U svakoj iteraciji (iks-oks) ili frejmu igre (trkanje) algoritmi dobijaju sliku ekrana, niz dostupnih komandi i informaciju o eventualnoj nagradi dodeljenoj na prelasku iz prethodnog stanja u trenutno. Umesto niza dostupnih komandi, mogla je biti korišćena cela tastatura, pri čemu veliki broj tastera ne bi imao nikakvu funkciju, što bi povećalo vreme učenja, a ne pravi suštinsku razliku. Stanja ekrana se ne koriste u obliku matrice, već se odmah hešuju i dalje se koriste samo heš vrednosti. Hešovanje predstavlja izračunavanje brojne vrednosti na osnovu neke kompleksnije strukture podataka. Ono poboljšava memorijsku efikasnost i ubrzava upoređivanje podataka. Za hešovanje je korišćen algoritam md5 (Rivest 1992). Kako se nagrada koja se dobija jednim potezom dodeljuje tek u sledećem, uvek se pamti heš vrednost stanja ekrana prethodnog frejma. Kada stigne informacija da je nagrada upravo dodeljena, prvo se proverava da li se u listi bitnih stanja već nalazi ono koje je prethodilo dodeli nagrade. Ukoliko nije, u heš tabelu koja predstavlja listu bitnih stanja se dodaje novi red, a za ključ se koristi heš vrednost prethodnog stanja ekrana. U heš tabelu se dodaje lista onoliko brojeva koliko ima poteza na raspolaganju i oni se postavljaju na inicijalnu vrednost. Ukoliko je povratna vrednost pozitivna, broj koji predstavlja potez koji je doveo do nagrade se uvećava, a ostali se umanjuju. U suprotnom, broj koji predstavlja izvršen potez se smanjuje, a ostali se povećavaju. Transformacije na ovoj listi se vrše tako da se promena verovatnoće odabiranja jedne akcije ravnomerno menja u odnosu na verovatnoće svih ostalih akcija. Ukoliko neki broj u listi padne ispod određene vrednosti, on se postavlja na nulu i više se ne uzima u razmatranje (neće više biti povećavan).

Nakon procesiranja povratne informacije, potrebno je odrediti akciju koja će biti izvršena u tekućem potezu. Isprobano je odabiranje akcija, takvo da akcija koja je u prošlosti bila uspešnija ima veću verovatnoću da

bude odabrana, ali jednako efikasno se pokazalo i potpuno nasumično odabiranje akcija od svih dostupnih, čije težine nisu jednake nuli. Iskorišćeno je potpuno nasumično odabiranje zbog mogućnosti da u suprotnom slučaju program slučajno zanemari akciju koja bi mogla da bude veoma dobar izbor. Bitna karakteristika ovog algoritma je da, počevši od nule (prazne heš tabele), program partiju za partijom unapređuje svoju igru, tj. nije potrebno davati programu „trening setove", već on sam uči od nule. Takođe, izbegnuta je potreba da se programu prvo da da uči, a zatim da pokaže najbolje što zna, zato što on učeći već igra najbolje što ume.

Igra iks-oks je implementirana sa standardnim pravilima. Ekran koji se prosleđuje algoritmu je predstavljen matricom sa tri reda i tri kolone, sa mogućim stanjima 0, 1 i 2. Ova stanja predstavljaju prazno polje, polje obeleženo sa X i polje obeleženo sa O. Komande u igri su interpretirane kao pritisci na devet različitih tastera na tastaturi, tako da se algoritam u svakom potezu odlučuje za akciju od 0 do 8. Pošto nesprovođenje akcije (propuštanje poteza) u ovoj igri znači samo dalje čekanje na komandu, zato algoritam nema tu mogućnost.

