Model autonomnog agenta algoritmom Deep Q-learning

Enes Zvorničanin
Vještačka inteligencija
Teorijska kompjuterska nauka
Prirodno-matematički fakultet
Sarajevo, BiH
E-mail: enes.zvornicanin@gmail.com

***Sažetak*—U ovom radu će biti prikazano kako primjeniti duboko učenje podrškom (deep reinforcement learning), tačnije algoritam Deep Q-Learning, na autonomnog agenta kojeg treba naučiti da se kreće određenom stazom. Čitalac će biti upoznat sa osnovnim pojmovima oblasti učenja podrškom te sa cjelokupnom implementacijom programa. Program je urađen u programskom jeziku Python, za grafički prikaz je korištena biblioteka Pygame te dio programa namjenjen za duboko učenje je implementiran uz pomoć biblioteke Keras.**

I. Uvod

Učenje podrškom je vrsta mašinskog učenja koja omogućava kreiranje autonomnog agenta koji će naučiti neko ponašanje u okruženju. Naime, autonomni agent interakcijom sa okruženjem, putem pokušaja i greške, pokušava da stekne znanje na koji će način djelovati u budućnosti. Nakon svake radnje koju izvrši, agent dobija povratnu informaciju o nagradi i sljedećem stanju okruženja. Zadatak agenta jeste da nauči strategiju koja će mu donijeti najveću moguću ukupnu nagradu tokom vremena.
Ukratko rečeno, učenje podrškom se može svesti na nekoliko koraka.

- Naš agent dobija stanje S_0 od okruženja
- Na osnovu trenutnog stanja, agent izvršava radnju A_0
- Okruženje se transformiše u novo stanje S_1
- Agent dobija neku nagradu od okruženja (nagrada može biti i negativna)

U nastavku će biti detaljnije obrađeni pojmovi vezani za učenje podrškom.

II. Neke pojedinosti učenja podrškom

Kao što je spomenuto, učenje podrškom je bazirano na hipotezi maksimiziranja ukupne tj. kumulativne vrijednosti nagrade.Generalno, postoje dva tipa zadataka, epizodičan i neprekidan. Za razliku od neprekidnog, kod epizodičnog postoje početno i krajnje stanje agenta, što čini jednu epizodu. Na primjer u igri Super Mario jedna epizoda bi trajala od početka igre do momenta kada Mario pogine ili pređe nivo. U ovom radu će biti razmatrani samo epizodični zadaci. Prema tome kumulativna nagrada u svakoj epizodi t će iznositi

$$G^{(t)} = R_1^{(t)} + R_2^{(t)} + ... + R_n^{(t)} \tag{1}$$

odnosno

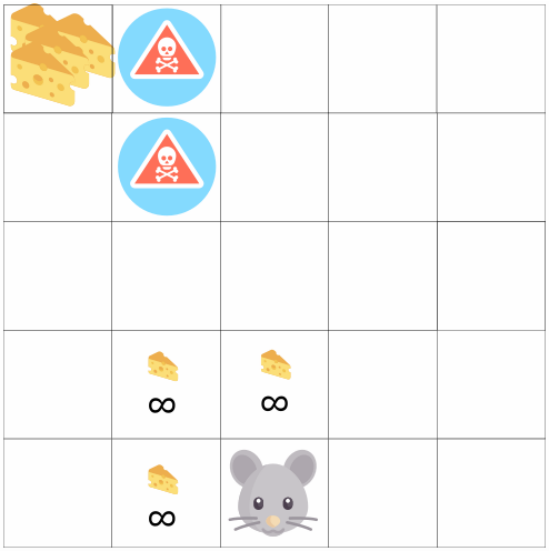
$$\sum_{k=1}^{n} R_k^{(t)} \tag{2}$$

gdje je $R_k^{(t)}$ nagrada koju agent dobija u k-tom koraku u epizodi t a n zavisi od dužine epizode. Međutim, u realnosti ne mogu se tek tako sumirati nagrade. Nagrade koje dolaze na početku epizode su više vjerovatne, jer neizvjesnost raste kako igra odmiče pa se stoga uvodi koeficijent popusta (discount rate) zvani gamma koji je realan broj u intervalu od 0 do 1. Stoga, kumulativna nagrada iznosi

$$\sum_{k=1}^{n} \gamma^k R_k^{(t)} \tag{3}$$

A. Kompromis između istraživanja i iskorištavanja

Cilj istraživanja jeste saznati više informacija o okruženju. Cilj iskorištavanja jeste upotrebljivanje već postojećeg znanja u cilju maksimiziranja kumulativne nagrade. Slika 1 je dobar primjer ovog kompromisa. Ideja je da se maksimizira



