Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.85
DOI: https://doi.org/10.24867/01BE47Radosavljevic

PRIMENA UČENJA USLOVLJAVANJEM NA OBUČAVANJE AGENTA ZA IGRANJE VIDEO IGRE ROAD FIGHTER

TRAINING A REINFORCEMENT LEARNING AGENT TO PLAY ROAD FIGHTER

Ivan Radosavljević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu dat je primer primene učenja uslovljavanjem agenta za igranje video igre Road Fighter. Agent je implementiran kao Duboka Q-mreža i obučavan metodom Dubokog Q-Učenja. Obučavanje je vršeno samo na osnovu slika ekrana video igre. Rezultati postignuti nakon 10000 epizoda obučavanja bili su slični rezultatima postignutim u srodnim radovima pod uslovom sličnih hardverskih ograničenja.*

Ključne reči: *Učenje uslovaljavanjem, neuronske mreže, duboka q mreža, duboko q učenje, video igra*

Abstract – *This paper presents the application of reinforcement learning for the purpose of training an agent capable of playing the video game Road Fighter. The agent is implemented as a Deep Q-Network and is trained via the Deep Q-Learning algorithm. The only data used for training the agent were screenshots of the game. After 10000 episodes of training the performance of the agent was comparable to that achieved in related approaches under similar hardware constraints.*

Keywords: *Reinforcement learning, neural networks, deep q network, deep q learning, video game*

1. UVOD

Metode mašinskog učenja koje spadaju u metode nadgledanog i nenadgledanog učenja su se pokazale kao veoma dobre u rešavanju problema klasifikacije i otkrivanja šablona u podacima, međutim uspešnost ovih metoda najčešće zavisi od toga koliko su dobro pripremljeni podaci nad kojima se primenjuju. Priprema podataka je najčešće mukotrpan proces koji vrše ljudski eksperti i podrazumeva uklanjanje šuma, rešavanje problema nedostajućih vrednosti, u slučaju nadgledanog učenja obeležavanje podataka, itd. Za razliku od ovih metoda, učenje uslovljavanjem, se ne oslanja na prethodno pripremljene podatke, umesto toga podaci za obučavanje se sakupljaju direktno u toku procesa obučavanja agenta za rešavanje problema u nekom okruženju. Ovo čini učenje uslovljavanjem veoma primamljivim za rešavanje problema za koje nije praktično vršiti ručnu pripremu podataka. Jedan takav problem predstavlja obučavanje agenata za igranje igara, jer dovoljno složene igre mogu generisati velike količine podataka koje je u većini slučajevima nemoguće ručno pripremiti.

Iz tog razloga je agent koji će biti predstavljen u ovom radu realizovan metodom učenja uslovljavanjem, predstavljenoj u radu [1], prilagođenoj za obučavanje agenata za igranje video igara. Agent je obučavan za igranje video igre Road Fighter koja predstavlja jednostavnu simulaciju trke izdatu za Nintendo Entertainment System konzolu. Cilj ovog rada bio je da se metoda iz rada [1] isproba na prosečnom hardveru i nešto složenijoj video igri u odnosu na video igre za Atari konzolu. Po uzoru na rad [1] obučavanje je vršeno samo na osnovu slike ekrana video igre, bez dodatnih podataka. bilo internih ili dostavljenih od strane ljudskog eksperta. Glavna razlika između rešenja predstavljenog u ovom radu i rešenja iz rada [1] jeste to što se u ovom rešenju za obučavanje agenta koristilo dve neuronske mreže, a ne jedna. Takođe za razliku od rada [1] u kojem je dobavljanje slika ekrana video igre i slanje akcija za upravljanje igrom vršeno direktno preko Atari emulatora, a u ovom radu je to izvedeno indirektno preko odvojene softverske komponente napravljene sepcijalno za tu svrhu. Rezultati postignuti ovakvim pristupm bili su u okvirima očekivanih rezultata za primenu metode iz rada [1] pod ograničenjem upotrebe hardvera prosečnih računskih mogućnosti. Nakon 10000 epizoda obučavanja agent je bio sposoban da u proseku u video igri bez greške upravlja automobilom dve sekunde, dok ljudski igrač u proseku uspeva da bez greške igra video igru oko 14 sekundi.