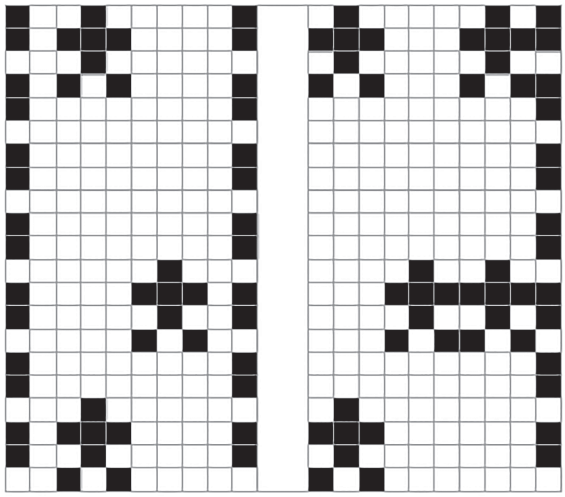
U implementaciju igre je uključen sistem bodovanja table. To znači da je za svako stanje na tabli moguće izračunati koji potez je optimalan. Tako da, ukoliko igrač koji je na potezu ima već dva svoja znaka u redu i mogućnost da završi partiju, taj potez se boduje kao najefikasniji (slika 1a). Ukoliko to nije slučaj, kao najefikasniji potez se boduje onaj u kome igrač blokira protivnikov niz od dva znaka (slika 1b). Prethodno opisani sistem bodovanja se vrši od strane same igre. Treba podsetiti da program povezuje dobijene bodove sa heš vrednostima matrica stanja igre (prikaz na ekranu), bez razumevanja koncepta igre. Kada algoritam izabere akciju i potez se odigra, povratna vrednost koja se vraća algoritmu predstavlja vrednost efikasnosti poteza koji je odigran. Ona ima vrednost 1 ukoliko je izabran optimalan potez, u suprotnom ima vrednost 0. Iz ovog primera se može videti da se algoritmi dizajnirani za učenje arkadnih igara takođe mogu primeniti na logičke igre, kao što je iks-oks.



Slika 1. a) Pošto O nije blokirao niz od dva X, X će dobiti nagradu ukoliko završi partiju; b) Pošto X ima niz od dva, a O nema mogućnost da završi partiju, O će dobiti nagradu ukoliko blokira niz od dva X

Figure 1. a) O did not block X, so X is going to get an award, if it finishes the game; b) X has two fields in a row and O does not have the possibility of winning the game, so O is going to get an award if it blocks X

Dve verzije **igre trkanja** su preuzete sa konzole Brick Game (slika 2) Automobil koji kontroliše igrač se nalazi na dnu ekrana, kontroliše se sa dva tastera koji označavaju komande levo i desno, tako da u prvom slučaju automobil može biti u dva, a u drugom u tri stanja. Protivnički automobili



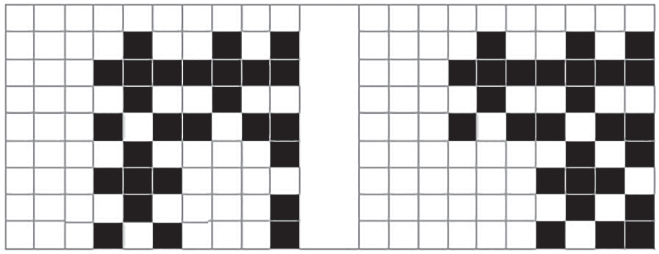
Slika 2.
Staza za trkanje sa dve trake (levo) i tri trake (desno)

Figure 2.
Two-lane racetrack (left) and tree-lane racetrack (right)

se pojavljuju na vrhu ekrana u nasumično odabranim trakama i kreću se na dole, cilj igre je ne sudariti se sa protivničkim automobilima.

