SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

PRIMJENA REINFORCEMENT LEARNINGA ZA KONTROLU 2D VOZILA.

Seminarski rad

Raspoznavanje uzoraka i strojno učenje

Luka Šimić

Osijek, 2020

Sadržaj

1. UVOD

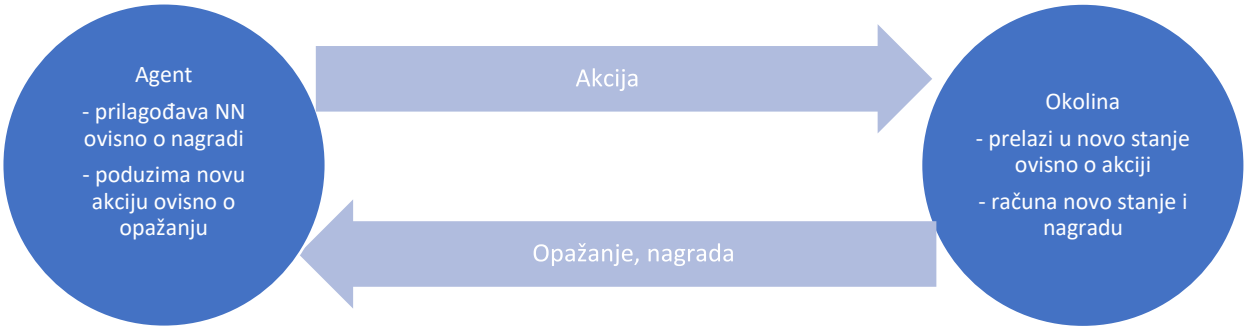
Reinforcement učenje (*engl. reinforcement learning*) je postupak kojim treniramo agent kako bi donosio odluke(*engl. action*) temeljene na dostupnim opažanjima (*engl. observation*). Agent se uči tako što za određenu odluku ili niz odluka dobije nagradu (*engl. reward*). U slučaju dobre odluke, nagrada je pozitivna, dok je u slučaju loše odluke nagrada negativna. Agent s vremenom teži tome da dobije najveću moguću nagradu, što će ostvariti donošenjem optimalnih odluka.

Projekt se sastoji od agenta koji uči upravljati pojednostavljenim modelom vozila. Cilj agenta je u što kraćem vremenu (u što manjem broju sličica(*engl. frames*) posjetiti unaprijed definirane točke u 2D ravnini.

Projekt je implementiran koristeći programski jezik *Python* i biblioteku *tf-Agents* [1] *(tensorflow-agents*).

2. REINFORCEMENT LEARNING

Pojednostavljeni prikaz *reinforcement learninga* prikazan je na slici 2.1.



Slika 2.1 - Pojednostavljeni tijek reinforcement learninga

Osnovni tijek učenja je slijedeći. Agent, odnosno NN se inicijalizira s nasumičnim težinama. Agent u ovisnosti o opažanju iz okoline na svom izlazu daje određenu akciju. Okolina mijenja svoje stanje u ovisnosti o poduzetoj akciji, te vraća vrijednosti opažanja i nagrade za prethodni korak. Zatim se težine NN podešavaju ovisno o tome je li nagrada bila pozitivna, odnosno negativna, i većeg, odnosno manjeg iznosa.

2.1. Agent

Reinforcement learning agent predstavlja algoritam kojim se rješava određeni *Reinforcement learning* problem. Agent za donošenje odluka najčešće koristi neuronsku mrežu, ali može koristiti i neke druge modele za donošenje odluke.

