Podržano učenje i Q-učenje

Blažević, Lucija

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Lucija Blažević

Podržano učenje i Q-učenje

Diplomski rad

Osijek, 2020.

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Lucija Blažević

Podržano učenje i Q-učenje

Diplomski rad

Mentor: izv. prof. dr. sc. Domagoj Matijević

Osijek, 2020.

Sadržaj

1 Uvod **1**

2 Strojno učenje **2**
2.1 Nadzirano učenje 3
2.2 Nenadzirano učenje 3
2.3 Podržano učenje 4

3 Podržano učenje **5**
3.1 Razlika podržanog učenja i ostalih vrsta strojnog učenja 5
3.1.1 Podržano učenje i nadzirano učenje 5
3.1.2 Podržano učenje i nenadzirano učenje 6
3.2 Karakteristike podržanog učenja 6
3.3 Osnovni pojmovi podržanog učenja 7
3.3.1 Tipičan scenarij podržanog učenja 7
3.3.2 Svojstva okoline 10
3.3.3 Komponente agenta 11

4 Algoritmi podržanog učenja **16**
4.1 Metode temeljene na vrijednosti 16
4.2 Metode temeljene na strategiji 16
4.2.1 Metode temeljene na strategiji i metode bez strategije 16
4.3 Metode temeljene na modelu 17
4.3.1 Metode temeljene na modelu i metode bez modela 17

5 Markovljev proces odlučivanja **18**

6 Q-učenje **21**
6.1 Osnove Q-učenja 21
6.2 Praktični primjer 23
6.2.1 Opis problema 23
6.2.2 Primjena Q-učenja 27
6.2.3 Usporedba rezultata 31

Zaključak **34**

Literatura **35**

Prilog 1 **37**

Prilog 2 **41**

Sažetak **42**

Summary **43**

Životopis **44**

1 Uvod

Tehnologija raste i razvija se iz dana u dan. Sukladno tome, razvija se i umjetna inteligencija. Ako se samo malo osvrnemo na svijet koji nas okružuje, možemo vidjeti kako se to područje računalne znanosti danas razvilo do tih granica da oko sebe imamo stvari za koje prije nismo mogli niti zamisliti da će postojati. Počevši od pametnih mobitela i uređaja koji prepoznaju naš govor pa sve do robota i raznih sustava preporuka koji nam nude baš ono što mi želimo. Razvoj umjetne inteligencije uvelike nam je olakšao našu svakodnevnicu pružajući nam pomoć u određenim radnjama.

Jedna grana umjetne inteligencije, koja je posljednjih godina dosta aktualna, naziva se strojno učenje. Strojno učenje je grana umjetne inteligencije koja se bavi pronalaženjem algoritama koji će računalima dati sposobnost samostalnog učenja bez eksplicitnog programiranja. Razvoj strojnog učenja započeo je prije 50-ak godina, no u zadnjih nekoliko godina doživljava veliki uspon jer uspjeva rješavati određene probleme bolje i od čovjeka. Samo neki od problema kojima se strojno učenje bavi su otkrivanje bolesti raka, omogućavanje automatizirane vožnje, pomoć u kućanskim poslovima, komunikacija, briga o zdravlju i slično.

U ovome radu posebno ćemo se posvetiti proučavanju jednog tipa strojnog učenja koji nazivamo podržano učenje. Prije nego krenemo sa samim proučavanjem podržanog učenja, u prvom poglavlju imat ćemo kratak uvod u strojno učenje kako bi razumjeli osnovne koncepte strojnog učenja te uvidjeli po čemu se podržano učenje razlikuje od ostalih učenja strojnog učenja. Nakon toga, detaljnije ćemo se upoznati s podržanim učenjem te vidjeti što to podržano učenje jest, koje su njegove komponente, gdje se ono koristi, na koji način i slično. Nakon što proučimo podržano učenje, osvrnut ćemo se na jedan algoritam podržanog učenja koji nazivamo Q-učenje te na praktičnom primjeru vidjeti kako ga možemo koristiti.

2 Strojno učenje

Strojno učenje je grana umjetne inteligencije koja proučava algoritme i metode koje koriste računalni sustavi za samostalno obavljanje određenog zadatka. Postoje dvije definicije strojnog učenja:

Definicija 1. (Arthur Samuel, 1959.) *Strojno učenje je polje proučavanja koje računalima daje mogućnost samostalnog učenja bez eksplicitnog programiranja.*

Ovo je starija, neformalna definicija, dok je nešto novija i modernija sljedeća definicija:

Definicija 2. (Tom M. Mitchell, 1997.) *Računalni program uči iz iskustva E s obzirom na neku klasu zadataka T i mjeru uspješnosti P, ako se njegove performanse na zadatcima T, mjerene s P, poboljšavaju s iskustvom E.*

Primjer 1. *Igranje šaha*

E = iskustvo igranja mnogo partija šaha
T = zadatak igranja šaha
P = vjerojatnost da će program dobiti sljedeću igru

Algoritmi strojnog učenja grade matematički model temeljen na uzorcima podataka, poznatim kao "podatci za treniranje" (eng. training data), kako bi se donijele prognoze ili odluke bez da su eksplicitno programirani za izvršavanje zadatka. Strojno učenje uglavnom je usmjereno na razvoj računalnih programa koji sami sebe mogu naučiti kako rasti i mijenjati se ako su izloženi novim podatcima.