Ekran koji se prosleđuje algoritmu je predstavljen matricom sa 21 redom i 10 kolona, sa mogućim stanjima 0 i 1. Ona predstavljaju stanje crno-belog piksela ekrana. Za kontrolu automobila se koriste dva tastera na tastaturi, a algoritam se u svakom prolazu odlučuje za jednu od te dve akcije ili za neizvršavanje nikakve komande. Sistem nagrađivanja u ovoj igri je takav da algoritam samo dobija negativne poene kada se njegov automobil sudari sa drugim i izgubi život.

Algoritam korišćen za učenje ove igre razlikuje se od prethodnog u tome što, kada dobije informaciju da je izgubio život, postavi efikasnost akcije koja je dovela do gubitka života za prethodno stanje ekrana na nulu. Takav pristup je dao odlične rezultate za prvu verziju igre. Međutim, u drugoj verziji igre postoje stanja u kojima se ne gubi život, ali dolazak u ta stanja dovodi do neizbežnog gubitka života (slika 3). Ovaj problem je rešen tako što će, nakon dovoljno igara, algoritam doći u problematično stanje dva puta i u tom stanju će ostati samo jedna akcija koja nije zabranjena. Kada sledeći put bude došao u isto stanje, izabrao jedinu preostalu akciju i



Slika 3. Stanje u kome je moguće izbeći gubitak života prestrojavanjem u levo (levo) i stanje u kome život još uvek nije izgubljen, ali je to nemoguće izbeći (desno)

Figure 3. State in which the player can avoid collision by moving to the left (left) and state in which a collision did not happen, but it is unavoidable (right)

izgubio život, algoritam će to sâmo stanje zapamtiti kao stanje u kome se gubi život, iako to stanje samo neminovno dovodi do gubitka života. Nakon ovog poteza, akcije koje ga dovode u ovo stanje biće okarakterisane isto kao akcije koje direktno dovode do gubitka života. Ovakav sistem podrazumevanja bezizlaznih stanja kao stanja u kojima se gubi život omogućava grananje tih stanja, tj. moguće je da program zaključi da iz jednog stanja može da dođe samo u više bezizlaznih stanja, tako da i to stanje dobija isti atribut.

Testiranje algoritama. Nakon implementacije igara i algoritama, pokrenuti su testovi (treninzi). Oni su se sastojali iz velikog broja odigranih partija. Omogućeno je treniranje u realnom vremenu, koje podrazumeva da se na ekranu prikazuje trenutno stanje igre, tako da se direktno može pratiti uspešnost računara, kao i mogućnost treniranja u pozadini, koje pri izvršavanju ne prikazuje trenutno stanje, što mu omogućava da višestruko ubrza igru i završi treniranje u mnogo kraćem roku.

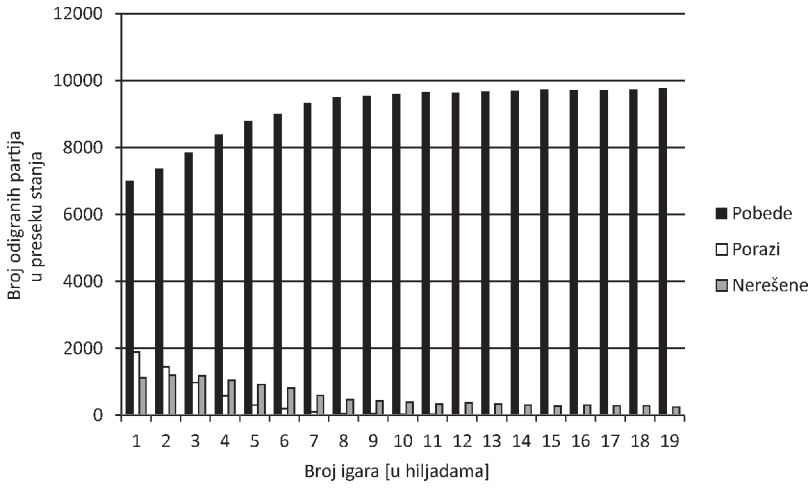
Algoritam za igru iks-oks je treniran u partijama sa protivnikom koji od raspoloživih potpuno nasumično odabira svoje poteze. Odvojeno su vršena treniranja za X i O poziciju.