Rad je podeljen u pet sekcija. U narednoj sekciji dat je kratak pregled postojećih rešenja. U trećoj sekciji predstavljena je primenjena metodologija. Četvrta sekcija opisuje postupak evaluacije rešenja i sadrži komentare na postignute rezultate. Konačno, u petoj sekciji, izložen je zaključak ovog rada.

2. POSTOJEĆA REŠENJA

Veliki broj radova na temu učenja uslovljavanjem tiče se primene ove metode mašinskog učenja na obučavanje agenata za igranje igara poput šaha [2], igre go [3] i sl., međutim tek u poslednje vreme su počeli da se pojavljuju radovi na temu obučavanja agenata za igranje video igara. Dva reprezentativna rada na ovu temu su [1, 4].

U radu [4] autori su predstavili implementaciju agenta za igranje video igre Dota 2. Agent je implementiran kao duboka neuronska mreža koja na osnovu podataka sa bot API-ja video igre donose odluke o akcijama koje treba da sprovede u igri. Podaci sa bot API-ja predstavljaju veoma složenu reprezentaciju stanja igre, sačinjenu od 20000 atributa. Agent se obučava igrajući partije igre protiv samog sebe. Nakon perioda od nekoliko meseci obuke

---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kovačević, vanredni prof.

agent je došao u stanje da može da pobedi igrače početnike, što je uzimajući u obzir složenost video igre za koju se obučava veliki uspeh.

Autori rada [1] predstavili su novu metdu za obučavanje agenata za video igre koja se ne oslanja na složene reprezentacije stanja. Metoda predstavljena u njihovom radu podrazumeva samo upotrebu slika ekrana video igre za obučavanje i eksploataciju agenta. Agent je implementiran kao duboka konvolutivna neuronska mreža koja na ulaz prima slike ekrana video igre a na izlaz daje aproksimiranu vrednost optimalne funkcije vrednosti stanja. Autori ovu mrežu nazivaju duboka Q mreža (eng. *Deep Q-Network*, DQN), dok metod obučavanje ove mreže nazivaju duboko q učenje (eng. *Deep Q Learning*). Primenom ove metode pri obučavanju agenata za igranje Atari video igara autori su uspeli da za neke video igre čak nadmaše performanse ljudskih eksperata. Ipak treba imati u vidu da su igre nad kojima je ova metoda primenjena uspešno dosta jednostavnije od Dota 2 video igre.

3. METODOLOGIJA

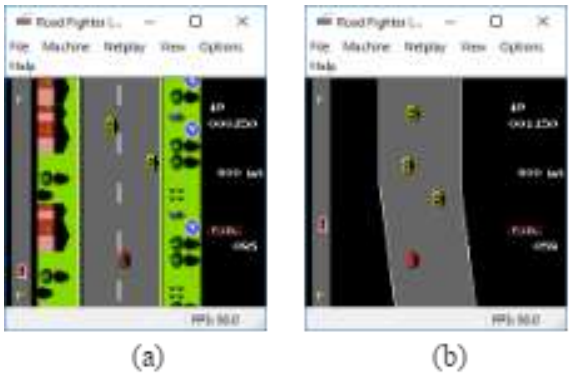
U ovoj sekciji biće predstavljeni detalji vezani za metodologiju primenjenu prilikom obučavanja agenta za igranje video igre Road Fighter. Prvo će biti opisana softverska komponenta napravljena za interakciju sa video igrom. Nakon toga će biti predstavljlena arhitektura i implementacija neuronske mreže upotrebljene za izvedbu agenta. Na samom kraju biće izneti detalji vezani za obuku agenta.