Podatci rastu iz dana u dan i nemoguće je većom brzinom obraditi sve podatke. Većina podataka je nestrukturirano. Podatci su masivni te se samim time povećava vrijeme potrebno za njihovu obradu. Upravo u ovim situacijama pomaže nam strojno učenje, kako bi se velika količina podataka obradila u minimalnom vremenu. Danas je strojno učenje prisutno u toliko mnogo segmenata tehnologije da to uopće ni ne shvaćamo sve dok ga ne počnemo koristiti.

Strojno učenje podijeljeno je u tri kategorije, a to su:

- Nadzirano učenje
- Nenadzirano učenje
- Podržano učenje

U nastavku ćemo ukratko opisati svaku od navedene tri kategorije te se nakon toga osvrnuti isključivo na proučavanje treće kategorije, tj. podržanog učenja.

2.1 Nadzirano učenje

Nadzirano učenje vrsta je strojnog učenja u kojem se algoritmi treniraju korištenjem obilježenih podataka. To znači da nam je za ulaz dan skup podataka za koji nam je već poznato kako bi trebao izgledati ispravan izlaz. Algoritam uči na način da uspoređuje predikcije modela s ispravnim izlazima kako bi pronašao pogreške te nakon toga, u skladu s tim, modificira model.
Podatke obično dijelimo na podatke za treniranje na kojima model uči i testne podatke koji nam služe za testiranje koliko je model dobro naučio.
Problemi nadziranog učenja kategorizirani su u regresijske i klasifikacijske probleme.

2.2 Nenadzirano učenje

Kod nenadziranog učenja, stvar je malo drugačija nego kod nadziranog učenja. Za treniranje algoritma koriste se neobilježeni podatci. Cilj ovog učenja je istražiti podatke i pronaći neku strukturu unutar njih. Neobilježeni podatci dani su kao unos algoritmu, nakon čega ih algoritam pokušava shvatiti i grupirati.
Još jedna razlika u odnosu na nadzirano učenje jest to što podatke ne možemo podijeliti na podatke za treniranje i testne podatke, što je i logično s obzirom da su oni neobilježeni pa nemamo povratne informacije, tj. nemamo ispravne izlaze s kojima bi usporedili rezultate dobivene od strane modela.
Nenadzirano učenje uglavnom je podijeljeno na dva dijela: grupiranje (klasteriranje) i smanjenje dimenzija (eng. Dimensionality Reduction).

2.3 Podržano učenje

Podržano učenje je treća vrsta strojnog učenja u kojem algoritam sam mora shvatiti situaciju u kojoj se nalazi. Podržano učenje često se primjenjuje u robotici, igrama i navigaciji. U podržanom učenju algoritam putem pokušaja i pogreške otkriva koje akcije (ili niz akcija) daju najznačajnije nagrade. Ova vrsta treniranja ima tri glavne komponente, a to su: agent (može se opisati kao učenik ili donositelj odluka), okolina (koja je opisana kao sve s čime agent ima interakciju) i akcije (radnje koje predstavljaju što sve agent može raditi).

Cilj ove vrste učenja jest da agent poduzme akcije koje će maksimizirati očekivanu nagradu tijekom određenog vremena. Agent će dostići cilj mnogo brže i bolje ako pronađe strategiju koja će mu pomoći pri odabiru akcija pa stoga možemo reći da je cilj podržanog učenja pronaći optimalnu strategiju koja će maksimizirati očekivanu dobit.

U nastavku ovoga rada, detaljnije ćemo se pozabaviti ovom vrstom strojnog učenja te vidjeti koliko je korisno podržano učenje u današnjem svijetu.

3 Podržano učenje

U ovom poglavlju detaljnije ćemo definirati što je to podržano učenje te po čemu se ono razlikuje od ostalih učenja strojnog učenja. Nakon toga, još ćemo jednom definirati ključne pojmove povezane s podržanim učenjem i potkrijepiti ih s primjerima. Posebno ćemo se dotaknuti agenata i okoline jer nam oni čine najbitniju sastavnicu za proučavanje podržanog učenja u računalnom svijetu.

3.1 Razlika podržanog učenja i ostalih vrsta strojnog učenja

Postoje neke važne značajke podržanog učenja koje ga razlikuju od nadziranog i nenadziranog učenja. Posebno ćemo opisati razlike između podržanog učenja i nadziranog učenja te između podržanog učenja i nenadziranog učenja.

3.1.1 Podržano učenje i nadzirano učenje

Spomenuli smo u prvom poglavlju, kako u nadziranom učenju za ulaz imamo obilježene podatke, tj. za svaki ulazni podatak poznato nam je koji je ispravan izlaz. To znači da za svaki ulazni podatak, algoritam nadziranog učenja donosi odluku neovisno o ostalim ulaznim podacima. Dakle, odluke u nadziranom učenju su nezavisne i donose se na podacima koji su nam dani odmah na početku. Kod podržanog učenja stvar je malo drugačija. Odluke zavise jedne o drugima i donose se sekvencijalno. U takvim problemima izvedljivije je učiti iz vlastitih iskustava i stjecati znanja iz njih. To je glavna razlika između podržanog i nadziranog učenja. U podržanom učenju oslanjamo se na metodu pokušaja i pogreške. Jedina povratna informacija koju imamo su nagrade koje dobijemo za svaki korak koji napravimo. Također, bitno je napomenuti kako povratne informacije u podržanom učenju mogu biti odgođene, tj. možda tek nakon dugo vremena uočimo kako je neka prijašnja akcija koju smo poduzeli bila dobra ili loša.
Možemo uzeti za primjer igru šaha gdje, u svakom stanju u kojem se nalazimo, postoji mnogo poteza koje možemo odigrati. Jedan krivi potez može nas dovesti do gubitka mnogo poteza kasnije. Šah nam daje pravi primjer sekvencijalnih odluka i odgođenih nagrada (ili kazni).