Rezultati

Korišćeni algoritmi su pokazali veliku uspešnost u savladavanju igara na kojima su testirani, uzimajući u obzir malu složenost igara, ali i uprošćenost algoritama.

Rezultati učenja igre iks-oks

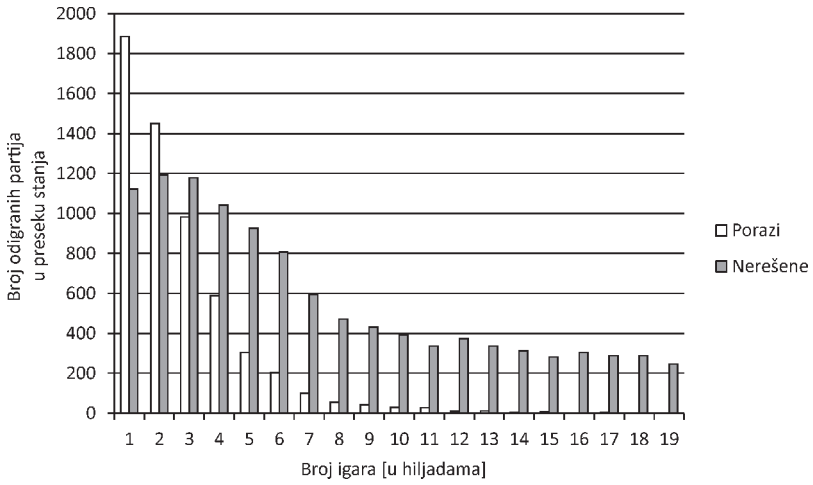
Grafici na slikama 4 i 5 prikazuju rezultate učenja pozicije X, a grafici na slikama 6 i 7 rezultate učenja pozicije O. Rezultati merenja su uzimani na svakih 10000 odigranih partija, a brojači pobeda, poraza i nerešenih partija su resetovani.



Slika 4.
Napredovanje u učenju pozicije X

Figure 4.
Advancement in learning the X side

Na slici 4 se može videti da je broj pobeda u samom početku veći od broja poraza, što se objašnjava time da tada algoritam igra potpuno nasumično, a pozicija X je favorizovana igranjem prvog poteza. Sa grafika se vidi da već na 110 hiljada partija broj pobeda dostiže maksimum, ali ne može se smatrati da je program naučio da igra, sve dok broj izgubljenih partija ne padne na nulu. Mali procenat nerešenih partija je prihvatljiv, zato što će protivnik koji igra nasumično u nekom broju partija slučajno sprečiti pobedu.



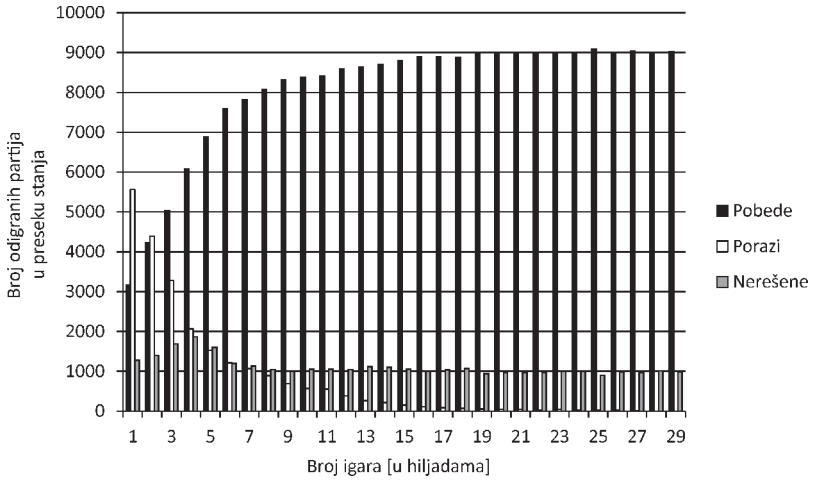
Slika 5. Napredovanje u učenju pozicije X, bez prikazanih pobeda