3.1. Upravljanje video igrom i dobavljanje podataka

Video igra Road Fighter izdata je 1994. godine kao video igra za Nintendo Entertainment System (NES) konzolu, i kao takva nije izvršiva na modernom računarskom hardveru. Kako bi se igra mogla pokrenuti neophodno je obezbediti emulator za NES konzolu. Za potrebe ovog rada, usled jednostavnosti upotrebe i preciznosti simulacije NES hardvera, odabran je Nestopia emulator [5]. Mana ovog emulatora ogleda se u nepostojanju ugrađenog interfejsa za povezivanje drugih programa sa emulatorom u cilju upravljanja emulatorom i dobavljanja podataka iz emulatora. Iz ovog razloga bilo je neophodno napraviti softversku komponentu koja će moći da obavlja ove zadatke. Ova komponenta implementirana je u programskom jeziku Python i sačinjena je od jedne klase pod nazivom GameControl. Klasa GameControl implementirana je uz oslonac na Windows API [6] i predstavlja apstrakciju za pristup prozoru video igre, odnosno emulatora. Upotreba Windows API-ja omogućuje pristup i upravljanje svim internim stanjima bilo kog prozora pod Windows operativnim sistemom. Ovo znači da se dobavljanje slike sadržaja prozora može vršiti veoma efikasno kopiranjem podataka iz grafičkog bafera prozora. Takođe slanje komandi u vidu pritisaka tastera tastature ili klikova miša se svodi na na pozive funkcija Windows API-ja za slanje poruka o događajima izazvanim upotrebom ulaznih uređaja. Za potrebe upravljanja emulatorom klasa GameControl implementira nekoliko metoda od kojih su najbitnije:

- *grab_screen()* - vrši dobavljanje slike sadržaja prozora emulatora, odnosno slike ekrana video igre.
- *get_current_state()* - vrši dobavljanje podataka o trenutnom stanju igre. Podaci se dobavljaju kao torka (slika ekrana, procenat pređene staze, preostala količina goriva, brzina). Pri tome slika ekrana predstavlja binarnu sliku samo dela ekrana igre na kojem su vidljivi samo staza za trkanje i automobili na stazi. Ova slika se dobija transformacijom slike dobijene pozivom metode grab_screen() u binarnu sliku i isecanjem dela koji sadrži stazu za trkanje. Poslednja tri podatka iz torke se dobavljaju izdvajanjem i interpretacijom delova slike dobijene metodom grab_screen() na kojima su prikazani ovi podaci.
- *execute_action(action_code, delay)* - izvršava zadatu akciju i blokira preuzimanje podataka ili slanje akcija u trajanju specificiranom argumentom delay. Blokiranje se vrši kako bi emulator imao vremena da obradi primljenu akciju. Dozvoljene akcije su skretanje u levo i skretanje u desno.
- *restart_game()* - vrši ponovno pokretanje igre sa početka staze.

Efikasno dobavljanje podataka o gorivu, brzini i poziciji sa slike ekrana video igre bilo je moguće zahvaljujući jednostavnom grafičkom dizajnu delova ekrana koji služe za prikaz tih podataka. Međutim izdvajanje dela ekrana na kojem se odvija sama trka i njegovo pretvaranje u binarnu sliku zahtevalo je izmenu same video igre u vidu uklanjanja mnogih ukrasnih elemenata koji se prikazuju na tom delu ekrana. Prikaz ekrana video igre pre i posle uklanjanja ovih elemenata dat je na slici 1.



Slika 1. *a) Izgled video igre pre uklanjanja ukrasnih elemenata, b) Izgled video igre nakon uklanjanja ukrasnih elemeanta*

