

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I
INFORMACIJSKIH TEHNOLOGIJA**

DDPG agent u HalfCheetah okruženju

Seminarski rad

Raspoznavanje uzoraka i strojno učenje

Denis Lazor

Osijek, 2020

SADRŽAJ

1	UVOD	3
2	NEURONSKE MREŽE	4
	2.1. Podržano učenje	6
3	RJEŠENJE PROBLEMA	7
	3.1. HalfCheetah okruženje	7
	3.2. Biblioteke	8
	3.3. DDPG	9
	3.4. Struktura neuronske mreže	10
	3.5. Parametri	12
4	Rezultati	13
5	ZAKLJUČAK	15

1 UVOD

Prošlih godina ostvario se značajan napredak u sposobnostima umjetne inteligencije (*eng. AI*) i metoda podržanog učenja. Moderni AI agenti temeljeni na algoritmima podržanog učenja mogu riješiti probleme za kojima bi čovjeku trebale godine u samo nekoliko sati. Imaju generalnu primjenu, to jest mogu rješavati veliki raspon zadataka sa minimalnim preinakama..

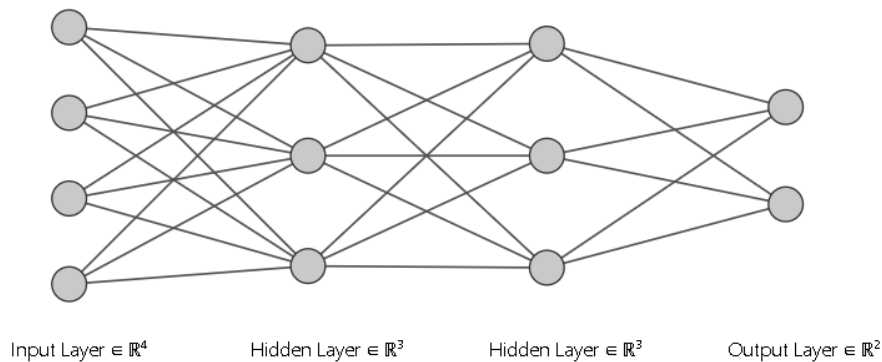
Glavna ideja ovoga rada je naučiti agenta da mijenja u vremenu vrijednosti momenta sila motora zglobova robotskog geparda kako bi se on što prirodnije i brže kretao prema naprijed

Za rješavanje ovog problema korišten je Python 3.7. programski jezik, PyCharm programsko okruženje i Conda okruženje za upravljanje bibliotekama o kojima će biti riječi kasnije.

Učenje je provedeno na AMD RX 580 8GB grafičkoj kartici.

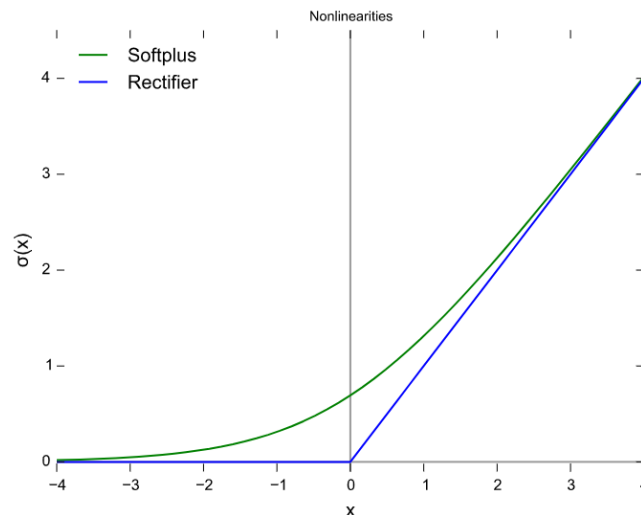
2 NEURONSKE MREŽE

Neuronske mreže, ili umjetne neuronske mreže, algoritmi su strojnog učenja modelirani po uzoru na biološke mreže neurona unutar ljudskog mozga. Današnje neuronske mreže su većinom sve po definiciji duboke neuronske mreže, ili neuronske mreže s dva ili više skrivenih slojeva. Skriveni slojevi (*eng. hidden layer*) su svi slojevi neuronske mreže između ulaznog (*eng. input layer*) i izlaznog sloja (*eng. output layer*). Shema jednostavne potpuno povezane neuronske mreže nalazi se na slici 1.