Svrha agenta je da procesira opažanja i nagradu koju dobije iz okoline, te generira akciju. Zatim na osnovu povratne informacije(novog opažanja i nagrade) odlučuje je li donesena odluka u prethodnom koraku bila dobra ili loša, te optimizira model/mrežu za donošenje odluke kako bi kod budućih slučajeva povećao vjerojatnost donošenja odluke koja daje najveću nagradu

Neki od algoritama koji se često koriste su *brute force, Value function, Monte Carlo methode, Q learning…*

2.2. Okolina, opažanja i akcije

Okolina (*engl. environment*) opisuje okruženje u kojemu se agent nalazi, te iz kojega dobiva opažanja. Okolina u svakom koraku, ovisno o poduzetoj akciji prelazi u novo stanje. Za okolinu ne mora biti poznato optimalno analitičko rješenje, niti okolina mora biti računalno simulirana, ali je bitno da agent smije donositi loše odluke bez štetnih posljedica.

Opažanja mogu biti vektori i matrice, odnosno slike, koji predstavljaju stanje okoline. Opažanja u sebi ne sadrže sve varijable koje sadrži okolina, već samo određene varijable koje su na neki način vidljive agentu.

Akcije mogu biti u obliku vektora ili matrice, te mogu biti diskretne ili kontinuirane. Kod diskretnih akcija agent odabire jednu od mogućih akcija, dok kod kontinuiranih odabire vrijednost iz prethodno definiranog raspona

2.3. Funkcija nagrade

Funkcija nagrade (*engl. reward function*) je funkcija koja računa nagradu nakon poduzete akcije.

Velik problem *reinforcement learniga* je upravo oblikovanje ove funkcije. U slučaju gdje okolina dodjeljuje nagradu samo u koraku kada dođe u željeno stanje, agent će morati dobro odabrati sve korake u nizu kako bi došao do nagrade. Često je taj broj koraka velik, pa je vjerojatnost da će agent svojim akcijama dovesti okolinu u traženo stanje minimalna. Zbog toga agent nikada ne može dobiti pozitivnu nagradu, pa zbog toga i samo rješenje nikada ne konvergira. Ta se situacija naziva *sparse reward setting*.

Jedan od načina na koji možemo pristupiti ovom problemu je da svaki korak damo malu nagradu, kako bismo bolje usmjerili agent prema željenom stanju. U ovom slučaju postoji vjerojatnost da agent ne konvergira prema optimalnom rješenju, već se prilagodi navedenim malim nagradama kako bi maksimizirao nagradu. U najgorem slučaju agent se može *overfittati* malim nagradama tako što pronađe način da ih eksploatira, i nikada neće odvesti okolinu u traženo stanje.

Također, jedan od problema je što određene akcije koje agent poduzme često nemaju trenutačni efekt, i njihov će utjecaj tek biti vidljiv nakon nekoliko koraka. Analogno tome, nagrada ne proizlazi nužno iz akcija koje su poduzete u prethodnom koraku.

3. IMPLEMENTACIJA

Programsko rješenje implementirano je koristeći programski jezik Python i biblioteku *tf-learn*. Okolina predstavlja vozilo s jednostavnim fizikalnim modelom [2] koje mora obići definirane točke u 2D ravnini. Osnovni izgled implementacije jedne okoline dan je isječkom koda 3.1.

Svaka okolina(*engl. environment*) treba sadržavati određene funkcije:

- step(action) – mijenja stanje okoline ovisno o poduzetoj akciji, i kao rezultat vraća nagradu(*engl. reward*) i opažanje(*engl. observation*).
- reset() – postavlja stanje okoline na prethodno definirane ili nasumične vrijednosti.

Uz to, okolina sadrži funkcije koje vraćaju specifikaciju opažanja i dostupnih akcija

- action_spec()
- observation_spec()

```python
class MyEnv(py_environment.PyEnvironment):
    def __init__(self):
        ...
        self._reset()

    def observation_spec(self):
        return self._observation_spec

    def action_spec(self):
        return self._action_spec

    def _reset(self):
        ...
        return time_step.restart(observation)

    def _step(self, action):
        ...
        return time_step...(observation, reward)
```

