SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2857

Automatizirani razvoj agenata za različita okruženja

Paulo Sanković

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2857

Pristupnik:	**Paulo Sanković (0036509900)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Automatizirani razvoj agenata za različita okruženja**

Opis zadatka:

Proučiti različita okruženja poput jednostavnih igara ili fizikalnih simulatora za koje je moguće razviti agente. Proučiti različite metode i modele poput umjetnih neuronskih mreža koje se mogu iskoristiti za upravljanje agentom u takvim okruženjima. Proučiti postojeća programska rješenja koja pružaju simulaciju pojedinih okruženja te istražiti mogućnosti povezivanja s takvim okruženjima. Razviti programski okvir koji omogućuje automatski razvoj agenata za odabrana okruženja te grafički prikazuje ponašanje razvijenih agenata. Ocijeniti efikasnost razvijenih agenata za odabrana okruženja. Predložiti moguća poboljšanja predloženog postupka s ciljem razvoja efikasnijih agenata. Radu priložiti izvorne programske kodove, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 27. lipnja 2022.

Zahvaljujem se mentoru doc. dr. sc. Marku Đuraseviću na pomoći, dobrim savjetima i razumijevanju tijekom izrade ovog diplomskog rada, kao i svim profesorima i kolegama s kojima sam proveo svoje osnovnoškolske, srednjoškolske i studentske dane. Posebno bih se zahvalio svojoj obitelji na pruženoj podršci i razumijevanju.

SADRŽAJ

1. Uvod

Inteligentni sustavi sposobni su analizirati, razumjeti i učiti iz dostupnih podataka putem posebno dizajniranih algoritama umjetne inteligencije. Zahvaljujući dostupnosti velikih skupova podataka, razvoja tehnologije i napretka u algoritmima došli smo do razine gdje nam razvijeni modeli mogu uvelike poslužiti u svakodnevnom životu.

Posebno je zanimljiva problematika u kojoj je bez znanja o pravilima i funkcioniranju specifične okoline, potrebno konstruirati specijaliziranog agenta koji se nalazi u određenom stanju okoline i ponavlja korake izvršavanja optimalne akcije i prijelaza u novo stanje okoline. Za svaku akciju agent prima određenu nagradu - mjeru koja označava koliko su akcije agenta ispravne za tu okolinu i koliko je napredak agenta ispravan. Agent izvršava akcije i prelazi u nova stanja sve dok se ne nađe u terminalnom (završnom) stanju. Dakle, cilj agenta u okolini jest pronaći optimalnu strategiju koja će maksimizirati očekivanu dobit (nagradu) u određenom vremenskom okviru.

Područje strojnog učenja koje se bavi prethodno navedenom problematikom naziva se podržano učenje (engl. *Reinforcement Learning*). Agenti koji se prilično dobro ponašaju u takvim okolinama možemo implementirati pomoću različitih algoritama podržanog učenja koji se temelje na umjetnim neuronskim mrežama (engl. *Artificial Neural Networks*). Umjetne neuronske mreže su dobri aproksimatori funkcija i najbolje se ponašaju u okolini koja ima kompozitnu strukturu gdje vrlo kvalitetno duboki model predstave kao slijed naučenih nelinearnih transformacija.

U sklopu ovog rada bilo je potrebno proučiti i razumjeti metode i algoritme podržanog učenja i funkcioniranje umjetnih neuronskih mreža. Nadalje, bilo je potrebno istražiti, proučiti i naposljetku implementirati neke od algoritama podržanog učenja koji se zasnivaju na umjetnim neuronskim mrežama i koje je trebalo uklopiti u okoline koje su prikladne za simulaciju i testiranje ponašanja naučenih agenta.

Programsko rješenje implementirano je u programskom jeziku *Python*, primarno koristeći *PyTorch* biblioteku (engl. *library*) zajedno s ostalim korisnim bibliotekama poput *numpy*, *tqdm*, *stable-baselines3*... Za simulaciju i testiranje ponašanja agenata (razvijenih modela) u posebnom okruženju korištena je biblioteka *OpenAI Gym*.

2. Podržano učenje

Strojno učenje (engl. *Machine Learning*) jest grana umjetne inteligencije (engl. *Artificial Inteligence*) koja se može definirati kao skup metoda koje u podatcima mogu automatski otkrivati obrasce, i potom te otkrivene obrasce iskorištavati pri budućem predviđanju podataka, ili obavljati druge zadatke odlučivanja u prisustvu nesigurnosti Čupić (2022). Drugim riječima, bez eksplicitnog programiranja moguće je napraviti sustave koji funkcioniraju kao ljudski mozak - imaju pristup podatcima, koriste ih za učenje i samim time bolje razumiju entitete, domene i veze između podataka.