Slika 1 - shema jednostavne potpuno povezane neuronske mreže [4]

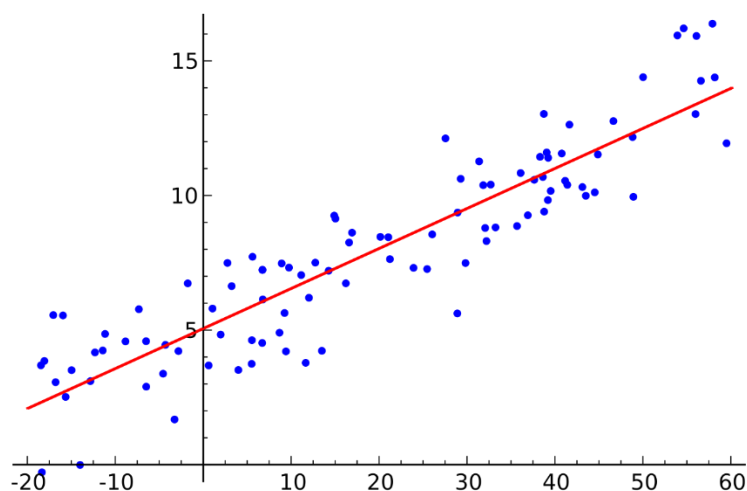
Struktura neuronske mreže s slike 1 je $(4 - 3 - 3 - 2)$, gdje su prvi i četvrti slojevi ulazni i izlazni slojevi, a drugi i treći slojevi su skriveni slojevi. Slojevi se sastoje od neurona (krugovi na slici 1), a svaki neuron je nositelj jednog realnog broja. Ova neuronske mreža je potpuno povezana jer je svaki neuron iz sloja k povezan s svakim neuronom iz slojeva $k-1$ i $k+1$. Za slojeve neurona mogu se vezati funkcije poput aktivacijske funkcije, funkcije za opadanje neurona (*eng. dropout*), *Softmax* [5] funkcija te mnoge druge. Aktivacijska funkcija se primjenjuje na sve neurone sloja te služi za uvođenje nelinearnosti u neuronsku mrežu, jer uvođenjem linearnosti neuronska mreža može estimirati nelinearne funkcije. Dodatno, kodomene aktivacijske funkcije se često prostiru od 0 do 1, pa služe za ograničavanje vrijednosti neurona. Često korištena aktivacijska funkcija je Relu ili Rectified linear unit (slika 2), kojoj je kodomena od 0 do $+\infty$.



Slika 2 - Relu aktivacijska funkcija (plava linija) [5]

Funkcija za opadanje nasumično isključuje neurone sloja kojeg je vezana, prije svake iteracije učenja neuronske mreže. Glavni parametar funkcije za opadanje je koeficijent opadanja neurona, ako je 0, svi neuroni su aktivni tijekom izvođenja iteracije učenja, a ako je 1, svi neuroni su ugašeni tijekom izvođenja iteracije učenja. Time se neuronska mreža manje prilagodi podacima na kojima se uči.

Kod neuronskih mreža imamo problem regresije i klasifikacije. Problem regresije se može shvatiti kao aproksimiranje funkcije i često se rješava jednostavnijim algoritmima strojnog učenja kao što su linearna (slika 3), logistička ili polinomna regresija. Ako se neka funkcija ne može aproksimirati jednostavnijim algoritmima, koriste se neuronske mreže.



Slika 3 - linearna regresija

Za razliku od problema regresije, problem klasifikacije se fokusira na raspodjelu podataka na određene klase. Jednostavniji primjeri su SVM (eng. +

2.1. Podržano učenje

Podržano učenje se sastoji od agenta i okoline. U ovom slučaju je neuronska mreža agent i njena zadaća je da djeluje u okolini. Okolina agentu daje nagrade te putem njih oblikuje agentove odluke. Kod podržanog učenja ne zna se unaprijed točan izlazni podatak Y za neki ulazni podatak X . Izlazni podatci i vrijednosti kriterijske funkcije se spremaju tijekom rada agenta. Tek kada agent dobije nagradu može znati jesu li njegove odluke dobre.