3.2. Arhitektura i implementacija neuronske mreže

Cilj ovog rada bilo je obučavanje agenta koji bi na osnovu tri uzastopne slike ekrana video igre donosio odluke o akcijama koje treba da sprovede. Upotreba tri slike bila je neophodna kako bi se mogli izdvojiti podaci o brzinama kretanja objekata na ekranu. Iz ovog razloga odabrana je arhitektura koja predstavlja konvolutivnu neuronsku mrežu sačinjenu od dva 3d konvolutivna sloja i tri potpuno povezana sloja. Ulaz u neuronsku mrežu sačinjen je od tri binarne slike ekrana video igre pripremljena na način opisan u sekciji 3.1. Izlaz iz ove neuronske mreže je vektor koji sadrži dva elementa čije vrednosti ukazuju na povoljnost izbora akcija za skretanje levo ili desno. Prvi sloj u neuronskoj mreži je 3d konvolutivni sloj sa filtorm dimenzija 5x5x2 koji se pomera za po dva piksela po x i y osi i za po jedan po z osi. Izlaz iz ovog sloja predstavlja 16 feature mapa sačinjenih od po dve slike dimenzija duplo manjih od ulaznih slika. Nad ovim feature mapama

se potom primenjuje 3d batch normalizacija čiji se rezultat prosleđuje ReLU aktivacionoj funkciji. Sloj za 3d batch normalizaciju uveden je kako bi se postigla stabilnija konvergencija tokom obučavanja [7]. Ovako transformisane feature mape se zatim prosleđuju drugom 3d konvolutivnom sloju čiji filter ima dimenzije 3x3x2. Korak ovog filtera isti je kao i korak filtera u prvom konvolutivnom sloju.

Izlaz iz ovog sadrži 32 feature mape koje se sastoje od po jedne slike duplo manjih dimenzija od ulaznih feature mapa. Takođe i nad ovim izlazom se vrši 3d batch normalizacija i primenjuje ReLU aktivaciona funkcija.

Nakon toga dobijene feature mape se pretvaraju u vektor od 37696 elemenata koji se prosleđuje prvom potpuno povezanom sloju. Izlaz iz ovog sloja je vektor od 32 elementa. Ovaj vektor se prosleđujeu narednom potpuno povezanom sloju koji kao izlaz daje vektor od 16 elemanata koji se potom prosleđuje konačnom potpuno povezanom sloju. Izlaz iz konačnog sloja predstavljaju vrednosti povoljnosti akcija.

Odabir predstavljene arhitekture posledica je testiranja različitih arhitektura za rešavanje problema predstavljenog u ovom radu. Testiranjem je ustanovljeno da jednostavnije arhitekture nisu davale dovoljno dobre rezultate, dok su složenije arhitekture zahtevale dosta duže vreme obučavanja. Kako je ova arhitektura pokazala zadovoljavajući odnos između performansi i brzine obučavanja bila je izabrana za potrebe rešavanja ovog problema.

Predstavljena neuronska mreža implementirana je kao Python klasa koja nasleđuje klasu Modul iz biblioteke PyTorch [8], modula torch.nn. Ova klasa redefiniše metodu *forward()* čija namena je specificiranje slojeva i veza između slojeva neuronske mreže kao i toka podataka kroz neuronsku mrežu. Povratna vrednost ove metode predstavlja izlaz iz neuronske mreže.

3.3. Obučavanje agenta

Proces obučavanja upotrebljen u ovom radu predstavlja kombinaciju metoda obučavanja predstavljenih u radovima [1, 9]. Prvi korak u obučavanju jeste povezivanje sa emulatorom preko instance GameControl klase. Nakon toga vrši se instanciranje dve neuronske mreže π_{net} i V_{net}, od kojih prva služi za odabir optimalne politike (eng. *policy network*) dok druga služi za vršenje procene vrednosti stanja (eng. *value network*). Težine obe mreže se inicijalizuju istim nasumično izabranim vrednostima. Potom se inicijalizuje i replay memorija kapaciteta od 100000 stanja. Sledeći korak je pokretanje glavne petlje obučavanja. Ova petlja se izvršava dok broj iteracija petlje ne dostigne zadati broj epizoda. Na početku svake epizode vrši se inicijalizacija reda F_{queue} koji može da skladišti do tri poslednje slike ekrana video igre. Nakon ovoga vrši se restartovanje igre i otpočinjanje petlje epizode. U svakoj iteraciji petlje epizode vrši se izbor akcije. Akcija se može izabrati nasumično ili na osnovu izlaza neuronske mreže π_{net}. Da li se akcija bira nasumično ili pomoću π_{net} određeno je ϵ-pohlepnom politikom. Verovatnoća nasumičnog izbora akcije računata je po jednačini 1 [10].