Figure 5. Advancement in learning the X side, without bars representing victories

Na slici 5 se preciznije vidi odnos izgubljenih i nerešenih partija. Uočava se mali skok u broju nerešenih partija u trenutku najvećeg pada broja izgubljenih partija, što se objašnjava time da je prioritet algoritma smanjenje broja izgubljenih partija, a zatim pretvaranja nerešenih partija u pobede. Takođe se uočava da broj izgubljenih partija na početku najbrže opada, a zatim sve sporije. Nulu dostiže tek na intervalu nakon 170 hiljada partija.

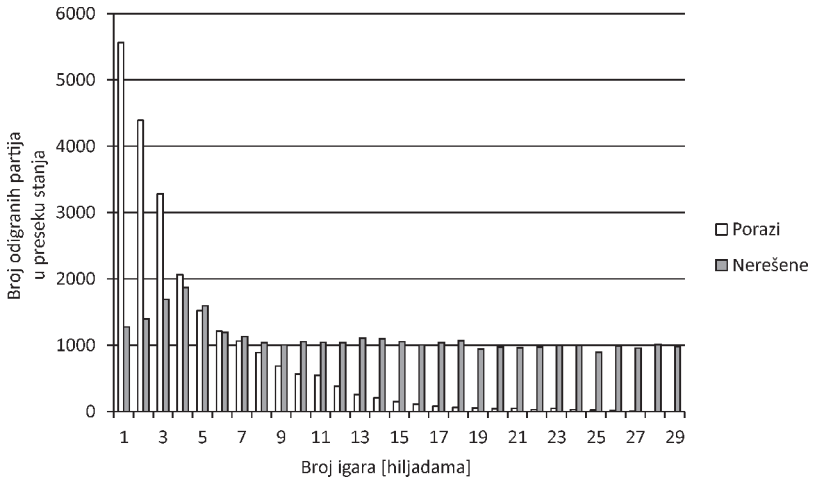
Sa grafika na slici 6 se vidi da je broj partija potreban da se savlada pozicija O dosta veći – 270 hiljada partija, naspram 170 hiljada potrebnih za savladavanje pozicije X. Iako bi ovi brojevi mogli da budu smanjeni optimizovanjem algoritma, njihov odnos bi ostao isti. Druga činjenica koja se uočava na grafiku je da je na početku broj izgubljenih partija veći od broja dobijenih. Razlog za to je isti kao i u prethodnom slučaju, samo što se program sada nalazi u podređenoj O poziciji. Kao u prethodnom slučaju, broj pobeda raste do intervala oko 190 000 partije, što je isti segment treninga u odnosu na ukupan broj partija, ali i sada je potrebno potpuno onemogućiti poraz. Procenat nerešenih partija je u ovom slučaju veći zato što je sada protivnik, koji igra nasumično, u favorizovanoj poziciji X.

Na slici 7 se vidi sličan trend kao i na slici 5. Broj nerešenih partija u početku skače, a zatim opet stagnira, dok broj izgubljenih partija teži nuli koju dostiže nakon 270 hiljada partija.