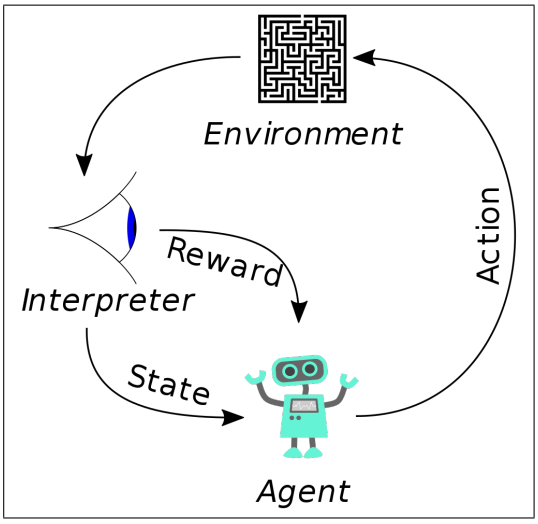
Strojno učenje dijeli se na 3 podvrste: nadzirano učenje, nenadzirano učenje i podržano (ojačano) učenje. Nadzirano učenje (engl. *supervised learning*) karakterizira učenje modela nad testnim podatcima koji su označeni. Model točno zna da za određeni ulaz mora vratiti izlaz koji je istovjetan unaprijed pridruženoj oznaci. Algoritam mjeri točnost kroz funkciju gubitka, prilagođavajući se sve dok se izračunata razlika izlaza modela i stvarnog izlaza (pogreška) ne smanji u određenoj mjeri. U nenadziranom učenju (engl. *unsupervised learning*) za razliku od nadziranog, posjedujemo podatke bez zadanog izlaza - podatci su dani bez ciljne vrijednosti i u tim situacijama treba pronaći određenu pravilnost. Postupci poput grupiranja, smanjenja dimenzionalnosti, otkrivanja veza između primjeraka... pripadaju nenadziranom učenju.

Posebna i nama najzanimljivija podvrsta strojnog učenja jest podržano učenje (engl. *reinforcement learning*). Podržano učenje bavi se optimizacijom ponašanja agenta koji je u interakciji s okolinom (u kojoj se nalazi) i koji na temelju informacija koje dobiva iz okoline izvršava akcije, i kao odgovor na svaku akciju dobiva nagradu ili kaznu. Za razliku od prethodno dvije navedene podvrste koje mapiraju ulazne podatke na određeni format izlaza, u podržanom učenju je naizraženije učenje iz iskustva koje je čovjeku kao biću ključan način na koji se razvija. Od najranije dobi, bića nastoje shvatiti i razumjeti okolinu u kojoj se nalaze na temelju niza aktivnosti kojima utječu na okolinu i opažanja kako okolina pri toj interakciji utječe na nas.

2.1. Ključni koncepti

Za potpuno razumijevanje podržanog učenja, bitno je u navesti i pojasniti glavne pojmove. Okolina (engl. *environment*) označava svijet u kojem se agent nalazi i s kojim interaktira. Stanje (engl. *state*) reprezentira presjek okoline u određenom trenutku. Agentu korisna informacija jest nagrada (engl. *reward*) koja predstavlja povratnu informaciju okoline. Način na koji agent bira akciju (engl. *action*) iz skupa svih dostupnih akcija naziva se politika (engl. *policy*).

Cilj podržanog učenja jest naći optimalnu strategiju (niz optimalnih akcija) koje maksimiziraju ukupnu (kumulativnu) nagradu. U svakom koraku interakcije agenta s okolinom, agent prima opis stanja okoline u kojoj se nalazi. S obzirom na to stanje, izvršava akciju koja vrši neku promjenu nad okolinom i prebacuje ju u novo stanje. Agent prima povratnu informaciju od okoline koja reprezentira koliko je odabrana akcija u skladu sa stanjem okoline. Opisana interakcija agenta s okolinom prikazana je na slici 2.1.



Slika 2.1: Prikaz ciklusa i interakcije agenta s okolinom

2.2. Duboki modeli

Duboko učenje (engl. *Deep learning*) jest tip strojnog učenja (točnije, podskup strojnog učenja) koje nastoji oponašati način zaključivanja i obrasce koje ljudski mozak koristi za učenje i donošenje odluka. Veliku ulogu u cijeloj ideji dubokog učenja imaju duboke neuronske mreže (engl. *Deep neural networks, DNN*) pomoću kojih se povezivanjem više slojeva procesnih elemenata (čvorova, neurona), dobivaju duboki modeli

