Upravljanje auta agentom obučenim učenjem sa podrškom

Filip Jakovljević
Fakultet tehničkih nauka

1. Uvod:

Automatizacije saobraćaja predstavlja jedan od najvećih problema sadašnjice. I ako se na ovom polju radi još od 20-tog veka, ono još uvek nije kompletno rešeno. Postoje više stepena automatizacije upravljanja vozilom i razni načini i tehnike su pokušane da se postigne željen stepen automatizacije. Kako je testiranje autonomnih vozila u stvarnom okruženju opasno i skupo, razvijeni su mnogi simulatori koji se trude da verno predstave stvarni svet. AirSim (Aerial Informatics and Robotics Simulation) je simulator za dronove i zemna vozila sa otvorenom licencom. Njega je razvio Microsoft u Unreal Engine-u kao podršku za istraživanja u veštačkoj inteligenciji.

Ovaj projekat pokušava da primenama tehnika veštačke inteligencije nauci agenta samostalnom upravljanju autom. Koristeći samo slike prednje kamere automobila iz AirSim simulatora, slično onim informacijama koje bi čovek imao gledajući kroz prednje staklo automobila, agent treba uspešno da se kreće po naselju i izbegavajući sudare. U ovom radu biće prezentovane korišćene tehnike veštačke inteligencije, problemi i ograničenja sa kojima sam se susreo, uspešnost modela kao i mogući dalji načini usavršavanja modela.

2. Osnova/Srodna Istraživanja

2.1 Učenje sa podrškom

Učenje sa podrškom predstavlja sistem od dva aktera, agentom i sredinom, u kojoj agent može da vrši akcije. Odabirom akcije agent vrši interakciju sa sredinom i sredina mu vraća novo stanje i nagradu. Problem učenja sa podrškom predstavljaju učenje šta treba da se radi u nekoj situaciji, odnosno mapiranje situacije na akciju tako da se maksimizuje nagrada [1].

Kao pogodna matematička formulacija za učenje sa podrškom izdvaja se MDP (Markov Decision Process). MDP modeluje skup stanja, skup akcija (koje može da učini agent), funkciju prelaza (funkcija koja nam govori sa kojom verovatnoćom određena akcija vodi u iz prvobitnog stanja u novo stanje) i funkciju nagrade koja se dobija. Ono sto učenje sa podrškom čini učenjem, a ne rešavanjem MDP problema jeste što mi ne znamo ništa od navedenog sem skupa mogućih akcija. Potrebno je sami da dođemo do mogućih stanja, do funkcije prelaza i najbolje politike uz definisanje funkcije nagrade. To radimo pomoću metoda istraživanja.

Ako stanja ne možemo da sagledamo u potpunosti, odnosno samo parcijalno vidimo stanje onda se umesto MPD formulacije koristi POMDP (partially-observed MDP), u kojoj se koristi distribucija mogućih stanja u kojima se nalazimo [2]. Pretpostavićemo da su stanja vidljiva u potpunosti i da radimo sa MDP sistemom.

Cilj je da nađemo politiku koja će uvek iz datog stanja birati najbolju akciju (akciju koja ce u budućnosti doneti najveću nagradu). Postoji podela na dva načina rešavanja problema učenja sa podrškom, rešenja zasnovana na modelu i rešenja koja ne koriste model. U ovom radu biće opisan rešenje bez modela Double DQN (Double Deep Q-Network) koje je korišćeno u projektu za traženje optimalne politike.

2.2. Q-learning

Cilj Q-learning je da nadje optimalnu politiku, odnosno da nadje optimalan niz akcija iz svakog stanja tako da konačna nagrada bude najveća. Kako konačna nagrada ne bi išla u beskonačnost uvodimo standardnu pretpostavku da su dalje nagrade manje bitne uvođenjem

faktora *γ*, time dobijamo da je buduća nagrada za neki korak *t*:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \quad (1)$$

Gde je *T* korak u kome se agent nalazi u završnom stanju, a faktor $\gamma \in [0,1]$.

Optimalnu funkciju stanja i akcije definišemo kao $Q^*(s,a)$, ona predstavlja maksimalnu očekivanu nagradu koju je moguće postići praćenjem neke politike od stanja s:

$$Q^*(s,a) = max_\pi E[R_t | s_t = s, a_t = a, \pi] \quad (2)$$

Gde *π* predstavlja politiku koja mapira stanja i akcije.