Slika 6.
Napredovanje u učenju pozicije O

Figure 6.
Advancement in learning the O side



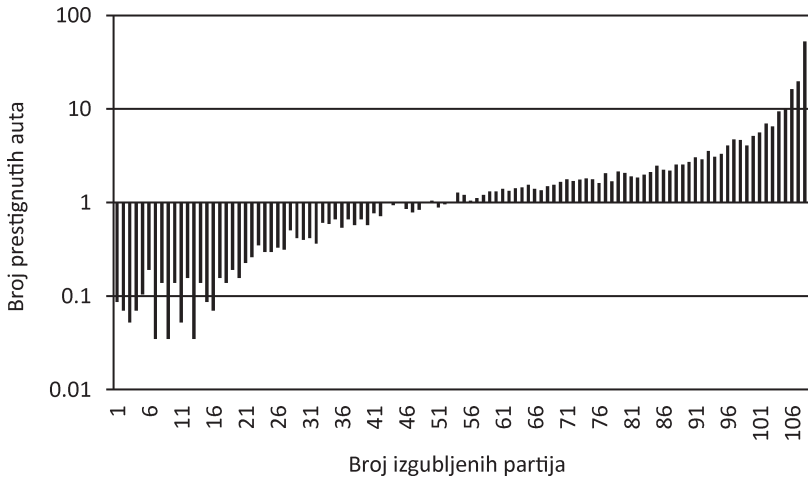
Slika 7.
Napredovanje u učenju pozicije O, bez prikazanih pobeda

Figure 7.
Advancement in learning the O side, without bars representing victories

Rezultati učenja dve verzije igre trkanja

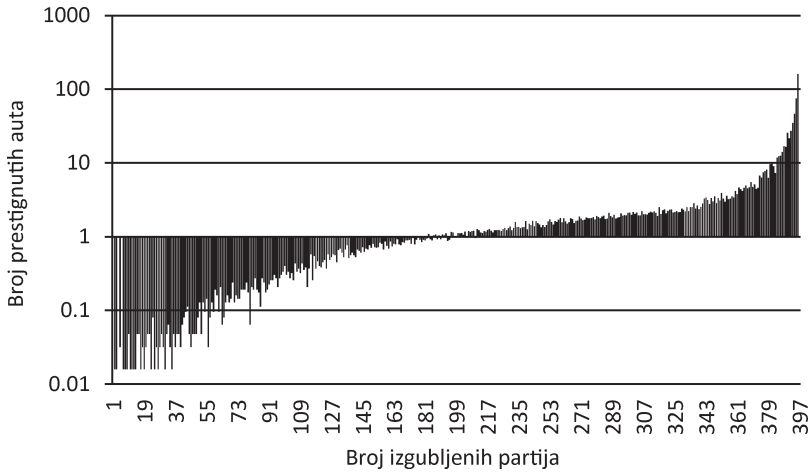
Verzija igre trkanja sa dve trake savladana je nakon 108 izgubljenih partija, a broj stanja koja mogu da dovedu do gubitka života, tako da su morala da budu sačuvana, 90 je. Razlika između ova dva broja postoji zato što u nekim stanjima do gubitka života može da dovede više različitih akcija. Verzija igre sa tri trake savladana je nakon 396 izgubljenih partija, a broj stanja koja su sačuvana iznosi 261. Važno je naglasiti da algoritam koji je učio prvu verziju, ne bi mogao da nauči i drugu, zbog veće složenosti, dok napredniji algoritam može da nauči obe.

Grafici na slikama 8 i 9 prikazuju odnos broja prestignutih protivnika nakon različitog broja izgubljenih života u verziji igre sa dve i tri trake. Prikazan je prosečan broj prestignutih protivnika, nakon velikog broja



Slika 8.
Napredovanje pri učenju igre trkanja sa dve trake

Figure 8.
Advancement in learning the racing game, with a two-lane racetrack



Slika 9.
Napredovanje pri učenju igre trkanja sa tri trake

Figure 9.
Advancement in learning the racing game, with a three-lane racetrack

ponovljenih učenja. Primećuje se da u slučaju ove igre nije moguće očekivati veliku uspešnost, dok igra nije potpuno naučena.

Zaključak

Testirani algoritmi su uspešno naučili odabrane igre. Ipak, mora se uzeti u obzir da odabrane igre nisu bile velike složenosti. Važno je imati na umu da ovi algoritmi vide igru kao „black box", tj. nemaju nikakav uvid u funkcionisanje ili pravila igre, a informacije koje dobijaju same po sebi ne znače ništa.