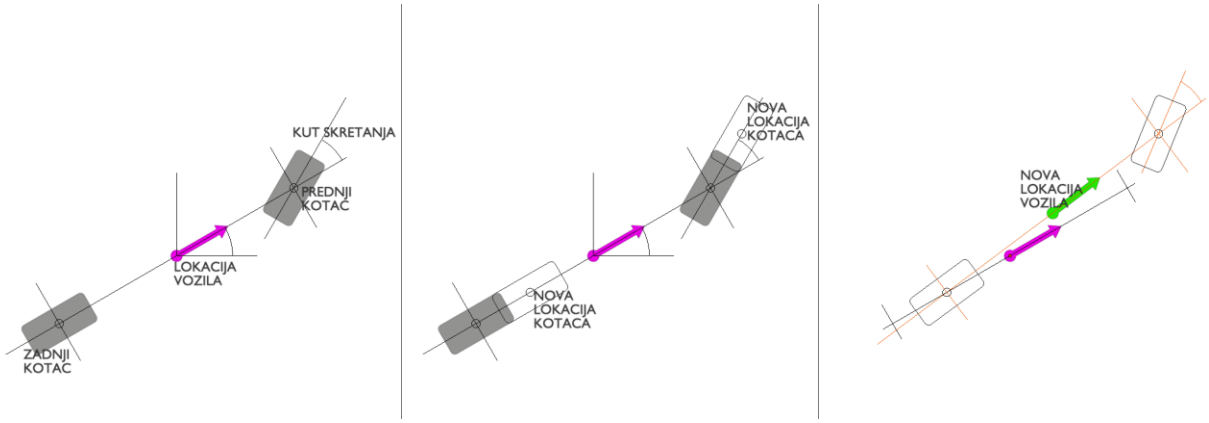
Isječak koda 3.1 – Osnovni oblik implementacije environmenta

Postoji nekoliko biblioteka, od kojih je najpoznatija *OpenAI Gym* [3], koje služe kao standardna implementacija određenih *reinforcement learning* problema, i koriste se kao *benchmark* za *reinforcement learning* algoritme.

3.1. Implementacija okoline

Fizika vozila radi na sljedećem principu koji je prikazan na slici 3.1.:

- Izračunamo poziciju prednje i stražnje osovine koristeći definirani osovinski razmak i poznati vektor kretanja vozila
- Pomaknemo stražnju osovinu u smjeru kretanja vozila, a prednju osovinu u smjeru kretanja vozila + kut skretanja
- Zatim izračunamo novu poziciju vozila kao sredinu između pozicija prednje i stražnje osovine.



Slika 3.1 - fizika vozila

Opažanje je oblika vektora od 7 vrijednosti, float32 tipa podatka. Unutar opažanja dostupne su slijedeće vrijednosti, normalizirane u interval [-1, 1]

- X, Y koordinate trenutnog cilja
- X, Y koordinate trenutne pozicije vozila
- X, Y elementi vektora koji predstavlja trenutni smjer kretanja.
- Kut između vektora kretanja i vektora smjera prema trenutnom cilju

Svaki korak može se poduzeti jedna akcija iz skupa {0, 1, 2}

- 0 – vozilo ne skreće
- 1 – vozilo skreće ulijevo (obrnuto smjeru kazaljke na satu)
- 2 – vozilo skreće udesno (u smjeru kazaljke na satu)

Simulacija se prekida u slijedećim slučajevima

- Vozilo izađe izvan definiranih granica simulacije
- Vozilo dosegne traženu točku
- Dosegnut je maksimalan broj koraka (*frameova*) simulacije.

