Nikola Jovanović
Novi Beograd, Antifašističke borbe 2/14
1996. godište, IV razred, Deveta gimnazija „Mihailo Petrović – Alas“

Autonomno računarsko igranje igara inspirisano "Deep Learning" principima

U ovom radu demonstrirana je mogućnost korišćenja jednostavne metode mašinskog učenja za igranje jednostavnih igrica. U tu svrhu implementirane su igre Iks-oks i igra trkanja sa konzole Brick Game, kao primeri logičke i arkadne igre. Algoritmi obrađeni u ovom radu su univerzalni, odnosno ne zavise od odabira konkretne igre, niti poznaju pravila i ciljeve igre. Algoritmi na raspolaganju imaju 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 pokazli 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 "pucačkih" igrica.

1. Uvod

Ideja ovog rada je da čitaocima približi koncept mašinskog učenja, 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 igrice, a najnaprednija rešenja sposobna su da nadmaše ljudskog eksperta u velikom broju igrara. Inspiracija za ovaj prijekat je rad *„Playing Atari with Deep Reinforcement Learning“* (1), 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. 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 piskela, 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. Jedan algoritam je demonstriran na društvenoj igri Iks-oks, a drugi na dve verzije igrice trkanja (*Racing*) sa konzole *Brick Game.*

2. Opis korišćenih igara i algoitama

Oba korišćena algoritma su u osnovi ista. U svakoji 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 komadi, 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 predstvlja izrčunavanje brojne vrednosti na osnavu neke kompleksnije strukture podataka, poboljšava memorijsku efikasnost i ubrzava upoređivanje podataka. Za hešovanje je korišćen algoritam *md5*. 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 ekana. U heš tabelu se dodaje lista onoliko brojeva koliko ima poteza na raspolaganju i oni se postavljaju na inicijalnu vrednost. Ukoliko je nagrada pozitivna, broj koji predstavlja potez koji je odveo do nagrade se uvećava, a ostali se umanjuju. Ukoliko je nagrada negativna broj koji predstavlja izvršen potez se smanjuje, a ostali se povećavaju. Trasformacije 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 nagrade, potrebno je odrediti akciju koja će biti izvršena u ovom potezu. Isprobano je težinsko nasumično odabiranje akcija u odnosu na broj koji predstavlja tu akciju u listi u heš tabeli za trenuitno stanje ekrana, ali jednako efikasno se pokazalo i potpuno nasumično odabiranje akcija od svih dostupnih, čije težine nisu jednake nuli. Drugi pristup je odabran zbog mogućnosti da u prvom slučaju program sličajno izignoriše 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 postaje sve bolji i bolj, 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 ne sprovođenje nkakve akcije (propuštanje poteza) u ovoj igir znači samo dalje čekanje na komandu, u ovom slučaju aligoritam nema tu mogućnost.

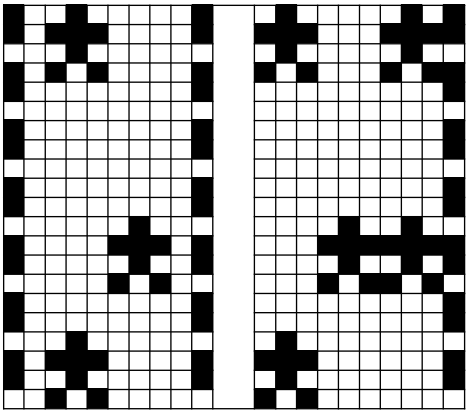
U implementaciju igre je uključen sistem bodovanja table. To znači da je za svako stanje na tabli moguće izračunati koji potez je optimalan. Tako da ukoliko igrač koji je na potezu ima već dva svoja znaka u redu i mogućnost da završi partiju, taj potez se boduje kao najefikasniji (Slika 1a). Ukoliko to nije slučaj kao najefikasnioji potez se boduje onaj u kome igrač blokira protivnikov niz od dva njegova znaka (Slika 1b). Prethodno opisani sistem bodovanja se vrši od strane same igre. Treba podsetiti da progam povezuje dobijene bodove sa heš vrednosti ekrana, bez razumevanja koncepta igre. Kada algoritam izabere akciju i potez se odrigra, nagrada 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 igra, kao što je Iks-oks.

O	O			O		
X	X			X	X	

Slika 1a
Pošto O nije blokirao niz od dva X, X će dobiti nagradu ukoliko završi partiju.

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

Dve igre trkanja su preuzete sa konzole *Brick Game* [1]. (Slika 2a i 2b) 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 se pojavljuju na vrhu ekrana u nasumično odabranim trakama i kreću se na dole, cilj igre je ne sudariti se sa protivničkim automobilima.

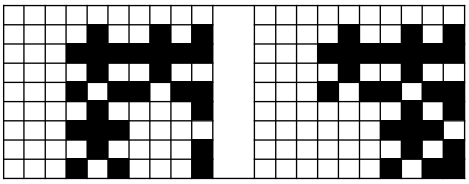


Slika 2a
Staza za trkanje sa dve trake.