3.1.2 Podržano učenje i nenadzirano učenje

U podržanom učenju agent putem pokušaja i pogreške pokušava naučiti što poduzeti u situaciji u kojoj se nalazi. U nenadziranom učenju, glavni zadatak je pronaći neku strukturu podataka i grupirati podatke. Najbolje ćemo ovu razliku shvatiti na jednom primjeru. Na primjer, ako imamo za zadatak predložiti korisniku članak o vijestima, algoritam nenadziranog učenja, potražit će članke slične onima koje je korisnik prethodno čitao, dok će algoritam podržanog učenja, sugerirati razne članke korisniku, dobiti povratne informacije od korisnika, te na temelju toga, polako "naučiti" koji će se članci toj osobi svidjeti.

3.2 Karakteristike podržanog učenja

U prethodnom odjeljku vidjeli smo što razlikuje podržano učenje od nadziranog i nenadziranog učenja. Već pri opisivanju razlika, mogli smo uočiti neke bitne stavke podržanog učenja. Nabrojimo sada sve te karakteristike podržanog učenja:

- U podržanom učenju, niti u jednom trenutku, nitko nam ne kaže koju akciju trebamo poduzeti, već se oslanjamo na metodu pokušaja i pogreške. Jedina povratna informacija koju imamo su nagrade koje dobijemo za svaki korak koji napravimo.

- Povratne informacije su odgođene. Ne dobijemo odmah povratnu informaciju jesmo li odabrali dobru akciju, ona može doći i nakon mnogo koraka kasnije.

- Za postizanje cilja potrebno je sekvencijalno donošenje odluka pa vrijeme igra veliku ulogu u problemima podržanog učenja.

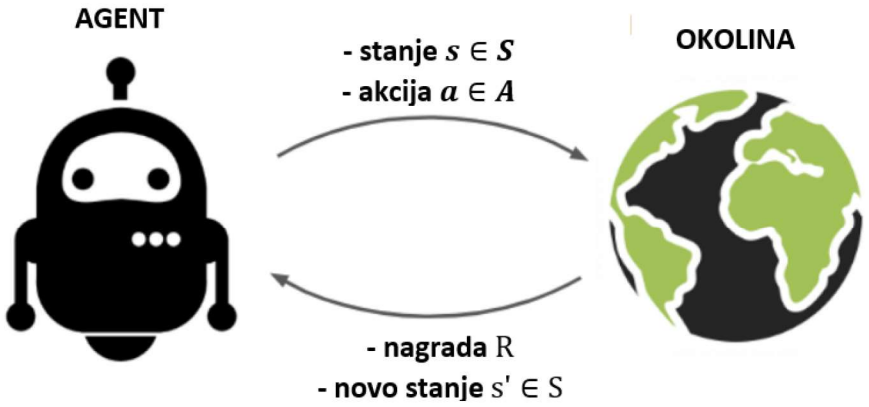
- Agentove akcije utječu na naknadne podatke koje će primiti, tj. različite akcije mogu ga dovesti u različita stanja pa se samim time mijenja i okruženje u kojem se našao.

3.3 Osnovni pojmovi podržanog učenja

Posvetimo se sada isključivo podržanom učenju te proučimo glavne termine koje ćemo koristiti.

3.3.1 Tipičan scenarij podržanog učenja

Kao što smo već ukratko vidjeli u uvodnom poglavlju, podržano učenje je pristup kroz koji inteligentni programi, poznati kao agenti, rade u poznatoj ili nepoznatoj okolini kako bi se konstantno prilagođavali i učili na temelju dobivenih bodova. Pogledajmo na sljedećoj slici jedan tipičan scenarij podržanog učenja.



Slika 1: Scenarij podržanog učenja

Sljedeći pojmovi sadržani su u svakom scenariju podržanog učenja:

- AGENT: subjekt koji obavlja akcije u okolini kako bi dobio neku nagradu
- OKOLINA (e): scenarij s kojim se agent mora suočiti/scenarij u kojem se agent nalazi (sve što agent ne može proizvoljno izmijeniti smatra se dijelom okoline)
- STANJE (s): trenutni položaj u okolini na kojem se agent nalazi
- AKCIJA (a): radnja koju agent poduzima kako bi prešao u sljedeće stanje

- NAGRADA (R): povratna informacija dana agentu nakon što obavi određenu akciju

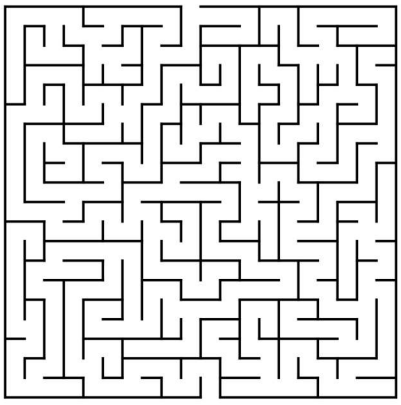
U scenariju podržanog učenja događa se sljedeće: u nekom trenutku t, agent se nalazi u stanju $s \in S$ dobivenom od strane okoline. U tom stanju, agent poduzima akciju $a \in A$ te prelazi u novo stanje $s' \in S$ za koje dobiva nagradu R.