$$\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end})e^{-\frac{n}{\epsilon_{decay}}} \quad (1)$$

Za parametre ϵ_{start}, ϵ_{end} i ϵ_{decay} uzete su vrednosti 0.95, 0.25, 120000. Izbor ovih vrednosti ustanovljen je empirijski. Nakon izbora akcije dobavlja se novo stanje video igre i pravi kopija reda F_{queue}, F'_{queue}. Ova kopija služi za čuvanje slika prethodnog stanja video igre. Zatim se vrši provera kraja epizode i popunjavanje replay memorije. Smatra se da je do kraja epizode došlo ukoliko je automobil usporio. Ukoliko je došlo do kraja epizode u replay memoriju se upisuje torka (F'_{queue}, $akcija$, $None$, 1), gde je $None$ oznaka za terminalno stanje, a 1 visina nagrade, i potom se izlazi iz petlje epizode. U suprotnom u replay memoriju se upisuje torka (F'_{queue}, $akcija$, F_{queue}, 1) i petlja epizode se nastavlja. Po završetku ažuriranja replay memorije i nakon svakih pet epizoda 100 puta se vrši ažuriranje težina mreže π_{net}. Ažuriranje težina π_{net} se vrši tako što se prvo iz replay memorije nasumično izabere 128 uzoraka za koje π_{net} izračuna predikcije vrednosti V. Potom se uz pomoć V_{net} računaju očekivane nagrade V' za izabrane uzorke po jednačini 2. Gde v predstavlja vrednost trenutnog stanja, γ faktor za koji se umanjuje vrednost posmatranog stanja s_t.

$$V' = v + \gamma \max(V_{net}(s_t)) \quad (2)$$

Vrednosti V i V' se zatim prosleđuju Huberovoj funkciji greške, jednačine 3 i 4 [11], koja se koristi u procesu minimizacije razlike između vrednosti V i V'.

$$E(V', V) = \frac{1}{n}\sum_{i}^{n} z_i \quad (3)$$

$$z_i = \begin{cases} 0.5(V' - V)^2, & |V' - V| < 1 \\ |V' - V| - 0.5, & |V' - V| \geq 1 \end{cases} \quad (4)$$

Dobijena vrednost greške se potom propagira niz π_{net} nakon čega se pokreće Adam algoritam za ažuriranje težina [12]. Težine neuronske mreže V_{net} ažuriraju se samo ukoliko je od njihovog poslednjeg ažuriranja proteklo 100 epizoda. Pri tome ažuriranje V_{net} podrazumeva kopiranje parametara mreže π_{net} u V_{net}. Nakon opcionog ažuriranja težina otpočinje se sledeća iteracija glavne petlje obučavanja.

4. REZULTATI

Evaluacija performansi ovog rešenja vršena je poređenjem rezultata postignutih eksploatacijom obučenog agetna i rezultata postignutih nasumičnim izborom akcija. Rezultati su izraženi u prosečnom broju akcija izvršenih do terminalnog stanja u 100 epizoda. Prikupljanje rezultata postignutih od strane agenta vršeno je u tri ključna trenutka obučavanja, nakon 0 epizoda, nakon 3000 epizoda i nakon 10000 epizoda obučavanja. Tabela 1 prikazuje ostvarene performanse agenta u navedenim trenucima.