$$Y_{pred} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \quad L = \begin{bmatrix} 5 \\ 4 \\ 2 \\ 8 \\ 10 \\ 1 \\ 5 \\ 20 \\ 5 \end{bmatrix} \Rightarrow R = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} \Rightarrow Y = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad L = \begin{bmatrix} -5 \\ -4 \\ -2 \\ 8 \\ 10 \\ 1 \\ 5 \\ -20 \\ -5 \end{bmatrix}$$

Izraz 1 - Utjecaj nagrade na kriterijsku funkciju

Y_{pred} – odluke neuronske mreže

L – vrijednosti kriterijske funkcije

R – dobivena nagrada

Y – dobivene točne odluke

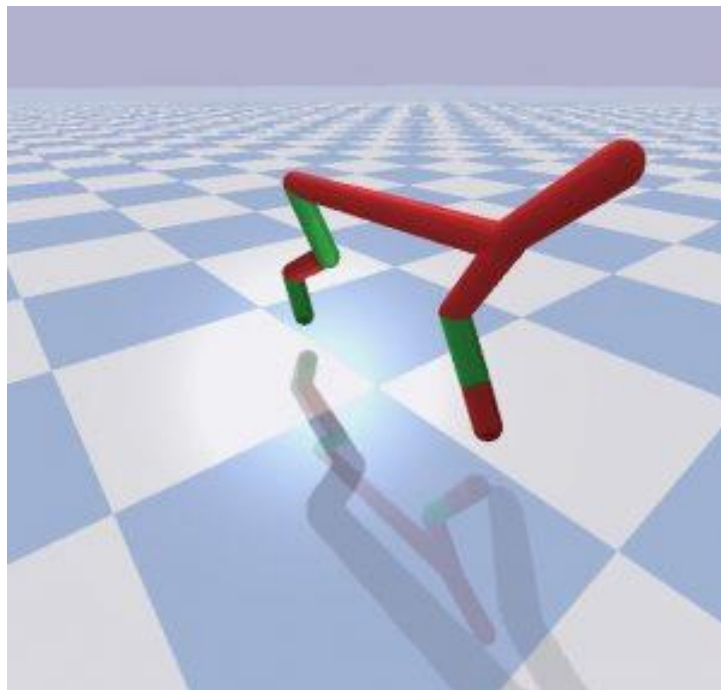
Vrijednosti Y se odnose na neuronsku mrežu s slike 1, gdje su moguće vrijednosti Y 1 ili 2. Vrijednosti kriterijske funkcije L uvijek su pozitivne prije nego li uzmemo nagradu u obzir. Pomoću negativnih nagrada saznamo koje odluke nisu točne, te njihove vrijednosti kriterijske funkcije postanu negativne. Promjene na parametrima neuronske mreže se vrše propagacijom unazad, kao i kod nadziranog učenja.

Kod podržanog učenja bitno je pravilno definirati pravila i nagrade okoline. Nepravilno definirane nagrade mogu uzrokovati sporo učenje agenta, ili rezultate koji nisu željeni

3 RJEŠENJE PROBLEMA

3.1. HalfCheetah okruženje

OpenAI Gym je alat koji sadrži gotova okruženja za agente podržanog učenja. Za potrebe ovog rada preuzeto je HalfCheetah kontinuirano okruženje. HalfCheetah okruženje sastoji se od ravne podloge i dvonogog robotsko geparda.



Slika 4 – HalfCheetah okruženje

Svaka noga geparda sadrži po 3 zgloba što je ukupno 6, tako da je moguće izvršiti 6 akcija nad ovom okolinom. Te akcije predstavljaju momente sila motora tih 6 zglobova.

Samo okruženje kao odgovor na akciju daje 26 veličina koje predstavljaju zapažanja okoline

3.2. Biblioteke

Korištene su sljedeće biblioteke:

Tensorflow-rocm

- ➔ ROCM verzija Tensorflow biblioteke za rad s AMD uređajima

Keras

- ➔ Integriran s Tensorflow-om u novijim verzijama

Keras-rl

- ➔ Sadrži gotove funkcije i klase napisane u Keras-u za rad s podržanim učenjem

Pybullet

- ➔ Besplatni omotač za Mujoco okoline

3.3. DDPG

Učenje agenta je izvedeno koristeći Deep Deterministic Policy Gradient algoritam. DDPG je namijenjen izrazito za rješavanje kontinuiranih problema. U algoritmu se koriste dvije neuronske mreže jer je on zapravo mješavina algoritama koji se svode na traženje tzv. Optimalne politike i algoritama koji se svode na traženje optimalne vrijednosti Q-funkcije.