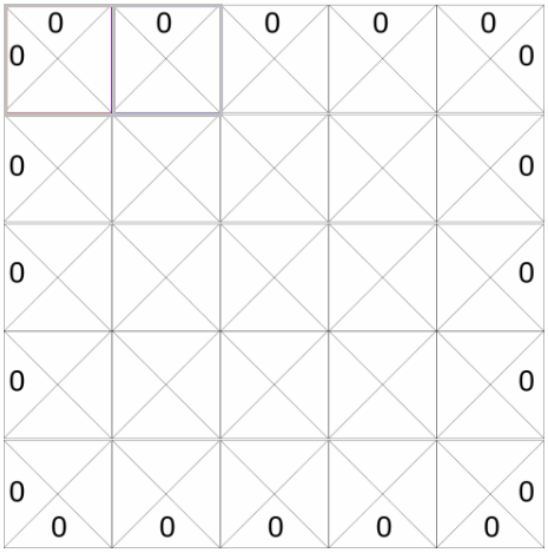
Slika 1: Istraživanje/iskorištavanje

kumulativna nagrada. U igri na slici 1 miš ima beskonačno

mnogo malih sireva koji recimo nose +1 nagradu dok je u lijevom gornjem uglu gomila sireva koja nosi +1000. Ukoliko se agent samo fokusira na skupljanje nagrade, odnosno na iskorištavanje, nikada neće stići do ogromne gomile sireva pri vrhu. Stoga, ovaj problem se rješeva principom sličnim simultanom kaljenju. Na početku agent samo istražuje. Kako epizode odmiću, agent sve manje i manje istražuje te se fokusira na iskorištavanje. Spomenuti metod će biti implementiran pomoću koeficijenata epsilon, koji je na početku jednak jedinici a s vremenom se smanjuje do neke minimalne vrijednosti (recimo 0.1 u zadatku). Što je epsilon veći to je veća vjerovatnoća da agent izabere slučajnu radnju u datom koraku. Epsilon se smanjuje iz epizode u epizodu na način tako što se na kraju svake epizode množi sa koeficijentom koga zovemo epsilon decay pri čemu se postupak množenja prekida kada epsilon dostigne neku unaprijed zadanu minimalnu vrijednost.

B. Q-learning

Za primjer naveden na slici 1 se može kreirati tablica prikazana na slici2. Svaki kvadrat predstavlja stanje agenta



Slika 2: Q tablica

i on je podijeljen na 4 trougla jer za svako stanje postoje 4 moguće radnje (idi lijevo,desno,dole,gore). Svaki trougao će sadržati broj koji će predstavljati očekivanu buduću vrijednost nagrade za dato stanje i radnju. Ta vrijednost će biti izračunata Bellman-ovom jednačinom oblika

$$Q(s,a) := Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s^{'}, a^{'}) - Q(s,a)] \tag{4}$$

gdje su s trenutno stanje, a trenutna radnja (akcija), α stopa učenja (learning rate), γ koeficijent popusta i $s^{'}$ novo stanje dobijeno iz prethodnog stanja s radnjom a. Postupak je dat sa

Algorithm 1 Q - learning

1: Inicijaliziraj Q-vrijednost za sve parove stanja i radnji
2: Dok učenje traje ponavljaj:
3: Izaberi radnju a za trenutno stanje s
4: Izvrši radnju a
5: Primjeni jednačinu 4

Ovakav pristup je efikasan samo za okruženja sa relativno malim brojem stanja. Nije efikasno kreirati ovakvu tablicu za okruženja sa velikim brojem stanja i radnji. Stoga se uvodi nova strategija rješavanja problema ove vrste.

III. Deep Q-Learning

Ideja Deep Q-Learning algoritma je da umjesto Q-tablice koristi neuronsku mrežu koja će aproksimirati Q-vrijednost za svaku akciju baziranu na stanju. U radu [1] DeepMind je uspješno predstavio ideju ovog algoritma igrajući neke Atari igre. Prikazano je kako autonomni agenti mogu igrati igre iako nemaju nikakvih informacija o istima osim slike (*screenshot*) igre u datom trenutku. U tom radu je korištena konvolucijska neuronska mreže koja je kao ulaz primala sliku trenutnog stanja igre. U ovom radu će biti korištena obična neuronska mreža sa ulazom koji će biti numerički vektor koji predstavlja vrijednosti određenih senzora. Detaljno objašnjenje arhitekture neuronske mreže i pojedinih njenih dijelova će biti izostavljeno jer to nije cilj ovog rada i smatrat će se da je to neka "*crna kutija*" koja na osnovu ulaznih parametara vrši predikciju što je i prikazano na 3.