Optimalna funkcija ima oblik Belmanove jednačine (Bellman equation), koja prati rekurzivni tok odabira narednih akcija ako je poznata optimalna funkcija za sve moguće akcije iz narednog stanja. Ona glasi:

$$Q^*(s,a) = E_{s',r\sim a}[r + \gamma max_{a'} Q^*(s',a')] \quad (3)$$

Gde je *s* trenutno stanje, *a* predložena akcija, *r* nagrada za naredno stanje *s'*, a *a'* su sve moguće akcije iz stanja *s'*. Ova jednačina je poznata kao Q-vrednost (Q-value). [3]

Pošto Q-vrednosti nisu poznate, cilj Q-učenja je da aproksimira te vrednosti. Kao što je već pomenuto Q-učenje je učenje sa podrškom u kome se ne pravi MDP model. Prvo inicijalizujemo sve Q-vrednosti, zatim kako prolazimo kroz stanja i radimo akcije svaki put pri tome poboljšavamo početnu procenu Q-vrednosti za parove stanja i akcija koje smo doživeli. Procena će se sa vremenom sve više poboljšavati i težiti da dostigne pravu (tačnu) Q-vrednost. Dokaz da algoritam konvergira neće biti dat u ovom papiru.

2.3. Q-network

I ako je Q-učenje jednostavan algoritam koji konvergira ka optimalnom rešenju, u praksi je skoro neupotrebljiv bez određenih modifikacija. Algoritam sam po sebi posmatra svaki par stanja i akcije zasebno, bez ikakve generalizacije, što vrlo brzo može da poraste u nedogled. Naime, ako bi uzeli da je stanje slika (što je slučaju u projektu) Q-učenje bi svaku moguću kombinaciju piksela posmatralo kao različito stanje, što zasigurno nije neophodno i učenje bi trajalo previše dugo.

Zbog ovoga javila se potreba da se stanja generalizuju, najjednostavniji vid generalizacije ranije korišćen bile su linearne aproksimacija. Sa razvojem neuronskih mreža kao univerzalnih generalizatora postale su moguće i kompleksne nelinearne aproksimacije koje se danas sve više koriste [3].

Kod Q-mreža učimo parametrizovanu funkciju $Q(s,a;\theta_t)$, koja generalizuje moguća stanja. Isto kao i kod običnog Q-učenja prilikom svakog koraka poboljšavamo procenu, odnosno ažuriramo naše parametre:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(s_t,a_t;\theta_t))\nabla_{\theta_t} Q(s_t,a_t;\theta_t) \quad (4)$$

Gde je a_t akcija koja preduzeta iz stanja s_t , a θ_t parametri mreže. Faktor učenja je *α*, a ciljna vrednost Y_t^Q se definiše kao:

$$Y_t^Q \equiv r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t) \quad (5)$$

Gde je *r* nagrada za naredno stanje s_{t+1}, a a_{t+1} su sve moguće akcije iz stanja s_{t+1}.[4]

2.4. Deep Q-network

Duboko Q-učenje dobijamo kada koristimo višeslojnu neuronsku mrežu sa Q-učenjem. Cilj duboke neuronske mreže je da izbaci vektor vrednosti za moguće akcije iz ulaznog stanja. Kako bi Q-učenje bilo stabilnije koristi se pomoćna ciljna neuronska mreža mreža i experience replay [5].

Cilja neuronska mreža sa težinama θ^- je ista kao i obučavana neuronska mreža. Obe neuronske mreže kreću sa istim težinama $\theta^- = \theta$, jedina razlika je u tome što se težine ciljne neuronske mreže ažuriraju tek na svakih *τ* koraka $\theta^-_t = \theta_t$, ostatak vremena ostaju iste. Za ciljnu vrednost koristimo izlaz pomoćne mreže, ovo smanjuje

oscilacije u učenju jer cilj ostaje fiksan jedno vreme. Ciljana vrednost sada izgleda:

$$Y_t^{DQN} \equiv r + \gamma \, max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t^-) \quad (6)$$

Experience replay je bufer u kome se skladište doživljene tranzicije, iz bafera uniformno biramo tranzicije na kojima će se obučavati neuronska mreža [6]. Takođe dovodi do velikog poboljšanja performansi algoritma [5].

2.5. Double Deep Q-network

Kod standardnog Q-učenja i Dubokog Q-učenja koristi se *max* operator i za selekciju i za evaluaciju akcije u određenom stanju. Ovo utiče na to da je verovatnije da se izberu precenjene akcije. Da bi sprečili ovaj problem možemo odvojiti selekciju akcije od njene evaluacije, to je ideja Duplog Q-učenja [4].