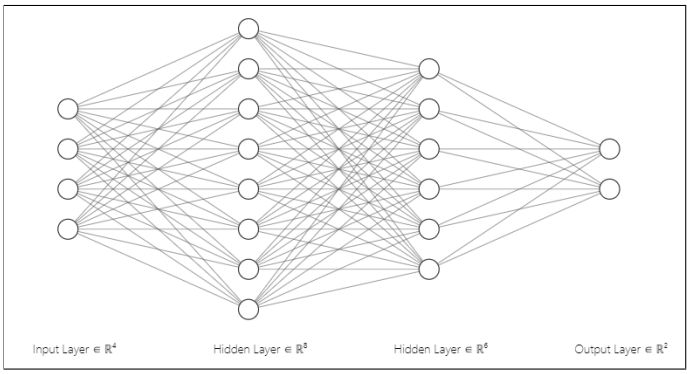
koji su sposobni učiti i baratati s podatcima kompozitne strukture. Primjenom dubokih modela dolazimo do slijeda naučenih nelinearnih transformacija kojima aproksimiramo funkciju odluke, učimo mapiranje između ulaznih podataka i izlazih podataka, te nastojimo postići dobru generalizaciju nad stvarnim podatcima.

2.2.1. Unaprijedni potpuno povezani modeli

Unaprijedni potpuno povezani modeli (engl. *Fully connected neural network*) (poznatiji i pod nazivom višeslojni perceptron (engl. *Multi-layer perceptron*)) sastoje se od lanaca potpuno povezanih slojeva. Svaki neuron iz prethodnog sloja povezan je s neuronom idućeg sloja.

Sastoje se od tri vrste slojeva - ulaznog sloja, izlaznog sloja i skrivenih slojeva. Ulaznom sloju dovode se podatci koje je potrebno obraditi. Izlaz neuronske mreže (u najosnovnijem obliku) predstavljen je logitima (engl. *logits*) - vektorima logaritama nenormaliziranih vrijednosti. Specifičnije, za slučaj da želimo provesti klasifikaciju podataka ili drugačije organizirati izlazne vrijednosti na izlaz dodajemo posebni sloj (npr. *Softmax* funkcija za klasifikaciju). Samo su ulaz i izlaz specificirani dimenzijama. Model ima slobodu da iskoristi skrivene slojeve na način koji osigurava najbolju aproksimaciju funkcije. Neuronskim mrežama želimo izgraditi modele koji nisu linearno odvojivi i zato koristimo nelinearnu aktivacijsku funkciju - najčešće ReLU (Rectified Linear Unit). Svaki od slojeva modelira jednu nelinearnu transformaciju.

Slika 2.2 prikazuje arhitekturu potpuno povezanog modela LeNail (2019) koji je sastavljen od sveukupno 4 potpuno povezana sloja - ulaznog (dimenzije 4), izlaznog (dimenzije 2) i dva skrivena sloja (svaki dimenzije 8). Kodom 1 prikazana je implementacija navedenog modela u biblioteci *PyTorch*.



Slika 2.2: Prikaz arhitekture potpuno povezanog modela

```
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(in_features=4, out_features=8, bias=True),
    nn.ReLU(),
    nn.Linear(in_features=8, out_features=6, bias=True),
    nn.ReLU(),
    nn.Linear(in_features=6, out_features=2, bias=True),
)
```

Kôd 1: Implementacija potpuno povezanog modela na slici 2.2 koristeći biblioteku *PyTorch*

2.2.2. Konvolucijski modeli

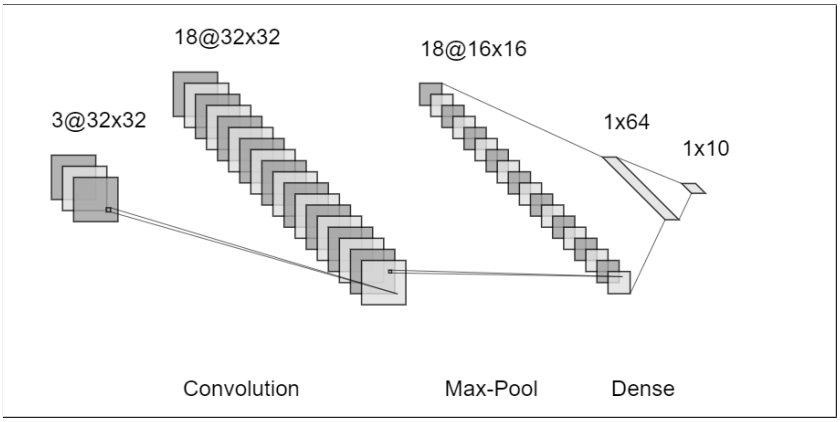
Konvolucijski modeli (engl. *Convolutional neural networks*) su modeli koji uz potpuno povezane slojeve imaju najmanje jedan konvolucijski sloj (engl. *Convolution layer*). Osim spomenutog konvolucijskog sloja i potpuno povezanog sloja, konvolucijski modeli sadrže i slojeve sažimanja (engl. *Pooling layers*), slojeve u kojima provodimo nelinearnost te sloj koji višedimenzionalni ulaz pretvara u jednodimenzionalni i pritom priprema podatke za obradu u potpuno povezanim slojevima (engl. *Flatten layer*).