3.2. Funkcija nagrade

Nagrade se dodjeljuju prema slijedećoj funkciji:

- $\frac{\text{početni kut - krajnji kut}}{\text{max. kut skretanja}}$ za svaki korak gdje se vozilo odmakne od trenutnog cilja
- $25 + 75 \cdot \left(1 - \frac{\text{tren. korak}}{\text{max } korak}\right)$ svaki put kada vozilo dođe do ciljane točke
- -100 ako vozilo izađe izvan definiranih granica simulacije
- $25 \cdot \left(1 - \frac{trenutna\ udaljenost}{početna\ udaljenost}\right)$ na posljednjem koraku simulacije
- 2.5 ako se vozilo stvori unutar cilja

3.3. Implementacija agenta i treniranje

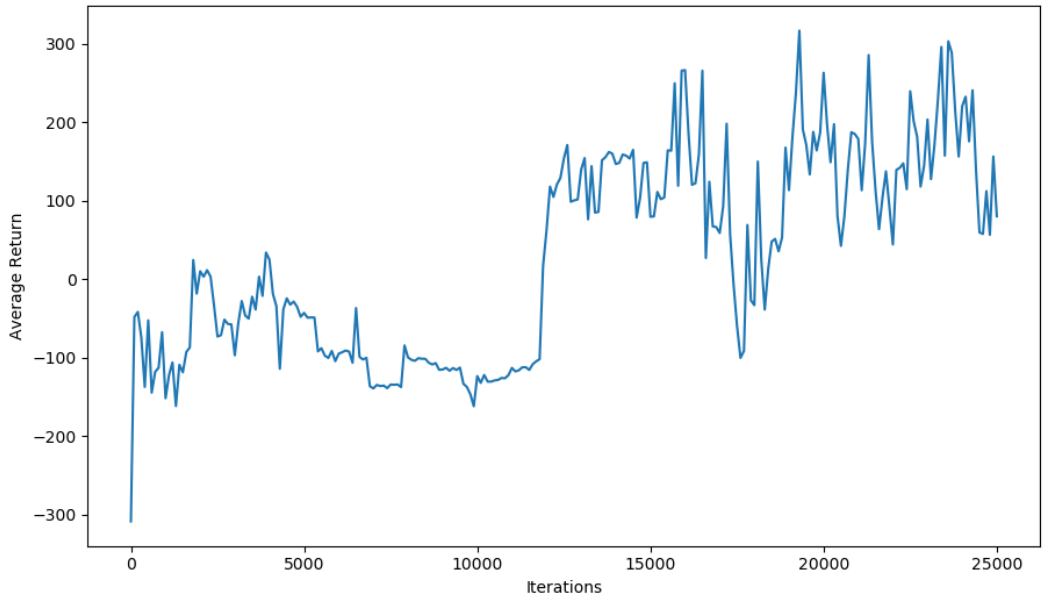
Za programsko rješenje koristi se DQN agent [4]. DQN agent koristi Q-Mrežu (*engl. QNetwork*), neuronsku mrežu koja predviđa Q-vrijednosti (*engl. QValues*) za svaku akciju iz trenutnog opažanja. Q-vrijednost predstavlja očekivanu nagradu.

Koristi se mreža sa ulaznim slojem od 7 neurona(pošto imamo 7 vrijednosti u vektoru opažanja), 3 izlazna neurona (zbog toga što postoje 3 moguće akcije), te dva *fully connected* sloja gdje svaki sadrži 32 neurona. Primjer stvaranja Q-Mreže dan je isječkom koda 3.2.

```
q_net = q_network.QNetwork(
    train_env.observation_spec(),
    train_env.action_spec(),
    fc_layer_params=(32,32)
)
```

Isječak koda 3.2 - stvaranje Q-Mreže koristeći biblioteku tf-agents

Kao metriku za testiranje koristimo prosječnu nagradu za deset okolina(koje su svaki puta pri testiranju i evaluaciji jednake, za što koristimo numpy.seed()). Mreža je trenirana 25000 iteracija, a svakih 100 iteracija provedena je evaluacija modela. Rezultati treniranja prikazani su na slici 3.2. Prosječna nagrada nakon 13000 iteracija oscilira oko 200.



Slika 3.2 – prosječna nagrada u ovisnosti o iteracijama.