U Duplom Dubokom Q-učenju ciljna vrednost sada izgleda:

$$Y_t^{DoubleDQN} \equiv r + \gamma Q(s_{t+1}, argmax_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t); \theta_t^-) \quad (7)$$

Može se primetiti da je sada za selekciju zadužena obučavana mreža, a da evaluaciju te akcije vrši ciljna neuronska mreža.

3. Pristup rešavanju problema

3.1. AirSim API i preprocesiranje slike

AirSim kao okruženje obezbeđuje pristup podacima simulatora preko AirSim API-ja. Za korišćenje API-ja potrebno je da prvo bude pokrenut simulator kako bi moglo da se počne sa preuzimanje slika koje će biti ulaz u neuronsku mrežu. U ovom projektu korišćeno je Neighborhood okruženje. Postoje više različitih kamera sa kojih mogu da se skupljaju različiti podaci u zavisnosti od potrebe, kao ulaz su korišćene slike scene sa prednje kamera automobila. Slika koju vraća API-je dimenzija 256x144x3, ova slika je prevelika pa se skalira na dimenzije 86x86x3.

Slika 1. Slika dobijena od simulatora

3.2. Arhitektura modela

U projektu je implementirana duboka Q-neuronska mreža koja se sastoji iz konvolucionog dela, koji je zadužen za izvlačenja karakteristika slike i generalizaciju i skroz povezanog dela koji je zadužen za evaluaciju stanja i kao izlaz daje vrednosti za moguće akcije.

Izgled modela:

1: Input: 86x86x3

2: Conv2D: 8 × 8 size, 1 stride, 64 filters

3: Relu: max(xi , 0)

4: MaxPooling2D: 2x2 size

5: Conv2D: 4 × 4 size, 1 stride, 128 filters

6: Relu: max(xi , 0)

7: MaxPooling2D: 2x2 size

8: Conv2D: 3 × 3 size, 1 stride, 64 filters

9: Relu: max(xi , 0)

10: Dropout: 15%

11: Flatten

12: Dense: 128 features

13: Dropout: 30%

14: Dense: 6 features

Kao izlaz iz neuronske mreže dobijaju se Q-vrednosti za svaku od mogućih akcija (blago levo, levo, pravo, desno, blago desno, koči)

3.3. Funkcija nagrade

Nagrada se dodeljuje agentu za vožnju auta što bliže sredini puta i brzinom koju vozi, ako agent udari u nešto dodeljuje se nagrada od -100. Funkcija nagrde izgleda ovako:

$$r = \ln(speed + 1) * \exp(-dist * 0.3) * 3 \quad (8)$$

Gde je speed brzina auta, dist predstavlja razdaljinu auta od sredine najbližeg puta. I speed i dist su neki broj iz intervala [0, ∞). Ako se auto kreće ispod brzine 0.1 nagrada je 0. Tako da nagrada uvek pozitivna sem u slučaju sudara. Tačke sredina puta su ručno nadjene i prenete u poseban fajl odakle se učitavaju.

3.4. Istraživanje

Istraživanje je jedan od najbitnijih elemenata u učenju sa podrškom. Samo istraživanje daje stohastičnost učenju, prilikom svakog poteza mi biramo da li ćemo uraditi najpovoljniji potez ili nasumični potez. U radu je implementirana strategija smanjujućeg ε. Algoritam kreće sa velikim ε i prilikom svakog poteza ono se smanjuje za neki deo. Efektivno ovo dovodi do toga da algoritam na početku vrši mnogo više istraživanja, a na kraju mnogo više eksploatiše svoje znanje.

3.5. Algoritam

Koraci algoritma idu sledećim redosledom:

1: pokretanje AirSim simulatora

2: konekcija AirSim API-ja sa simulatorom

3: inicijalizacija experience replay bufera

4: zadavanje početne ε vrednosti za istraživanje

5: inicijalizacija $Q_{obučavana}$ i Q_{ciljna} neuronske mreže, inicijalizacija τ i brojača

6: učitavanje stanja (preuzimanje slike)

7: **while true** #početak obučavanja

8: računanje nagrade

9: **if** rand < ε **then**

10: izbor akcije slučajno

11: **else**

12: izbor akcije preko $Q_{obučavana}$

13: **end if**

14: smanjivanje ε

15: izvršavanje akcije, učitavanje novog stanja (preuzimanje slike)