Operacija konvolucije provodi se nad ulazom i jezgrom (engl. *kernel*) (slobodnim parametrima koje učimo) gdje kao rezultat dobivamo mapu značajki koja pokazuje gdje se koja značajka nalazi u ulaznim podatcima (npr. slici). Dimenzije mapa značajki i njihovu robusnost korigiramo korištenjem atributa koraka konvolucije (engl. *stride*) i nadopunjavanja ulaznih podataka (engl. *padding*). Slično konvolucijskom sloju, sloj sažimanja odgovoran je za smanjenje prostora značajki (smanjenje dimenzionalnosti podataka) i dodatno za izdvajanje dominantnih značajki. Razlikujemo dvije vrste sažimanja: sažimanje maksimalnom vrijednosti (engl. *Max pooling*) i sažimanje srednjom vrijednosti (engl. *Average pooling*). Prilikom sažimanja značajki maksimalnom vrijednošću u obzir uzimamo samo značajku najveće vrijednosti (potencijalno najvažniju značajku) te na taj način uklanjamo šum ulaza.

Korištenje konvolucijskih modela biti će nam iznimno potrebno u situacijama kada su ulazni podatci u formi slike, odnosno kada su nam važne lokalne interakcije između ulaznih podataka (piksela) te njihova vremenska i prostorna ovisnost.

Slika 2.3 prikazuje jednostavni konvolucijski model koji se sastoji od konvolucijskog sloja, sloja sažimanja maksimalnom vrijednosti, sloja koji 3-dimenzionalne po-

datke pretvara u 1-dimenzionalne, te dva potpuno povezana sloja. Ulaz u konvolucijski sloj predstavlja RGB slika (3 kanala) dimenzije 32×32. Primjenom konvolucije (veličina jezgre 3, korak 3, nadopuna 1) izvlačimo 18 kanala značajki dimenzije 32×32 (dimenzija se nije promijenila iz razloga što nadopunjavamo ulaz). Primjenom sažimanja maksimumom (veličina jezgre 2, korak 2) smanjujemo broj značajki na dimenziju 16×16. Prvi potpuno povezani sloj na svoj ulaz dobije vektor dimenzije 4608 kojeg pretvara u vektor izlaza dimenzije 64. Posljednji potpuno povezani sloj koji je ujedno i posljednji sloj u ovom konvolucijskom modelu za izlaz predaje vektor dimenzije 10. Isječak koda koji prikazuje implementaciju jednostavnog konvolucijskog modela koristeći bibilioteku *PyTorch* prikazan je kodom 2.



Slika 2.3: Prikaz arhitekture konvolucijskog modela

```
import torch.nn as nn

model = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=18, kernel_size=(3, 3),
    ↪  stride=(1, 1), padding=1),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(in_features=18 * 16 * 16, out_features=64),
    nn.Linear(in_features=64, out_features=10)
)
```

Kôd 2: Implementacija konvolucijskog modela na slici 2.3 koristeći biblioteku *PyTorch*

2.3. Algoritmi podržanog učenja

Onaj dio s towards science oko explorationa...

Svaki model strojnog učenja definiran modelom, gubitkom i metodom optimizacije. Model jest postupak obrade (odnosno skup funkcija) sa slobodnim parametrima koji za određen ulaz daje pripadajući izlaz. Gubitak jest mjera koja na formaliziran način vrednuje slobodne parametre modela, odnosno pokazuje u kojoj mjeri se mi ne slažemo s onim što je model predstavio kao izlaz. Metoda optimizacije (optimizacijski postupak) jest način na koji pronalazimo optimalne parametre koji su važno kako bi minimizirali prethodno navedenu komponentu - gubitak. Navedene tri glavne komponente biti će važno napomenuti pri svakom predstavljaju algoritma jer su to glavne odrednice pri analizi algoritama strojnog učenja.

2.3.1. Deep Q Learning

2.3.2. Double Deep Q Learning

2.3.3. Actor Critic

3. OpenAI Gym

OpenAI Gym jest Python biblioteka (engl. *library*) otvorenog koda (engl. *open source*) koja služi za razvijanje i usporedbu agenata u odabranim okolinama. Iznimno je popularna u sferi simpatizera i programera koji se bave razvijanjem modela podržanog učenja zbog jednostavnosti korištenja, velikog broja dostupnih okolina i jednostavnog stvaranja novih okolina, te jednostavne interakcije agenta i okoline. OpenAI Gym biblioteka se redovito održava i trenutno je na verziji `0.24.1`.