Agent se trenira u petlji, gdje u svakoj iteraciji prikupimo iskustvo(engl. Experience) određeni broj epizoda (nizovi opažanja, akcije i nagrade) koje spremimo u *replay buffer*. Nakon toga treniramo se agent trenira na tom iskustvu i prilagođava težine neuronske mreže. Uz to, u određenim intervalima provjeravamo prosječnu nagradu i spremamo *checkpoint* (trenutno stanje) mreže. Primjer petlje za treniranje agenta dan je isječkom koda 3.3.

```
for _ in range(num_iterations):
    collect_data(train_env, agent.collect_policy, collect_episodes_per_iteration)
    experience, unused_info = next(iterator)
    train_loss = agent.train(experience)
    step = agent.train_step_counter.numpy()
    if step % eval_interval == 0:
        numpy.random.seed(10)
        avg_return = compute_avg_return(eval_env, agent.policy, num_eval_episodes)
        print('step = {0}: Average Return = {1}'.format(step, avg_return))
        numpy.random.seed()
    if step % checkpoint_interval == 0:
        checkpoint = tensorflow.train.Checkpoint(q_net=q_net, optimizer=optimizer)
        checkpoint.save(file_prefix=CHECKPOINT_PREFIX)
```

Isječak koda 3.3 – petlja za treniranje agenta

3.4. Testiranje treniranog agenta

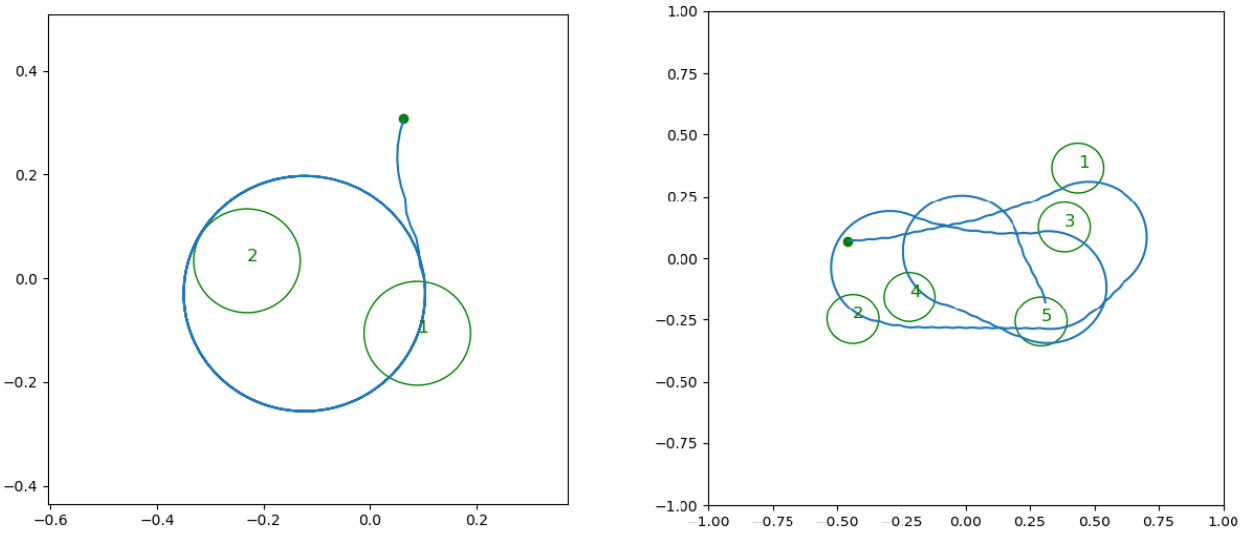
Testiranje se vrši na okolini koja ima slično ponašanje, ali nakon što vozilo posjeti ciljanu točku, trenutni korak simulacije postavljamo na nulu, te nasumično generiramo novu točku. Q-Mrežu učitavamo koristeći *checkpoint* sustav.