DDPG spada u Actor-Critic metode. Što znači da jedna neuronska mreže predlaže akcije s obzirom na zapažanja okoline(Actor) dok druga mreža procjenjuje koliko su te akcije zapravo dobre ili loše(Critic). Actor kao ulaz uzima zapažanja iz okoline, a kao izlaz daje akcije. Critic uzima akcije i zapažanja i daje Q vrijednost koja predstavlja ukupnu nagradu za buduće akcije ako se iz trenutnog stanja poduzme određena akcija i do se nakon toga prati trenutna politika. Za učenje tih mreža koristi se 'minibatch', nasumično odabrani skupovi mogućih akcija i stanja iz buffer-a.

DDPG je off-policy algoritam je ne koristi istu politiku i za odabir akcija i za optimizaciju same politike

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process \mathcal{N} for action exploration
 Receive initial observation state s_1
 for $t = 1, T$ **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned}$$

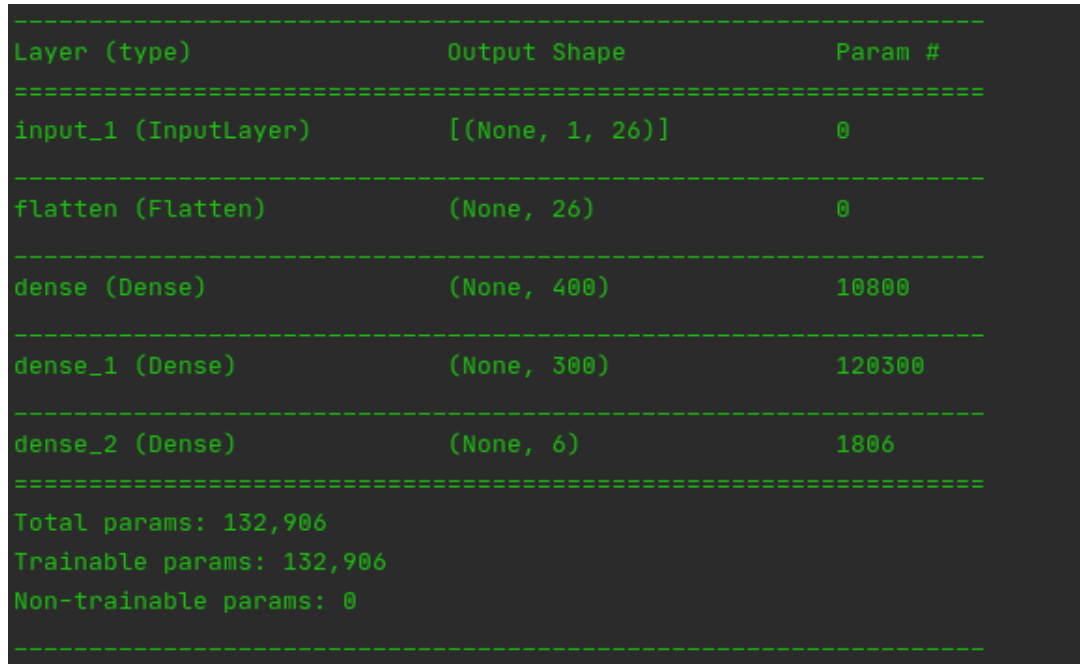
end for
end for

Slika 5 DDPG algoritam

Na slici 5 prikazan je DDPG algoritam.

3.4. Struktura neuronske mreže

Actor mreža se sastoji od ulaza 26 ulaznih neurona pošto kao ulaz prima 26 opažanja iz okoline. Zatim slijede dva skrivena *Dense* sloja sa 400 i 300 neurona s *Relu* aktivacijskom funkcijom, te izlazni sloj sa 6 neurona pošto toliko ima mogućih akcija i *Tanh* aktivacijskom funkcijom.



```
-----
Layer (type)                 Output Shape              Param #
-----
input_1 (InputLayer)         [(None, 1, 26)]          0
-----
flatten (Flatten)            (None, 26)                0
-----
dense (Dense)                 (None, 400)              10800
-----
dense_1 (Dense)               (None, 300)              120300
-----
dense_2 (Dense)               (None, 6)                1806
-----
Total params: 132,906
Trainable params: 132,906
Non-trainable params: 0
-----
```