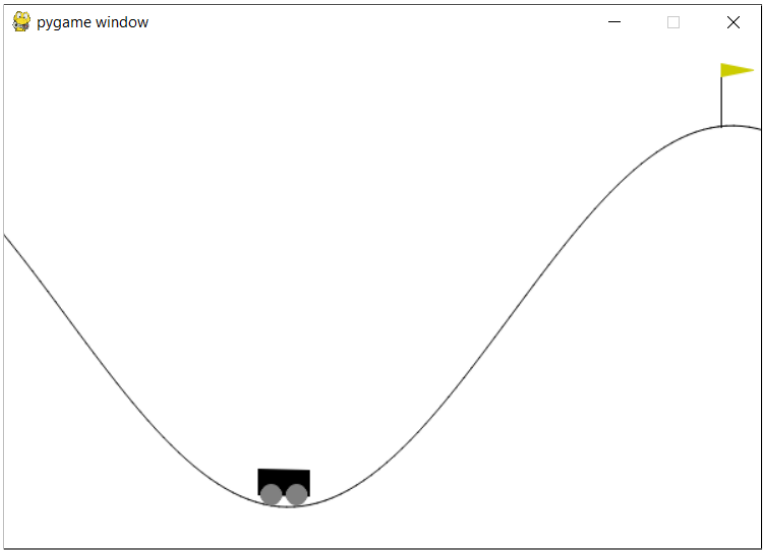
3.1. Struktura

Interakcija agenta i okoline podijeljena je na epizode. Na početku svake epizode, početno stanje se nasumično uzorkuje iz distribucije, i interakcija se nastavlja sve dok se okolina ne nađe u terminalnom stanju Brockman et al. (2016).

```
import gym

env = gym.make("MountainCar-v0")

observation = env.reset()

done = False
while not done:
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)

    env.render()

env.close()
```

Kôd 3: Jednostavan primjer integracije agenta i Gym okoline (1 epizoda)

Prikazan je potpuno funkcionalni kod 3 koji reprezentira jednostavnu interakciju agenta i okoline. Agent u ovom jednostavnom slučaju nasumično odabere akciju iz skupa svih dostupnih akcija za tu okolinu (linija 9). Osnovni kostur se sastoji od koraka specifikacije okoline (linija 3), inicijalizacije okoline (linija 5) te interakcija okoline i agenta - agent predaje okolini odabranu akciju, okolina vraća povratnu informaciju (linije 7 - 14).



Slika 3.1: Rezultat pokretanja koda 3

3.1.1. Okolina

Temelj oko kojeg se zasniva OpenAI Gym biblioteka jest razred (engl. *class*) `Env` koji u suštini implementira simulator koji pokreće okruženje u kojem naš agent može interaktirati s okolinom. Točnije rečeno, enkapsulira sva potrebna ponašanja i metode koje su potrebne za jednostavnu interakciju. Objekt tipa `Env` stvara pozivanje funkcije `gym.make(id: str, **kwargs)` (kod 3 linija 3) kojoj se predaje identifikator okoline (`id`) zajedno s opcionalnim argumentima (metapodacima).

3.1.2. Interakcija s okolinom

Kao što je vidljivo iz koda 3 osnovne metode koje se pozivaju nad instancom razreda `Env` su `reset` i `step`. Funkcija `reset` postavlja okruženje u početno stanje i vraća njegovu vrijednost (kod 3. linija 5). S druge strane, funkciji `step` (kod 3. linija 10) predaje se jedna od ispravnih akcija, akcija inicira prijelaz okoline iz jednog stanja u drugo. Funkcija vraća 4 vrijednosti: vrijednost prostora stanja (engl. *observation*),

iznos nagrade (engl. *reward*) kao rezultat poduzimanja određene akcije, zastavicu koja signalizira jesmo li došli u završno stanje okoline, te neke dodatne informacije.

Još jedna vrlo često korištena funkcija jest `render` (kod 3. linija 12) koja služi kako bi se u određenom formatu prikazala okolina. Dostupni formati su: `human` (otvara se skočni prozor sa slikom stanja okoline - slika 3.1), `rgb_array` (`numpy` array RGB vrijednosti) i `ansi` (string reprezentacija okoline).

3.1.3. Prostor akcija i prostor stanja

Osnovna struktura okruženja opisana je atributima `observation_space` i `action_space` koji su dio razreda `Env` i čija se vrijednost može razlikovati zavisno o okolini. Atribut `action_space` opisuje numeričku strukturu svih legitimnih akcija koje se mogu izvesti nad određenom okolinom. S druge strane, atribut `observation_space` definira strukturu objekta koje predstavlja stanje u kojem se okolina nalazi.

Format validnih akcija i stanja okoline, odnosno struktura tih podataka, definirana je razredima `Box`, `Discrete`, `MultiBinary` i `MultiDiscrete`. Svi navedeni razredi nasljeđuju i implementiraju glavne metode nadrazreda `Space`.