Svako opažanje spremamo u listu, koju na kraju zapisujemo u *.csv* datoteku, te prikazujemo putanju kroz točke koristeći biblioteku *matplotlib*. Primjer petlje za testiranje dan je isječkom koda 3.4., dok je primjer putanje kroz 10 točaka dan na slici 4.1.

```
time_step = eval_env.reset()
result.append(time_step.observation[0].numpy())
while not time_step.is_last():
    action_step = agent.policy.action(time_step)
    time_step = eval_env.step(action_step.action)
    result.append(time_step.observation[0].numpy())
with open("trained_outputs.csv","w", newline='') as outfile:
    csv_out=csv.writer(outfile)
    for row in result:
        csv_out.writerow(row)
```

Isječak koda 3.4 – petlja za testiranje agenta

4. REZULTATI



Slika 4.1 – primjeri generirane putanje.

Generirana *.csv* datoteka se koristi za generiranje animacije u programskom paketu Blender. Blender je alat za 3D modeliranje, koji pruža mogućnost automatiziranja jednog dijela procesa koristeći Python skripte. Python skripta je napisana koja za svaki *frame* animacije postavlja lokaciju i rotaciju vozila, te objekta koji predstavlja trenutni cilj. Primjer *renderane* animacije dostupan je u repozitoriju.

Agent dobro obilazi definirane točke i izbjegava izlazak izvan granica simulacije. Do problema dolazi u slučajevima gdje su uzastopne točke relativno blizu, te vozilo ne može skrenuti dovoljno jako kako bi se približilo ciljanoj točki(slika 4.1. lijevo)

5. ZAKLJUČAK

Reinforcement learning je dobar postupak u slučajevima gdje okolinu (*engl. environment*) možemo promatrati kao crnu kutiju (*engl. black box*). Primjenjiv je u slučajevima gdje je moguće dozvoliti agentu da izvodi kritične greške, poput računalnih simulacija(npr. Igara, sustava upravljanja i slično). Pri stvaranju takvih simulacija bitno je da one što bolje predstavljaju stvarne uvjete. Dobro ga je primijeniti u slučajevima gdje nije moguće analitički doći do dobrog rješenja, ali je poznat željeni cilj/stanje sustava prema kojemu agent treba težiti.

Najveći problem kod *reinforcement learning* algoritama je oblikovanje funkcije nagrade. U slučajevima kada za svaki korak ne dajemo nagradu agent će rijetko doći do zadovoljavajućeg stanja, te će zbog toga konvergencija biti gotovo nemoguća. Pri oblikovanju funkcije nagrade također je bitno da agenta ne ograničimo previše, jer će se u tom slučaju prilagoditi funkciji nagrade i neće naći neko drugo (možda optimalnije) rješenje. Velik problem predstavljaju i rubni slučajevi (*engl. edge cases*), koji će se rijetko pojavljivati pri treniranju, ali kasnije svojom pojavom mogu dovesti do grešaka.

Zbog toga je još uvijek dobro koristiti konvencionalne algoritme. Iako konvencionalni algoritmi možda nisu optimalna rješenja, nude prednosti pri obradi rubnih slučajeva, lakšu proširivost i često bolju brzinu izvođenja, pa su zbog toga još uvijek dobar izbor. Te je algoritme moguće proširiti *reinforcement learning* algoritmima kako bi se pronašla moguća druga rješenja.

6. LITERATURA

[1] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke i E. Brevdo, »TF-Agents: A library for Reinforcement Learning in Tensorflow,« 2018. [Mrežno]. Available: https://github.com/tensorflow/agents. [Pokušaj pristupa January 2020].

[2] »Simple 2D car steering physics in games,« 2010. [Mrežno]. Available: http://engineeringdotnet.blogspot.com/2010/04/simple-2d-car-physics-in-games.html. [Pokušaj pristupa January 2020].

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang i W. Zaremba, *OpenAI Gym,* 2016.

[4] V. Mnih, K. Kavukcuoglu, D. Silver i e. al., »Human-level control through deep reinforcement learning,« *Nature,* svez. 518, pp. 529-533, 2015.