16: punjenje experience replay bafera

17: stanje = novo stanje

18: treniranje mreže

19: brojač++

20: **if** brojač **%** τ **then**

21: Q_{ciljna} dobija težine $Q_{obučavana}$

22: **end if**

23: **end while**

4. Rezultati i moguća poboljšanja

Ograničenja sa resursima za testiranje (4GB Ram i grafička nVidia GeForce GTX 1050) doveli su do toga da se odlučim za jednostavniju (sa manje parametara i slojeva) neuronsku mrežu nego sto bi bilo idealno koja bi se brže obučavala. Pošto se obučavanje vršilo ne pauzirajući simulator (real-time learning) mreža je morala u relativnom kratkom roku da se obučava u svakom koraku što je glavni razlog koji je doveo do smanjena batch size i jednostavnije arhitekture. Memorija je uticala na moguću veličinu slike i samu veličinu experience replay bafera, previše velike slike bi popunile memoriju suviše brzo i bufer bi bio premali. Kasnije je dodato da se na svakih *n* koraka pauzira simulator da se izvrši dodatno obučavanje.

Uprkos ovim ograničenjima prilikom obučavanja agenta mogu se izdvojiti dva stadijuma koja govore da agent uči.

Posle besmislenog kretanja po putu u nekom trenutku agent shvati da mu je najisplativije da ide

samo pravo sa najvećom brzinom dok ne udari. Ovo mu omogućava početna pozicija u simulatora koja je na dosta dugom pravcu.

Drugi stadijum je trenutak kada agent shvati da samo pravo i dalje na kraju dovodi do negativne nagrade, te odluči da samo stoji.

Drugi problem bi se rešio smanjenjem negativne nagrade. Dok prvi problem bi zahtevao način da agent pametnije istražuje. Pošto je prvo zahtevno skretanje dosta daleko od početne pozicije auta dok se agent obuči da dođe do njega (nauči da ide prava) ε postane vrlo malo te auto ne istražuje jako malo.

Da se agent duže trenirao (duže od jedne noći) mislim da bi rezultati bili značajno bolji. Kao poboljšanja koja bi verovatno značajno unapredila brzinu obučavanja jesu pametnije istraživanje i pametniji izbor scena za obučavanje iz experience replay bafera. Takođe moglo bi se dodati automatska detekcija ivica puta i sa time drugačije napraviti funkcija nagrade. Ono sto bi isto doprinelo agentu jesu i dodatne informacije koje se mogu slati neuronskoj mreži (brzina, neki senzori ili niz slika preko kojih bi sama neuronska mreža mogla da odredi pravac i brzinu kretanja)

5. Zaključak

Učenjem sa podrškom je nenadgledano učenje gde agent obučava sam sebe pomoću iskustava sa sredinom. Q-učenje je moćan algoritam, koji konvergira ka tačnom rešenju i ako se ne zaglavi u suboptimalnom ciklusu eventualno će doći do njega. Ali mane njega i ako uvek konvergira su sto za realne probleme je suviše resursno zahtevno i trajalo bi previše dugo. Kao veliko poboljšanje Q-učenje je ukombinovano sa univerzalni aproksimatorima neuronskim mrežama, ipak i dalje obučavanje agenta može da traje dug vremenski period. Jedan od najbitnijih faktora je postavljanje funkcije nagrade bez kojeg agent ne bi mogao da se obuči. Sama ideja učenja sa podrškom ima utemeljenje u psihologiji i blizu je ljudskom razmišljanju. Još jedna od prednosti ovakvog učenja je što se može dešavati u realnom vremenu, gde radeći neku akciju odmah dobijamo potvrdu koliko je dobra i u skladu sa time prilagođavamo naše ponašanje

Reference:

[1] Richard S. Sutton and Andrew G. Barto, (© 2014, 2015). "Reinforcement Learning: An Introduction"

[2] Ted Li , Sean Rafferty, 2017 "Playing Geometry Dash with Convolutional Neural Networks "

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, 2013 "Playing Atari with deep reinforcement learning"

[4] Hado van Hasselt and Arthur Guez and David Silver, 2015 "Deep Reinforcement Learning with Double Q-learning "

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015 "Human-level control through deep reinforcement learning"

[6] L. Lin. 1992 "Self-improving reactive agents based on reinforcement learning, planning and teaching"