Slika 3: Model neuronske mreže

Dakle, svaki neuron na izlazu odgovara po jednoj radnji agenta pa će se izabrati ona radnja koja ima najveću Q vrijednost. Prije nego što će biti prikazano objašnjenje na koji način se neuronska mreža trenirati treba uvesti jos nekoliko pojmova vezanih za model. Kako model ima tendenciju da zaboravlja prethodno iskustvo uvodi se niz fiksne dužine, sa osobinom strukture red, koji će predstavljati memoriju. Neka agent ima početno stanje **state** i neka je uradio akciju **action** i dobio nagradu **reward** te se našao u novom stanju **nextState**. Neka varijabla **done** označava da li je igra završena. Tada će se u memoriju u svakom koraku ubacivati vektor.

$$(state, action, reward, nextState, done) \tag{5}$$

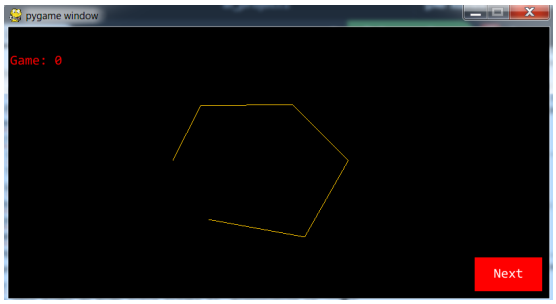
Kada agent pogine, odnosno na kraju svake epizode vršit će se trening. Iz memorije se na slučajan način uzima određeni broj elemenata (*minibatch*) i sa njima se vrši trening. Generalno kada se trenira neuronska mreža osim ulaznih podataka mora se proslijediti i ona vrijednost koja je tačna za dati ulaz (*label*). Na primjer ako se radi klasifikacija ručno napisanih cifara, za ulaz koji odgovara cifri 1 mora se negdje i reći da je to cifra 1. Dakle, ako ulazni parametri govore da je sa lijeve strane prepreka onda se očekuje da će agent skrenuti desno. To se postiže algoritmom

Algorithm 2 Replay

1: Na slučajan način izaberi minibatch veličine s iz memorije
2: Za svaki (state, action, reward, nextState, done) iz minibtch ponavljaj:
3: $t = reward$
4: Ako nije done:
5: $t = reward + \gamma \max_{a'} Q(nextState, a')$
6: $t_f = (Q(state, a1), Q(state, a2), Q(state, a3))$
7: $t_f[action|action == a_{i_0}, i_0 \in (1, 2, 3)] := t$
8: Treniraj mrežu za ulaz: state i label: t_f

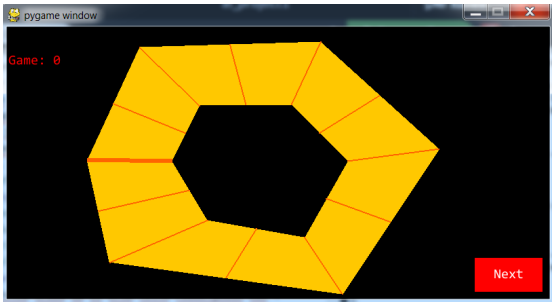
IV. Kreiranje okruženja

Projekat se sastoji iz 5 fajla: car.py, constants.py, DQNAgent.py, game.py, track.py. U fajlu constants.py su definisane neke konstante kao i matematičke funkcije za orijentaciju, presjek segmenata i poligona, udaljenost tačke od tačke ili plave i slično. U fajlu track.py implementirana je klasa Track koja će predstavljati stazu po kojoj se agent kreće. U klasi car.py napisana je klasa Car koja predstavlja agenta. Osim toga u ovoj klasi se nalazi i cjelokupna logika oko presjeka agenta sa trakom kao i računanju senzora. U fajlu DQNAgent je urađena cjelokupna logika vezana za Deep Q-Learning algoritam te u game.py logika igre. Kada se pokrene program prikaže se prazan ekran i na njemu korisnik započinje crtati prvo unutrašnju ivicu staze tako što klikne na mjesto gdje treba da bude čvor. Preporučuje se crtanje u smjeru kazaljke na satu. Kada se iscrtaju svi čvorovi potrebno je kliknuti na dugme *Next* te se počinje crtanje vanjske ivice trake. Nije potrebno crtati posljednju ivicu tj. dovoljno je označiti gdje su svi čvorovi kao na slici 4. Vanjska ivica treba da se crta



Slika 4: Unutrašnje ivica

istim redosljedom kao i unutrašnja te se preporučuje da ima isti broj čvorova. Kada se iscrta vanjska ivica, dobije se staza prikazana na slici 5. Nakon toga se ispred podebljane linije

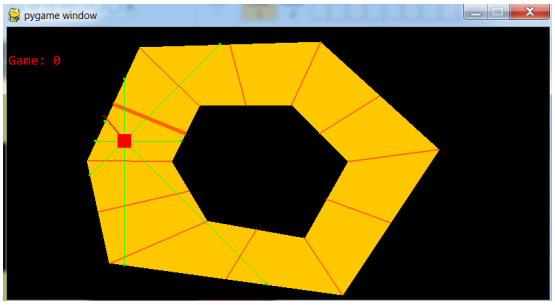


Slika 5: Prikaz staze

klikne da bi se postavio agent te se klikne na dugme *Next* za početak. Staza je prekrivena linijama (reward gates) kroz koje kada agent pređe dobija nagradu. Tačnije, podebljana linija je trenutna i kada agent prođe kroz nju dobija nagradu te se trenutna prebacuje na sljedeću.