Koristeći podržano učenje, učimo kako preslikati situacije u akcije kako bi maksimizirali konačnu nagradu. Agentu nije rečeno koju akciju treba poduzeti, već sam mora otkriti koje akcije donose najveće nagrade. U većini slučajeva, akcije, ne samo da utječu na trenutnu nagradu, već i na sljedeću situaciju u kojoj se agent nađe te sve naknadne nagrade koje dobije.

Primjenimo sada ove pojmove na sljedeća dva primjera, kako bi nam bilo lakše shvatiti koncept podržanog učenja.

Primjer 2. *Primjer labirinta*

Labirint nam može poslužiti kao dobar primjer na koji možemo primjeniti podržano učenje. Cilj je odrediti ispravne poteze kako bi dovršili labirint i došli do kraja labirinta što je brže moguće.



Slika 2: Labirint

Odredimo sada spomenute pojmove podržanog učenja:

- Agent je subjekt koji se kreće kroz labirint
- Okolina je labirint
- Stanje je mjesto u labirintu na kojem se agent nalazi
- Akcija je potez koji moramo poduzeti za prelazak u sljedeće stanje
- Nagrada su bodovi povezani s dostizanjem određenog stanja

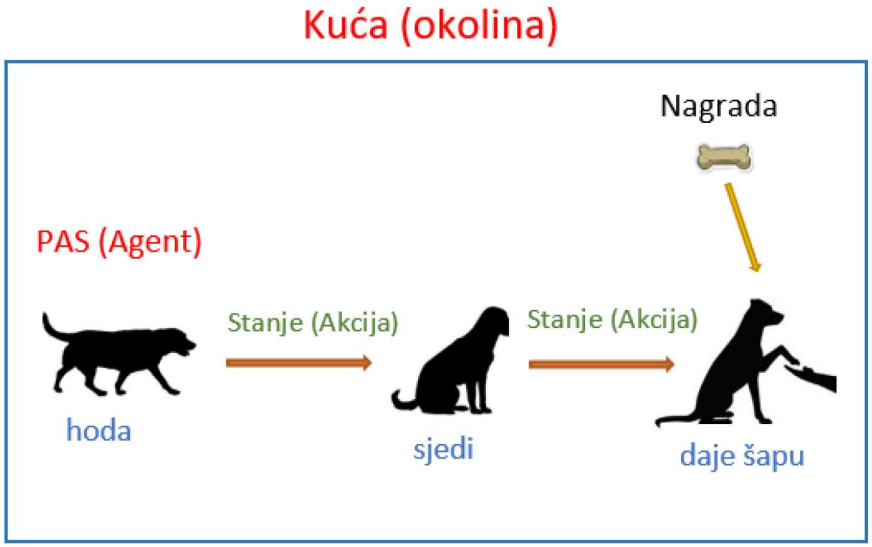
Vrlo vjerojatno se većina nas našla u situaciji "prakticiranja" podržanog učenja, makar možda toga nismo ni svjesni. Sljedeći primjer iz stvarnog života, također nam pomaže na jednostavan način približiti koncept podržanog učenja.

Primjer 3. *Podučavanje psa*

Zamislimo da smo u situaciji gdje želimo naučiti psa da nam da šapu. Kako pas ne razumije ljudski jezik, ne možemo mu izravno reći što da radi. Umjesto toga, slijedimo drugačiju strategiju. Cilj nam je naučiti psa da željeno reagira na našu zapovijed. Ako pas da šapu kada mu to kažemo, dat ćemo mu poslasticu. Sada kada je pas izložen istoj situaciji, on poduzima sličnu akciju u očekivanju da dobije još poslastica. Na taj način, koristeći pozitivne nagrade, pas uči što treba raditi.

Odredimo i na ovom primjeru osnovne pojmove podržanog učenja.

- Agent je pas
- Okolina je npr. naša kuća
- Stanje je položaj u kojem se pas nalazi, npr. sjedi, hoda, daje šapu
- Akcija je prelazak iz jednog stanja u drugo, npr. pas prelazi iz hodajućeg stanja u sjedeće stanje
- Nagrada je u ovom slučaju poslastica, ukoliko pas napravi trik koji želimo, odnosno da nam šapu



Slika 3: Pas i podržano učenje

Nakon što smo se na primjerima malo bolje upoznali s osnovnim scenarijem podržanog učenja, posvetimo se sada okolini i agentu.

3.3.2 Svojstva okoline

Okoline u podržanom učenju imaju određena svojstva koja utječu na agenta podržanog učenja. Pošto se agent mora prilagoditi i ponašati u skladu s okolinom, svojstva okoline su nam dosta važna.
Ovo su neke osnovne značajke okoline:

- **diskretna ili neprekidna**
  Ako postoji ograničen broj različitih, jasno definiranih stanja okoline, okolina je diskretna, inače je neprekidna. Primjer diskretne okoline bila bi igra šaha, a neprekidne vožnja automobila.

- **potpuno ili djelomično vidljiva (uočljiva)**
  Ukoliko je moguće odrediti sva stanja okoline, u svakom trenutku, gdje god se nalazili, kažemo da je okolina potpuno vidljiva ili vidljiva. U suprotnom je djelomično vidljiva. Ako je okolina potpuno vidljiva, imamo savršen scenarij za implementaciju podržanog učenja.

Primjer potpune vidljivosti je igra šaha. Primjer djelomične vidljivosti je igra pokera, gdje su neke od karata nepoznate igračima.

- **statička ili dinamička**
  Ako se okolina ne mijenja dok agent djeluje svojim akcijama, tada je ona statička, u suprotnom je dinamička.