Slika 2b
Staza za trkanje sa tri trake.

Ekran koji se prosleđuje algoritmu je predstavljen matricom sa 20 redova 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 svakm prolazu odlučuje za jednu od te dve akcije, ili za neizvršavanje nikakve komande. Sistem nagrađivanja u ovoj igrici 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 izgubio život, algoritam će to samo stanje preinačiti u stanje u kome se gubi život, iako to stanje samo neminovno dovodi do toga. Nakon ovog poteza akcije koje ga dovode u ovo stanje biće karakterisane 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.



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

3. Testiranje algoritama

Nakon implementacije igara i algoritama, pokrenuti su testovi (treninzi). Oni su se sastojali iz velikog broja odigranih partija.

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

4. Rezulatati

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

Grafici 1 i 2 prikazuju rezultate učenja pozicije X, a grafici 3 i 4 rezultate učenja pozicije O. Rezultati merenja su uzimani na svakih 10 000 partija, a brojači pobeda, poraza i nerešenih partija su resetovani.

Na Grafiku 1 se mže 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 110 000 partija broj pobeda dostiže maskimium, 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.

Na Grafiku 2 se preciznije vidi odnos izgubljenih i nerešenih partija. Uočava se mali skok u broj nerešenih partija u trenutku najvećeg pada broja izgubljenih partija, što se objašnjava time da je prioritet algoritma smanjenje boja 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 170 000. partije.

Sa Grafika 3 se vidi da je broj partija potreban da se savlada pozicija O, dosta veći, 270 000 partija naspram 170 000 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 3 je da je na početku broj izgubljenih partija veći od broja dobijenih. Razlog za to je isti kao i za prethodni slučaj, 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 nerešenihg partija je u ovom slučaju veći zato što je sada protivnik, koji igra nasumično, u favorizovanoj poziciji X.

Na Grafiku 4 se vidi sličan trend kao i na Grafiku 2. Broj nerešenih partija u početku skače, a zatim opet stagnira, dok broj izgubljenih partija teži nuli koju dostiže nakon 270 000 partija.

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 je 90. Razlika između ova dva broja potoji zato što u nekim stanjim 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 je 261. Na graficima 5 i 6 se vidi odnos osvojenih poena nakon različitog broja izgubljenih života u verziji igre sa dve i tri trake. Prikazane vrednosti su srednje vrednosti osvojenih poena nakon velikog broja ponovljenih učenja. Primećuje se da u slučaju ove igre nije moguće očekivati bilo kakve rezultate dok igra nije potpuno naučena.

5. Zaključak

Testirani algoritmi su se odlično pokazali na odabranim igrama. Ipak mora se uzeti u obzir da odabrane igre nisu bile velike složenisti. Važno je imati na umu da ovi algoritmi vide igru kao „*black box*“, tj. nemaju nikakav uvid u funkcionisanje ili pravila igara, a inforamcije koje dobijaju same po sebi ne znače ništa.

Pomenuti *Deep Reinforcement Learning* pristup je mnogo kompleksiji 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 genaralnog 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, za odabir akcije, nebitan estetski detalj (ivica puta u igri trkanja), ali može biti i loptica u igri *Brick breaker.*

Kako bi ovde opisane algoritme unapredili tako da prevaziđu navedene probleme, može se koristiti metoda analize signala. Ulaz je takođe matrica ekrana i algortiam i dalje nema nikakvih informacija o funkcionisanju same igre. Nad matricom se izvršava niz načina evaluacije i dobijene vrednosti se čuvaju umesto heš vrednosti. Vremenom algortam 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 kome je izgubljen život, kako bi bila preduzeta odgovarajuća akcija.

Literatura:

(1) Mnih V; Kavukcuoglu K; Silver D; Graves A; Antonoglou I; Wierstra D, Riedmiller M (2013). "*Playing Atari with Deep Reinforcement Learning*".

Autonomus computer game playing inspired by "Deep Learning" principles

This paper demonstrates how we can use simple machine learning method for playing simple computer games. For that purpose we implemented Tic-Tac-Toe game and racing game from 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 they know anything about rules or goals of the game. Algorithms have information about current state of the screen (matrix of pixels) and they are able to perform certain actions. Those actions can not be directly correlated to the actual intended gameplay. Games are

designed so that they are calculating algorithm's efficiency and sending that information back to the algorithm along with the current screen data. 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, algorithm would increase the probability of repeating the same strategy, while in the opposite case, the same probability would decrease. Racing game algorithm is based only on avoiding loosing a life. Described algorithms successfully learned to play games they were tested on, keeping in mind that games were simple, as well as the algorithms. More complex games require more complex methods, so algorithms used here are not able to play "shooter" games.