A. Verzija 1

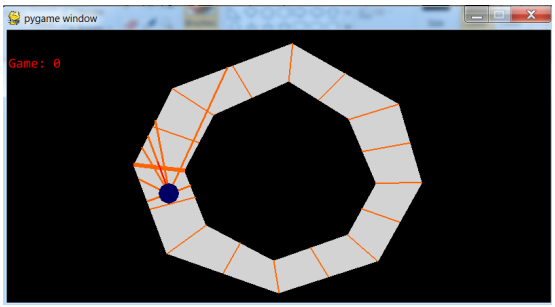
Prva verzija ovog programa nije dala neke rezultate iako su pravljene izmjene ako arhitekture neuronske mreže, štimanja parametara, raznih vrijednosti nagrada, uvedena pozitivna, negativna i neutralna memorija. Senzori koji su korišteni, prikazani na slici 6, su statički i oni se aktiviraju kada korisnik pritisne dugme S na tastaturi.



Slika 6: Statički senzori

B. Verzija 2

U verziji 2 su uvedeni dinamički senzori koji se rotiraju zajedno sa agentom a prikazani su na slici 7.

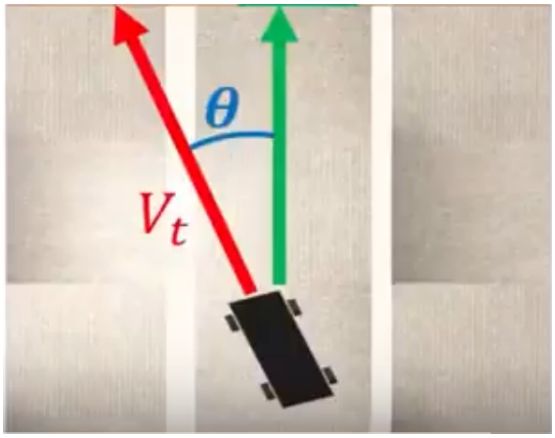


Slika 7: Dinamički senzori

Tada su se pojavili i prvi rezultati što se može vidjeti na videu 1 priloženom uz ovaj rad. Postojala je potreba da se kreira neka funkcija koja bi računala nagradu na način da kada se agent vraća unazad daje negativnu nagradu a u suprotnom pozitivnu. Prvi pokušaj je bio implementiran tako što bi se poredili prethodna i trenutna udaljenost agenta do aktivne kapije i na osnovu toga nagrada bi bila -0.1 ukoliko se udalio te u suprotnom 0.1. Druga ideja je da se računa ugao Θ prikazan na slici 8 te se nagrada R_t dobija iz formule

$$R_t = \alpha V_t(\cos\Theta - \sin\Theta - trackPos) \tag{6}$$

gdje je $trackPos$ udaljenost agenta od zelenog vektora na slici 8, α neka mala konstanta (0.03 u ovom primjeru) i V_t vektor brzine.



Slika 8: Računanje nagrade

Na ovaj način se dobija vrijednost nagrade koja kažnjava agenta ukoliko se on vraća nazad. Rezultat ovakvog pristupa je pokazan u videu2 koji dolazi kao prilog ovom radu

ZAKLJUČAK

Ovim radom je pokazano na koji način se može kreirati autonomni agent pomoću algoritma Deep Q-Learning. Postupak nije jednostavan jer program sadrži mnogo parametara koji se trebaju namještati i samo eksperimentisanje sa jednim skupom parametara može da traje satima. Ostavljeno je prostora za poboljšanje rezultata a može se i probati primjeniti neki drugi algoritam na ovo okruženje (recimo NEAT algoritam).

LITERATURA

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. *Playing atari with deep reinforcement learning*. arXiv preprint arXiv:1312.5602.

[2] Deep Q-Learning with Keras and Gym https://keon.io/deep-q-learning/ (pristupljeno 04.05.2019)

[3] An introduction to Reinforcement Learning, https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419 (pristupljeno 29.04.2019)

[4] Diving deeper into Reinforcement Learning with Q-Learning, https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe (pristupljeno 30.04.2019)

[5] Autonomous Driving Car using Deep Q Network and Reinforcement Learning // Python, Unity https://www.youtube.com/watch?v=ST6aFILmnjc (pristupljeno 04.05.2019)