- **deterministička ili nedeterministička (stohastička)**
  Ako je sljedeće stanje okoline u potpunosti određeno trenutnim stanjem i agentovim akcijama, tada je okolina deterministička, inače je nedeterministička odnosno stohastička. Drugim riječima, okolina je deterministička ako točno znamo koje je sljedeće stanje u koje ćemo doći, s obzirom na akciju koju smo poduzeli, inače je stohastička.

- **jednoagentna ili višeagentna**
  Okolina može sadržavati jednog ili više agenata.

3.3.3 Komponente agenta

Postoje tri komponente agenta podržanog učenja:

- **Strategija** (π)**:** definira agentovo ponašanje
- **Funkcija vrijednosti** (V **ili** Q)**:** definira koliko je dobro biti u određenom stanju i koliko je dobro odabrati određenu akciju
- **Model:** opisuje kako agent vidi okolinu

Agent podržanog učenja može sadržavati jednu ili više navedenih komponenti. U nastavku ćemo ukratko objasniti svaku od njih.

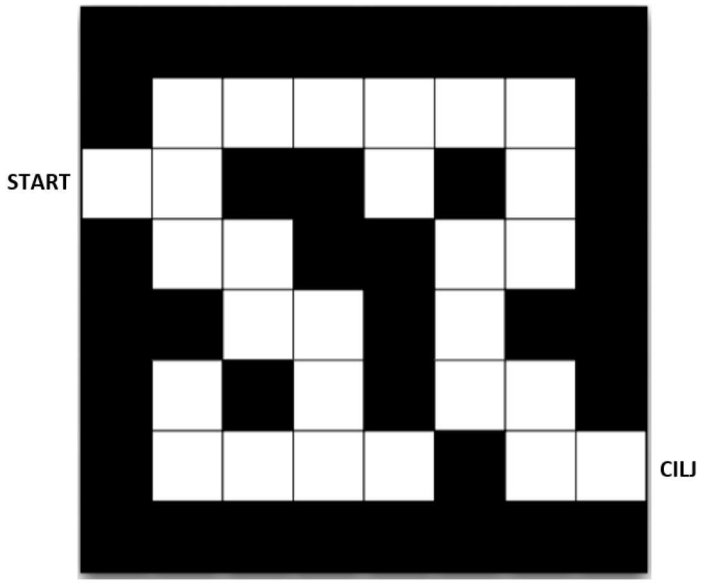
STRATEGIJA
Strategija je funkcija koja opisuje ponašanje agenta podržanog učenja. Strategiju agent primjenjuje kako bi odlučio koju akciju poduzeti u odnosu na trenutno stanje u kojem se nalazi.
Strategija može biti:

- Deterministička: $a = \pi(s)$
  Ukoliko u svakom trenutku, za svako stanje u kojem se nalazimo, znamo koju akciju trebamo poduzeti, strategija je deterministička. Dakle, deterministička strategija je funkcija koja nam za određeno stanje, kao rezultat, daje akciju koju trebamo poduzeti u tom stanju.

- Stohastička: $\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$
  Stohastička strategija definirana je kao vjerojatnost poduzimanja akcije a s obzirom na trenutno stanje s. Ponekad je ova strategija bitna kako bi malo istražili područje i možda proširili prostor stanja.

Strategiju želimo naučiti iz iskustva i želimo da ona bude takva da nas "odvede" do najveće ukupne nagrade.

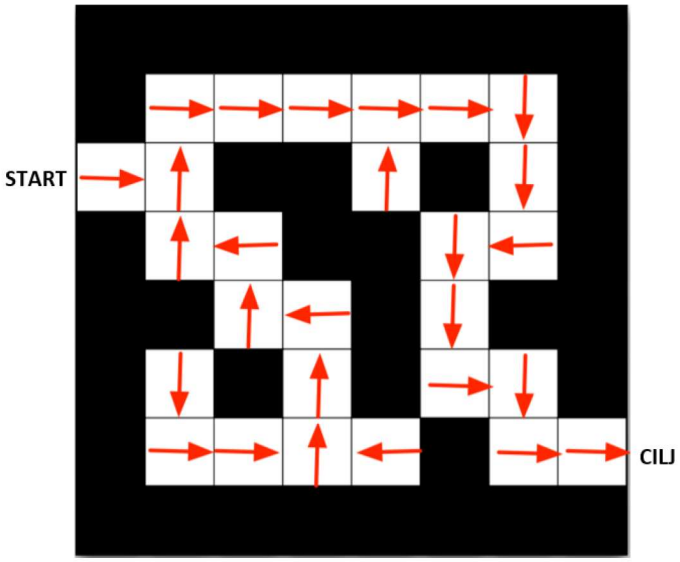
Pogledajmo sljedeći primjer labirinta koji će nam poslužiti kako bi na tom primjeru prikazali kako izgleda strategija te kasnije funkcija vrijednosti i model i lakše shvatili razlike među tim agentovim komponentama.



Slika 4: Primjer labirinta

Na ovoj slici, neka su nam nagrade -1 za svako stanje, odnosno svaki korak koji napravimo. Stanje, u ovom primjeru, je agentova lokacija u labirintu. Akcije koje imamo na izbor su lijevo, desno, gore i dolje.

Pogledajmo na sljedećoj slici kako bi na ovom primjeru označili strategiju.