Tabela 1. *Performanse agenta u ključnim trenucima obučavanja*

Broj epizoda obučavanja	Prosečan broj akcija izvršenih do terminalnog stanja
0	74.87
3000	116.77
10000	123.94
Nasumičan izbor akcija	109.55

Rezultat dobijen nakon 0 epizoda obučavanja posledica je nasumično inicijalizovanih težina neuronske mreže koje uzrokuju da agent uvek bira istu akciju što u proseku nakon 74 akcije dovodi do silaženja automobila sa puta i završetka epizode. Iako rezultat ostvaren nasumičnim izborom deluje prilično blizu rezultatu ostvarenom nakon 3000 epizoda obučavanja, prava razlika između ova dva rezultata ogleda se u tome što agent nakon 3000 epizoda počinje da ciljano izbegava silaženje sa puta. Nakon 10000 epizoda agent počinje da uči obilaženje drugih automobila. Međutim kako je obilaženje složenije od izbegavanja ivice puta agent često ne uspeva da obiđe druge automobile pa je iz tog razloga rezultat približan rezultatu ostvarenom nakon 3000 epizoda obučavanja. Iako ostvareni rezultati deluju ispod prosečnih, oni potvrđuju da se metode predstavljene u radovima [1, 9] mogu primeniti na obučavanje agenata za igranje video igara složenijih od Atari igara, ali i da se za postizanje dovoljno dobrih performansi mora potrošiti puno vremena na obučavanje.

5. ZAKLJUČAK

U ovom radu predstavljena je primena učenja uslovljavanjem za obučavanje agenta za igranje video igre Road Fighter. Obučavanje agenta je vršeno, po uzoru na rad [1], samo uz pomoć slika ekrana video igre. Metoda primenjena za obučavanje agenta predstavlja modifikovane metode iz radova [1, 9]. Rezultati postignuti ovom metodom, iako lošiji od rezultata koje ostvaruje prosečan ljudski igrač, pokazuju da je motda primenljiva ali da zahteva velike količine hardverskih resursa i vremena za obučavanje.

U cilju daljeg poboljšanja performansi agenta moguće je izvršiti izmene postojećeg rešenja poput: povećanja kapaciteta replay memorije, povećanja broja slika koje se koriste za opis stanja video igre, upotrebe ϵ politike takve da se tokom obučavanja daje prednost nasumičnim akcijama na neistraženim delovima igre i upotreba ekspertskog znanja u ranijim fazama obučavanja.

6. LITERATURA

[1] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," p. 9.

[2] D. Silver *et al.*, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *arXiv:1712.01815 [cs]*, Dec. 2017.

[3] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[4] "OpenAI Five," *OpenAI Blog*, Jun-2018. Dostupno na: https://blog.openai.com/openai-five/.

[5] "Nestopia - NES/Famicom Emulator." Dostupno na: http://nestopia.sourceforge.net/.

[6] M. Satran, "Programming reference for Windows API.". Dostupno na: https://docs.microsoft.com/en-us/windows/desktop/api/.

[7] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Feb. 2015.

[8] "PyTorch dokumentacija." Dostupno na: https://pytorch.org/docs/stable/index.html.

[9] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," p. 7.

[10] A. Paszke, "Reinforcement Learning (DQN) Tutorial — PyTorch Tutorials." Dostupno na: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[11] P. J. Huber, "Robust Estimation of a Location Parameter," *Ann. Math. Statist.*, vol. 35, no. 1, pp. 73–101, Mar. 1964.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014.

Kratka biografija:

Ivan Radosavljević je rođen 20. 11. 1993. godine u Loznici, Republika Srbija. Osnovnu školu „Vuk Karadžić" završio 2008. godine. Iste godine se upisao u Srednju tehničku školu u Loznici, smer elektrotehničar računara. Srednju školu završio 2012. godine. Godinu dana kasnije upisuje se na Fakultet tehničkih nauka u Novom Sadu, odsek Elektrotehnika i računarstvo, smer Softversko inženjerstvo i informacione tehnologije u izdvojenom odeljenju u Loznici. Godine 2017. završio osnovne akademske sutdije i upisao master akademske studije na Fakultetu tehničkih nauka u Novom Sadu, odsek Elektrotehnika i računarstvo, smer Softversko inženjerstvo i informacione tehnologije, modul Inteligentni sistemi. Položio sve ispite predviđene planom i programom.