Slika 6 Struktura Actor neuronske mreže

Critic mreža ima također ulazni sloj od 26 neurona, zatim Dense sloj od 400 neurona i *Relu* aktivacijskom funkcijom nakon čega se slijedi sloj gdje se dodaju akcije kao ulazi i spajaju se u jedan tenzor sa opažanjima okoline. Zatim slijedi još jedan Dense sloj od 300 neurona i *Relu* aktivacijskom funkcijom te izlazni sloj od 1 neurona koji predstavlja Q vrijednost aktiviran linearnom aktivacijskom funkcijom.

```

-----
Layer (type)           Output Shape          Param #             Connected to
-----
obs_input (InputLayer) [(None, 1, 26)]       0
-----
flatten_1 (Flatten)    (None, 26)            0                   obs_input[0][0]
-----
dense_3 (Dense)         (None, 400)           10800               flatten_1[0][0]
-----
action_input (InputLayer) [(None, 6)]           0
-----
concatenate (Concatenate) (None, 406)           0                   dense_3[0][0]
                                                                action_input[0][0]
-----
dense_4 (Dense)         (None, 300)           122100              concatenate[0][0]
-----
dense_5 (Dense)         (None, 1)              301                 dense_4[0][0]
=====
Total params: 133,201
Trainable params: 133,201
Non-trainable params: 0
=====

```

Slika 7 Struktura Critic neuronske mreže

3.5. Parametri

Parametri neuronskih mreža i agenta su preuzeti iz rada “Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control”¹.

```
agent = DDPGAgent(nb_actions=actions_n[0], actor=actor, critic=critic, batch_size=64, critic_action_input=action_input,
                  memory=memory, nb_steps_warmup_critic=1000, nb_steps_warmup_actor=1000,
                  random_process=random_process, gamma=.99)

agent.compile([Adam(lr=1e-4), Adam(lr=1e-3)], metrics=['mse'])
```

Slika 8 Parametri DDPG agenta

Veličina batch-a (minibatch u ovom slučaju) je parametar koji se mora prilagoditi sposobnostima računala. Isprobani su veličine 32 , 64, 128, 200. Najveće veličine su brzo konvergirale većim nagradama, ali se učestalo trening znao prekinut. Na kraju je postavljena veličina od 64.

Također je postavljen parametar koji pušta neuronske mreže da polako povećavaju stopu učenja to zadane u 1000 koraka kako bi se izbjegao rani *overfitting*. Gama parametar je jako važan u podržanom učenju, a pogotovo u kontinuiranim zadacima jer osigurava konvergiranje vrijednosti budućih nagrada ka nekom broju jer kontinuirani prostor ima neograničen broj mogućih akcija. Isto tako će više na značaju davati trenutnim nagradama , a manje onima u budućnosti.

Na objema mrežama korišten je Adam optimizator.

```
agent.fit(env, env_name=ENV_NAME, nb_steps=500000, action_repetition=5, visualize=False, verbose=1)
```

Slika 9 Parametri fit() funkcije

Agent je učen na 2500 epizoda, pri čemu je je ponavljao iste akcije 5 puta prije nego li je istraživao dalje što ubrzava proces učenja , a slabo utječe na performanse agenta.

¹ <https://arxiv.org/pdf/1708.04133.pdf>

4 REZULTATI

Učenje je provedeno na 2500 epizoda odnosno 500000 koraka. Početna nagrada je iznosila -954, a konačna 1300. Vrijednost nagrade se na početku brzo povećavala dok je kasnije sve sporije i sporije rasla.

```
Training for 500000 steps ...
Interval 1 (0 steps performed)
2020-08-29 16:44:12.983195: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcblas.so
10000/10000 [=====] - 140s 15ms/step - reward: -4.7714
50 episodes - episode_reward: -954.289 [-1750.002, 165.911] - loss: 3.217 - mse: 6.433 - mean_q: -23.502

Interval 2 (10000 steps performed)
10000/10000 [=====] - 178s 18ms/step - reward: -0.9990
50 episodes - episode_reward: -199.803 [-1034.381, 358.908] - loss: 7.349 - mse: 14.698 - mean_q: -26.438

Interval 3 (20000 steps performed)
10000/10000 [=====] - 170s 17ms/step - reward: -0.2644
50 episodes - episode_reward: -52.800 [-1120.052, 404.198] - loss: 13.913 - mse: 27.826 - mean_q: -18.320

Interval 4 (30000 steps performed)
10000/10000 [=====] - 168s 17ms/step - reward: -0.1860
50 episodes - episode_reward: -37.194 [-1139.450, 425.160] - loss: 17.623 - mse: 35.246 - mean_q: -7.582
```