Slika 5: Strategija na primjeru labirinta

Za svako stanje (položaj u labirintu), imamo na izbor akcije lijevo, desno, gore, dolje. Strelice predstavljaju strategiju $\pi(s)$ koja nam za svako stanje s kaže koju akciju trebamo poduzeti kako bi došli do cilja s najvećom ukupnom nagradom koju možemo zaraditi.

FUNKCIJA VRIJEDNOSTI

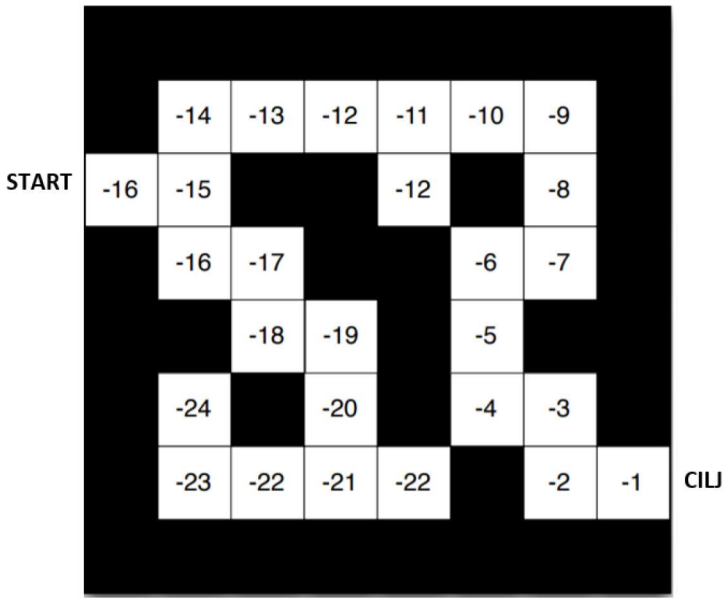
Funkcija vrijednosti definira koliko je dobro biti u određenom stanju ili koliko je dobro odabrati određenu akciju u nekom stanju. Ona nam daje očekivanu dugoročnu nagradu, za razliku od same nagrade R koja nam daje nagradu samo za trenutno stanje s u koje dođemo. Razlikujemo dvije funkcije vrijednosti: **funkciju vrijednosti stanja** $V_\pi(s)$ te **funkciju vrijednosti akcije** $Q_\pi(s, a)$. Ovdje ćemo, kada govorimo

o funkciji vrijednosti, prvenstveno misliti o funkciji vrijednosti stanja $V_\pi(s)$, dok ćemo funkciju vrijednosti akcije, $Q_\pi(s, a)$, spominjati nešto kasnije u ovome radu.
Funkciju vrijednosti, $V_\pi(s)$, koristimo kako bi usporedili stanja, odnosno procjenili koje stanje je bolje. Funkcija vrijednosti definira se kao očekivana ukupna nagrada s obzirom na trenutno stanje s i trenutnu strategiju π, tj.

$$V_\pi(s) = \mathbb{E}_\pi[\, R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \,]$$

U prethodnoj formuli γ, $0 < \gamma < 1$, je faktor umanjenja kojeg koristimo kako bi umanjili buduće nagrade. Time veću važnost dajemo trenutnim nagradama negu budućim, što i ima smisla s obzirom na to da je budućnost neizvjesna i nema smisla davati jednaku važnost budućim nagradama za koje, ukoliko je strategija stohastička, nismo niti sigurni hoćemo li ih dobiti.

Pogledajmo ponovno na primjeru labirinta kako bi izgledala funkcija vrijednosti:



Slika 6: Funkcija vrijednosti na primjeru labirinta

Brojevi u labirintu predstavljaju iznos funkcije vrijednosti $V_\pi(s)$ za svako stanje s.

MODEL

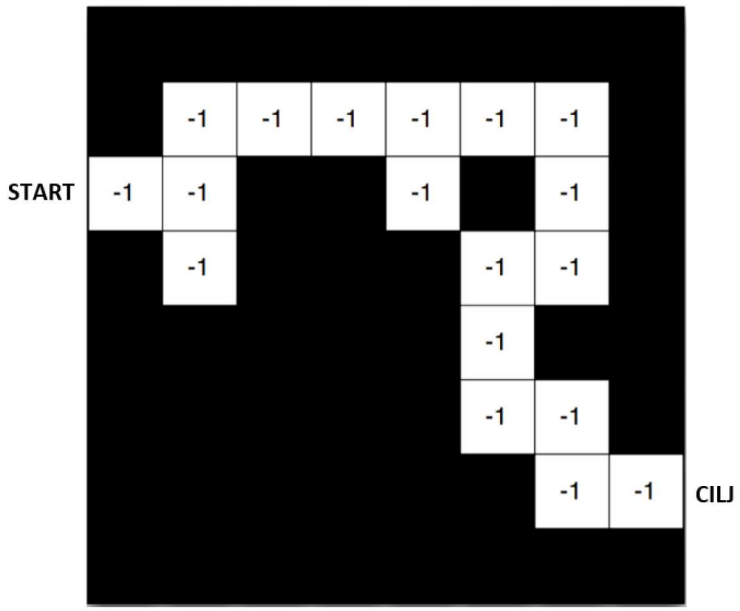
Model je agentova reprezentacija okoline, tj. opis kako agent "vidi" okolinu u kojoj se nalazi.

Postoje dva dijela modela:

- Model prijelaza: $\mathcal{P}^a_{ss'} = \mathbb{P}\,[\,S_{t+1} = s' \mid S_t = s, A_t = a\,]$
  Model prijelaza predviđa sljedeće stanje okoline, odnosno govori nam kolika je vjerojatnost da ćemo doći u sljedeće stanje s' ako se trenutno nalazimo u stanju s i poduzmemo akciju a.