Pomenuti Deep Reinforcement Learning pristup je mnogo kompleksniji i napredniji od ovde opisanih algoritama. Jedna od mana ovde korišćenog pristupa je to što je potrebno zapamtiti u memoriji svako stanje za koje je potrebno izabrati određenu akciju umesto neke druge. To može predstavljati memorijski problem, ali još bitnije, ne postoji mogućnost

generalnog prepoznavanja situacija koje zahtevaju određenu akciju. Ukoliko je samo jedan piksel na ekranu različit, heš funkcija daje potpuno različitu vrednost stanja. A taj jedan piksel može biti nebitan estetski detalj (ivica puta u igri trkanja) za odabir akcije, ali može biti i loptica u igri Brick breaker.

Kako bi ovde opisane algoritme unapredili tako da prevaziđu navedene probleme, mogu se koristiti metode analize signala. Ulaz je takođe matrica ekrana i algoritam i dalje nema nikakvih informacija o funkcionisanju same igre. Nad matricom se vrši evaluacija niza koeficijenata i dobijene vrednosti se čuvaju umesto heš vrednosti matrice. Vremenom algoritam uči koja od vrednosti koje se računaju svojom promenom može da predvidi osvajanje poena ili gubljenje života. Bitna prednost ovog pristupa je to što se vrednosti evaluirane nad trenutnom matricom ne moraju u potpunosti slagati sa vrednostima koje predstavljaju neko prethodno stanje u kom je izgubljen život, kako bi bila preduzeta odgovarajuća akcija.

Literatura

Arel I., Rose D. C., Karnowski T. P. 2010. Deep Machine Learning – A New Frontier in Artificial Intelligence Research. *IEEE Computational Intelligence Magazine*, **5** (4): 13.

Kaelbing L. P., Mittman M. L., Moore A. W. 1996. Reinforcement learning: A Survey. *Journal of Artificial Intelligence Research*, **4**: 237.

Mnih V., Kavukcuoglu K., Silver D., Rusu A. A., Veness J., Bellemare M. G., *et al*. 2015. Human-level control through deep reinforcement learning. *Nature*, **518**: 1529.

Tesauro G. 1994. TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, **6** (2): 215.

Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. 2013. Playing Atari with Deep Reinforcement Learning. NIPS (Neural Information Processing Systems) Deep Learning Workshop 2013. Dostupno na: https://arxiv.org/pdf/1312.5602v1.pdf

Rivest R. 1992. The MD5 Message-Digest Algorithm. Internet Engineering Task Force. Dostupno na: https://tools.ietf.org/html/rfc1321

Nikola Jovanović

Autonomous Computer Game Playing Inspired by "Deep Learning" Principles

This paper demonstrates how we can use a simple machine learning method for playing simple computer games. For that purpose, we implemented a Tic-Tac-Toe game and a racing game from a Brick Game console, as examples of logic and arcade games. In this paper we used universal algorithms which neither depend on the specific chosen game, nor do they know anything about the rules or the goals of the game. Algorithms had information about the current state of the screen (matrix of pixels) and they were able to perform certain actions. Those actions could not be directly correlated to the actual intended gameplay. The games were designed so that they calculate the algorithm's efficiency and send that information back to the algorithm along with the current screen data. The Tic-Tac-Toe algorithm was designed to play against an opponent playing completely random moves, in order to have a chance of winning. If a game was successful, the algorithm would increase the probability of repeating the same strategy, while in the opposite case, the same probability would decrease. The racing game algorithm was based only on avoiding losing a life. The described algorithms successfully learned to play the games they were tested on, but one must keep in mind that the games were simple, as were the algorithms. More complex games require more complex methods, so the algorithms used here would not be able to play "shooter" games.