Razred `Box` predstavlja strukturu podataka u kontinuiranom `n`-dimenzionalnom prostoru. Prostor i njegove validne vrijednosti omeđene su gornjim i donjim granicama koje se jednostavno postave pri inicijalizaciji strukture pridruživanjem željenih vrijednosti atributima `high` i `low`. Kod 4 prikazuje inicijalizaciju `Box` strukture podataka koja je sastavljena od `3`-dimenzionalnog vektora čije su vrijednosti omeđene odozdo i odozgo vrijednostima `-1` i `-2`. Metoda `sample(self)` nasumično uzorkuje element iz prostora koristeći različite distribucije ovisno o ograničenjima prostora.

```
1 >>> import numpy as np
2 >>> from gym.spaces import Box
3 >>> b = Box(low=-1.0, high=2.0, shape=(3,), dtype=np.float32)
4 >>> b.sample()
5 array([-0.3791686 , -0.35007873,  0.8138365 ], dtype=float32)
```

Kôd 4: Primjer korištenja strukture kontinuiranog prostora `Box`

Razred `Discrete` s druge strane, predstavlja strukturu podataka u diskretnom `n`-dimenzionalnom prostoru gdje su validne vrijednosti sve cjelobrojne vrijednosti unutar intervala $[0, n-1]$ (početna vrijednost se može specificirati). Kod 5 prikazuje inicijalizacije `Discrete` strukture podataka ovisno o specificiranoj početnoj vrijednosti.

```
>>> from gym.spaces import Discrete
>>> d = Discrete(3)              # {0, 1, 2}
>>> d = Discrete(3, start=-1)    # {-1, 0, 1}
>>> d.sample()
1
```

Kôd 5: Primjer korištenja strukture diskretnog prostora `Discrete`

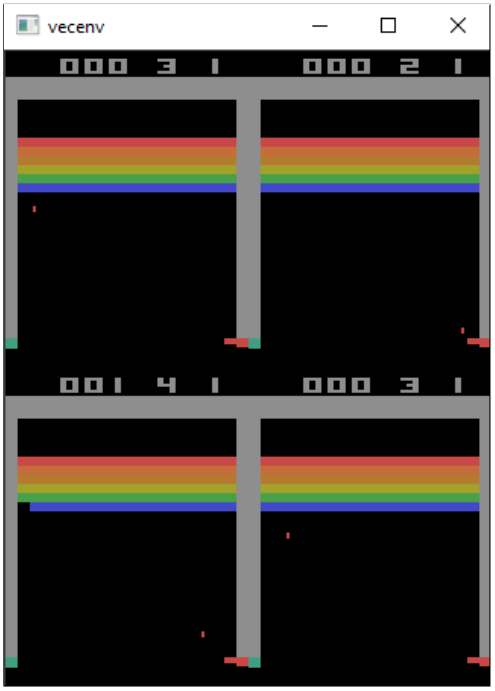
3.1.4. Omotači

Omotači (engl. *Wrappers*) su prikladne strukture koje omogućavaju izmjenu elemenata postojećeg okruženja bez potrebe za mijenjanjem originalnog koda. Omotači omogućavaju modularnost, mogu se implementirati prema vlastitim potrebama i ulančavati. Ova funkcionalnost vrlo se često koristi u situacijama kada pri treniranju modela želimo normalizirati ulaze (skalirati piksele slike), provesti regularizaciju (podrezivanje vrijednosti nagrade), transformirati ulaze u Pytorch dimenzije, implementirati preskakanje slikovnih okvira... Navedene funkcionalnosti moguće je postići tako da definiramo vlastiti omotač koji će nasljeđivati ili obični `Wrapper` nadrazred ili specifičnije razrede poput `ObservationWrapper`, `RewardWrapper`, `ActionWrapper`...

3.1.5. Vektorizirana okruženja

Koristeći standardne metode stvaranja i interakcije s Gym okruženjem, pokrećemo samo jednu instancu okruženja i na taj način ne iskorištavamo potpuno računalnu snagu koja nam je dostupna. Vektorizirana okruženja (engl. *Vectorized environments*) su okruženja koja paralelno pokreću više kopija istog okruženja u svrhu poboljšanja učinkovitosti i ubrzanja procesa učenja agenta.

?? TODO nastaviti ovo jer su sada objavili novu verziju u kojoj je već u Gym-u ukomponiran Vector API a ja sam do sada bio koristio njihov library *baselines*. ?? - ponovna implementacija i usporedba



Slika 3.2: Prikaz vektoriziranog okruženja sa 4 paralelne asinkrone instance

3.2. Okruženja

U OpenAI Gym ekosustavu dostupno je puno okruženja koja omogućuju interakciju s agentom. Neki od njih su: *Atari* - skup od Atari 2600 okolina, *MuJoCo* (punim nazivom *Multi-Joint dynamics with Contact*) - skup okolina za provođenje istraživanja i razvoja u robotici, biomehanici, grafici i drugim područjima gdje je potrebna brzina i točna simulacija, *Classic Control* - skup okolina koje opisuju poznate fizikalne eksperimente. Opisat ćemo neke od najkorištenijih okolina.