- Model nagrađivanja: $\mathcal{R}^a_{ss'} = \mathbb{E}\,[\,R_{t+1} \mid S_t = s, S_{t+1} = s', A_t = a\,]$
  Model nagrađivanja predviđa nagradu koju osigurava okolina. Točnije, daje nam očekivanu nagradu s obzirom da se nalazimo u stanju s i poduzmemo akciju a.

Kada razmotrimo ova dva dijela modela, općenito možemo reći kako model predviđa što će nam sljedeće okolina dati ako poduzmemo određenu akciju u određenom stanju.



Slika 7: Model na primjeru labirinta

U primjeru labirinta, izgled mreže predstavlja model prijelaza, odnosno kako smo se kretali kroz labirint od starta do cilja. Brojevi u labirintu predstavljaju trenutnu na-

gradu koju smo dobili za dolazak u pojedino stanje (kako smo definirali na početku, nagrada je -1 za svako stanje).

4 Algoritmi podržanog učenja

S obzirom na prethodno nevedene komponenete agenta podržanog učenja, postoje različiti algoritmi podržanog učenja koje možemo odabrati.
Algoritam koji ćemo odabrati ovisi o našem željenom pristupu rješavanja tog problema. Želimo li izravno saznati agentovu strategiju ili želimo naučiti funkciju vrijednosti? Želimo li raditi bez modela ili po uzoru na model?
Postoje tri pristupa za implementaciju algoritma podržanog učenja: metode temeljene na vrijednosti (eng. Value-Based), temeljene na strategiji (eng. Policy-Based) i temeljene na modelu (eng. Model-Based).

4.1 Metode temeljene na vrijednosti

U metodi podržanog učenja temeljenoj na vrijednosti, pokušavamo maksimizirati funkciju vrijednosti $V(s)$. U ovoj metodi agent pokušava pronaći vrijednosti stanja u skladu sa strategijom π. Drugim riječima, proučavamo koliko je dobro biti u određenom stanju s obzirom na danu strategiju.
Kod metoda temeljenih na vrijednosti, implicira se da imamo strategiju. Ukoliko strategija nije posebno zadana, strategija koja se koristi jest odaberi najveću vrijednost. Ovakvu strategiju nazivamo pohlepnom strategijom (eng. greedy policy).

4.2 Metode temeljene na strategiji

U metodi podržanog učenja koja se temelji na strategiji, pokušavamo pronaći takvu strategiju da za akciju koju poduzmemo u pojedinom stanju dobijemo maksimalnu nagradu u budućnosti.

4.2.1 Metode temeljene na strategiji i metode bez strategije

Svaki algoritam podržanog učenja mora slijediti određenu strategiju kako bi odlučili koju akciju izvršiti u pojedinom stanju. Ipak, algoritam, tijekom učenja, ne mora

uzeti u obzir tu strategiju. Upravo to je razlika između metoda temeljenih na strategiji i onih bez strategije. Metode bez strategije ne znače da strategija ne postoji, već znači da je algoritam tijekom učenja na neki način ignorira i donosi odluke pohlepno, koristeći pohlepnu strategiju.

Dobro poznati algoritam bez strategije je Q-učenje jer njegovo pravilo ažuriranja koristi akciju koja će dati najvišu Q-vrijednost, dok bi možda stvarna strategija odabrala neku drugu akciju. Ovoj vrsti učenja posvetit ćemo se u nastavku ovog rada gdje ćemo je proučiti te vidjeti na primjeru kako se koristi.
Varijacija Q-učenja koja se temelji na strategiji poznata je pod nazivom SARSA, gdje pravilo ažuriranja koristi akciju koju je odabrala dana strategija.

4.3 Metode temeljene na modelu

U metodi podržanog učenja koja se temelji na modelu, cilj nam je stvoriti model pomoću kojeg ćemo opisati kako izgleda i kako funkcionira okolina u kojoj se agent nalazi.

4.3.1 Metode temeljene na modelu i metode bez modela

Model nam daje dinamiku okoline u kojoj se nalazimo. Nakon što sagradimo model, tražimo strategiju ili funkciju vrijednosti. Uvijek nam nešto od toga treba kako bi donijeli odluku koju akciju poduzeti.
Problem kod algoritama temeljenih na modelu jest taj što oni postaju vrlo nepraktični kako prostor stanja i prostor akcija raste.

S druge strane, kod algoritama bez modela, nema potrebe da pokušavamo opisati okolinu i razumjeti je. Umjesto toga, odmah prelazimo na pronalaženje strategije ili funkcije vrijednosti. Metodom pokušaja i pogreške skupljamo iskustva i gradimo znanje kako se ponašati, bez da smo napravili korak gdje pokušavamo shvatiti kako okolina funkcionira. Kao rezultat, nije potreban prostor za pohranu cijelog prostora stanja i akcija.

5 Markovljev proces odlučivanja

Nakon što smo se upoznali s osnovama podržanog učenja, postavlja se pitanje kako matematički formulirati problem podržanog učenja.
Upravo ovdje, dolazi nam u pomoć Markovljev proces odlučivanja koji formalno opisuje okolinu podržanog učenja. Važno je za napomenuti da se gotovo svaki problem podržanog učenja može formalizirati kao Markovljev proces odlučivanja.

Za početak, dat ćemo definiciju Markovljevog procesa odlučivanja pa ćemo objasniti svaku od navedenih sastavnica koje ćemo spomenuti.