Slika 10 Prvih 200 epizoda treninga

```
Interval 15 (140000 steps performed)
10000/10000 [=====] - 189s 19ms/step - reward: 5.1726
50 episodes - episode_reward: 1034.514 [673.899, 1283.113] - loss: 247.349 - mse: 494.698 - mean_q: 304.272

Interval 16 (150000 steps performed)
10000/10000 [=====] - 189s 19ms/step - reward: 5.0950
50 episodes - episode_reward: 1018.995 [7.914, 1222.569] - loss: 263.805 - mse: 527.609 - mean_q: 311.345

Interval 17 (160000 steps performed)
10000/10000 [=====] - 189s 19ms/step - reward: 5.3641
50 episodes - episode_reward: 1072.815 [679.698, 1229.916] - loss: 259.333 - mse: 518.666 - mean_q: 319.316

Interval 18 (170000 steps performed)
10000/10000 [=====] - 189s 19ms/step - reward: 5.4025
50 episodes - episode_reward: 1080.501 [729.814, 1306.441] - loss: 277.939 - mse: 555.877 - mean_q: 327.098

Interval 19 (180000 steps performed)
10000/10000 [=====] - 189s 19ms/step - reward: 5.1317
50 episodes - episode_reward: 1026.341 [741.819, 1233.052] - loss: 296.584 - mse: 593.168 - mean_q: 333.809

Interval 20 (190000 steps performed)
10000/10000 [=====] - 188s 19ms/step - reward: 5.2666
```

Slika 11 Zadnjih 300 epizoda

Intervala je zapravo bilo 50 ukupno, ali pošto je trening znao stati prije završetka, težine mreža su se spremale i učitanje su pri ponovnom pokretanju dok se nije izvelo 2500 epizoda.

```
Testing for 5 episodes ...  
Episode 1: reward: 1129.183, steps: 1000  
Episode 2: reward: 1092.632, steps: 1000  
Episode 3: reward: 1310.326, steps: 1000  
Episode 4: reward: 1086.311, steps: 1000  
Episode 5: reward: 1124.010, steps: 1000
```

Najbolji testni rezultat od 5 epizoda bio je 1310. Nagrada je konzistentna u testiranju, ali je za ovo okruženje i dalje mala. Agent se kreće prema naprijed koristeći zadnju nogu dok prednju jako malo koristi što je problem nedovoljnog broja epizoda. Agent vrlo rano nauči koristiti zadnju nogu jer mu donosi najveću nagradu dok prednju tek uči koristiti pri kraju treniranja. Mnogi radovi pokazuju da se DDPG algoritmom mogu postići bolji rezultati, ali na jako velikom broju epizoda i s većom minibatch veličinom.

5 ZAKLJUČAK

Pojačano učenje je danas vrlo popularno za učenje agenta čija se učinkovitost uspoređuje s ljudskom. Ona je često veća od ljudske. Agenti mogu jako brzo učiti, ali su naučeni samo za specifične zadatke dok se u novim okolnostima ne snalazi. Još smo daleko od nekog inteligentnog generalnog AI-a, ali nas odlično služe za specifične zadatke i zamjenjuju dosta ljudskih poslova. Samo podržano učenje je napravljeno na principu čovjeka koji uči na pokušajima i pogreškama.

DDPG metoda je jako osjetljiva na parametre stoga treba napraviti jako dobru optimizaciju parametara pri korištenju ove metode. U većini radova DDPG je postizao solidan rezultat tek nakon 10000 epizoda.

Nemogućnost korištenja većeg Minibatch-a je također utjecalo na rezultate.

Iako je DDPG noviji algoritam i ima jako dobre rezultate na ovom okruženju, stalno se pojavljuju noviji i bolji algoritmi. Podržano učenje je prilično mlado područje i treba očekivati njegov nagli rast.

Jedno od ograničenja je dakako i samo sklopovlje koje mora podržati te zahtjevne algoritme.