Upravljanje agentima i njihovo izvođenje akcija u određenoj atari okolini interno je implementirano koristeći objektno orijentiranu razvojnu cjelinu (engl. *framework*) *The Arcade Learning Environment* (skraćeno *ALE*). Na taj način razdvajaju se slojevi emulacije okruženja (koristeći Atari 2600 emulator *Stella*), sloj koji kontrolira samog agenta *(ALE)* i sloj koji pruža jednostavnu programsku implementaciju i interakciju *(OpenAI Gym)*Bellemare et al. (2013).

3.2.1. Okruženje CartPole

Ovo okruženje modelira fizikalni problem održavanja ravnoteže. Inačica je sličnog fizikalnog problema pod nazivom *obrnuto njihalo* (engl. *Inverted pendulum*). Za pomična kolica zakvačen je stupić. Njegovo težište nalazi se iznad središta mase i na taj

način osigurava da sustav nije stabilan. Zglob, odnosno dodirna točka između stupića i kolica nema trenja niti drugih gubitaka. Također, kolica koja se kreću vodoravno po putanji u 2 smjera nemaju trenja niti drugih gubitaka. Cilj ovog fizikalnog problema jest uravnotežiti stup primjenom sila i pomicanjem kolica u lijevom ili desnom smjeru.

Za svaki poduzeti korak okolina dodjeljuje nagradu u vrijednosti $+1$. Struktura valjanih akcija koje agent može poduzeti (`action_space`) instanca je razreda `Discrete(2)` - skup akcija je diskretan i u svakom koraku je moguće odabrati 1 od maksimalno 2 dostupne akcije. Opis značenja svake akcije prikazan je u tablici 3.1. S druge strane, objekt koji predstavlja strukturu stanja okoline u određenom vremenskom trenutku (`observation_space`) instanca je razreda `Box(4)` – stanje se sastoji od 4 kontinuirane vrijednosti od kojih su neke ograničene i odozdo i odozgo. Točan opis i granice vrijednosti predočeni su u tablici 3.2.

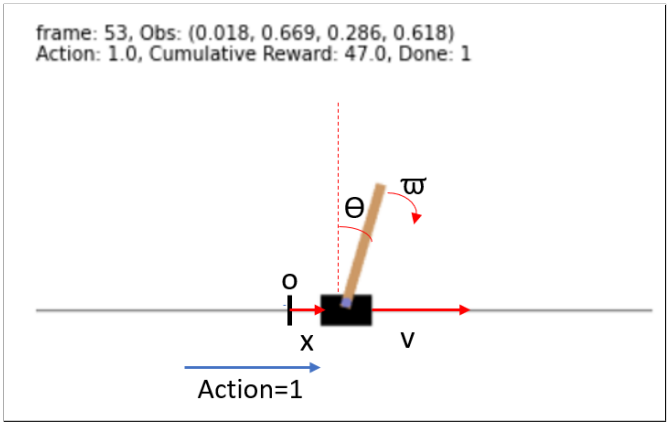
Tablica 3.1: Opis valjanih akcija okoline CartPole - atribut `action_space`

Akcija	Opis akcije
0	Pomak kolica ulijevo
1	Pomak kolica udesno

Tablica 3.2: Opis strukture okoline okoline CartPole - atribut `observation_space`

Indeks	Opis	Donja granica	Gornja granica
0	Pozicija kolica	-4.8	4.8
1	Brzina kolica	$-\infty$	∞
2	Nagib štapića i kolica	-0.418rad	0.418rad
3	Brzina štapića na vrhu	$-\infty$	∞

Početno stanje okoline inicijalizira pozivom metode `reset()`, slučajnim vrijednostima iz uniformne razdiobe na intervalu ± 0.05. Okolina podržava 3 uvjeta zaustavljanja (uvjeti koji označuju da je riječ o terminalnom stanju): nagib štapića i kolica je izvan intervala ± 0.2095rad, pozicija sredine kolica je izvan intervala ± 2.4 (sredina kolica dotiče rub vidljivog prostora) i duljina epizode je veća od 500 koraka. Na slici 3.3 prikazana je okolina CartPole zajedno s vrijednostima okoline.



Slika 3.3: Prikaz CartPole okoline zajedno s vrijednostima okoline - TODO

3.2.2. Okruženje Breakout

Ova okolina simulira poznatu Atari 2600 igru u kojoj je cilj sakupiti što više bodova pomičući platformu i održavajući lopticu na ekranu. Platforma je postavljena na dnu ekrana, na fiksnoj visini i moguće ju je pomicati u dva smjera. Loptica se odbija između zidova, platforme i 6 razina *ciglenih* blokova čijim razbijanjem se sakupljaju bodovi. Ako loptica padne ispod platforme koju igrač kontrolira, gubi se život. Igra završava kada igrač potroši 5 života, odnosno kada 5 puta loptica padne ispod platforme.

Skup validnih akcija je instanca razreda `Discrete(4)` - skup akcija je diskretan i u svakom koraku je moguće odabrati 1 od maksimalno 4 dostupne akcije. Opis značenja svake akcije prikazan je u tablici 3.3. Kao opis trenutnog stanja okoline moguće je dobiti RGB vrijednosti svakog piksela slike (slike dimenzije 210 * 160) ili vrijednosti radne memorije *ALE* okoline (128 bajta) - što je korisno jer možemo preskočiti korak učenja reprezentacije okoline (preskačemo dio gdje algoritmi učenja moraju iz piksela slike naučiti reprezentaciju). Razlike u strukturi objekta okoline prikazane su u tablici 3.4.

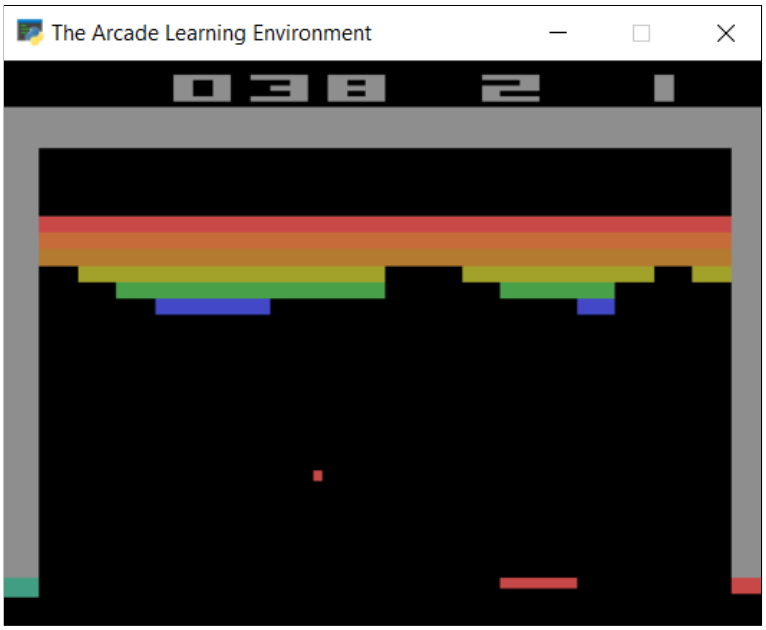
Tablica 3.3: Opis valjanih akcija okoline Breakout - atribut `action_space`

Akcija	Opis akcije	Detaljniji opis akcije
0	NOOP	Ne poduzima se nikakva akcija
1	FIRE	Akcija koja pokreće igru
2	RIGHT	Platforma se pomiče udesno
3	LEFT	Platforma se pomiče ulijevo

Tablica 3.4: Opis strukture objekta okoline Breakout - atribut `observation_space`

Indeks	Struktura
RAM vrijednosti	`Box(0, 255, (128,), uint8)`
Vrijednosti RGB slike	`Box(0, 255, (210, 160, 3), uint8)`

Nagrada dolazi u obliku bodova koji se dobivaju uništavajući *ciglene* blokove. Vrijednost nagrade ovisi o boji cigle. Izgled same okoline prikazan je na slici 3.4.



Slika 3.4: Prikaz Breakout okoline

4. Implementacija

4.1. Okolina CartPole

4.1.1. Implementacija Deep Q Learning algoritma

4.1.2. Implementacija Double Deep Q Learning algoritma

4.1.3. Implementacija Actor Critic algoritma

4.1.4. Usporedba

4.2. Okolina Breakout

4.2.1. Implementacija Deep Q Learning algoritma

4.2.2. Implementacija Double Deep Q Learning algoritma

4.2.3. Implementacija Actor Critic algoritma

4.2.4. Usporedba

4.3. Usporedba algoritama

5. Moguća poboljšanja

6. Zaključak

Bože pomozi hehe.

LITERATURA

M. G. Bellemare, Y. Naddaf, J. Veness, i M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, i Wojciech Zaremba. Openai gym, 2016.

Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019. doi: 10.21105/joss.00747. URL `https://doi.org/10.21105/joss.00747`.

Marko Čupić. *Umjetna Inteligencija - Uvod u strojno učenje*. 2022. http://java.zemris.fer.hr/nastava/ui/ml/ml-20220430.pdf.

Automatizirani razvoj agenata za različita okruženja

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.