Markovljev proces odlučivanja je petorka $(S, A, \mathcal{P}, \mathcal{R}, \gamma)$ gdje je:

- S skup stanja
- A skup akcija koje agent može poduzeti
- $\mathcal{P}$ matrica prijelaza, $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a \right]$
- $\mathcal{R}$ funkcija nagrade, $\mathcal{R}^a_{ss'} = \mathbb{E}\left[R_{t+1} \mid S_t = s, S_{t+1} = s', A_t = a \right]$
- γ faktor umanjenja

U Markovljevom procesu odlučivanja, S je skup svih stanja u okolini u kojoj se nalazimo (npr. sve pozicije na koje možemo doći i slično).

A je skup akcija koje agent može poduzeti u određenom stanju.

Matrica prijelaza definira sva pravila okoline u kojoj se nalazimo. Ono što njome dobijemo jest vjerojatnost da ćemo završiti u stanju s' s obzirom na to da smo bili u stanju s i poduzeli akciju a.
Matrica prijelaza je dosta bitna jer nam govori s kojom vjerojatnošću će se nešto dogoditi ako poduzmemo određenu akciju u određenom stanju.

R je funkcija nagrade koja nam daje vrijednost koju dobijemo za to što smo došli u određeno stanje.

γ je, već ranije spomenuti, faktor umanjenja kojeg koristimo kako bi s vremenom umanjili važnost budućih nagrada.

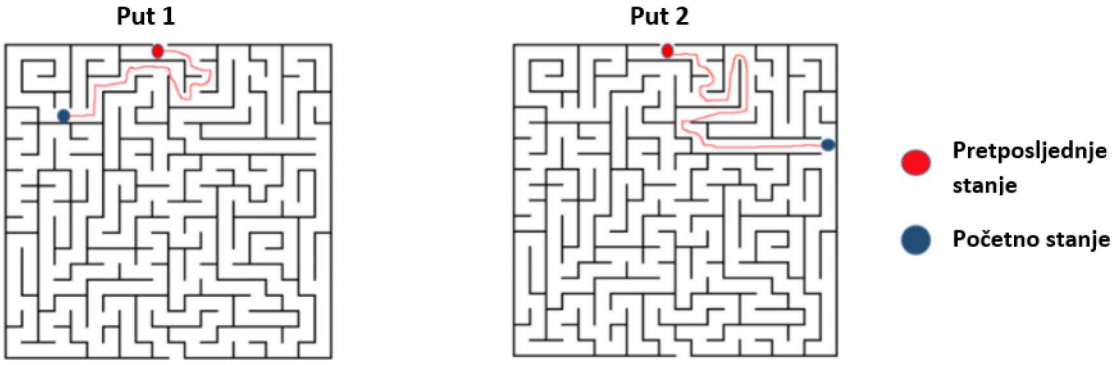
Postoje dvije stvari koje su posebno bitne kada govorimo o Markovljevom procesu odlučivanja. Prva stvar je Markovljevo svojstvo.
Markovljevo svojstvo nam govori kako je samo sadašnjost bitna. Dakle, vjerojatnost da završimo u nekom stanju s', s obzirom da se nalazimo u stanju s i poduzmemo akciju a, ovisi samo o trenutnom stanju s. Trenutno stanje s pamti sve bitne informacije koje trebamo znati iz prošlosti.
Postoji i sljedeća definicija Markovljevog stanja:

Definicija 3. *Stanje S_t je Markovljevo ako je*

$$P\left[\, S_{t+1} \mid S_t \,\right] = P\left[\, S_{t+1} \mid S_1, ..., S_t \,\right]$$

Svako stanje u Markovljevom procesu odlučivanja je Markovljevo, odnosno zadovoljava Markovljevo svojstvo.

Pogledajmo na sljedećem primjeru intuiciju koja se krije iza Markovljevog svojstva:



Slika 8: Markovljevo svojstvo

Pretpostavimo da imamo dva različita polazišta označena plavom bojom od kojih trebamo doći do istog stanja koje je označeno crvenom bojom. Primjetimo kako je crveno označeno stanje direktno pred izlazom iz labirinta. Dakle, kada dođemo u crveno stanje treba nam samo jedna akcija kako bi izašli iz labirinta. Na ovaj način

želimo pokazati kako zapravo nije bitno kojim putem smo došli do crvenog stanja, odnosno kroz koja sve stanja smo do tada prošli. Potrebne su nam samo informacije o crvenom/pretposljednjem stanju kako bi odabrali sljedeću najbolju akciju, odnosno izašli iz labirinta. Kojim putem smo došli do tog stanja, nije nam važno, a to je točno ono što Markovljevo svojstvo podrazumijeva.

Druga stvar koja je bitna kod Markovljevog procesa odlučivanja jest ta što su u Markovljevom procesu odlučivanja pravila **stacionarna**, odnosno ne mijenjaju se tijekom vremena. Prisjetimo se, sva pravila okoline u kojoj se nalazimo definirana su matricom prijelaza.

Ova petorka $(S, A, \mathcal{P}, \mathcal{R}, \gamma)$, zajedno s prethodno dva navedena svojstva, definira Markovljev proces odlučivanja. Kako smo rekli na početku ovog poglavlja, Markovljev proces odlučivanja je matematička formulacija problema podržanog učenja. Rješenje tog problema jest strategija, odnosno funkcija koja nam govori koju akciju trebamo poduzeti s obzirom na stanje u kojem se nalazimo.
Strategija može biti puno, a cilj je pronaći optimalnu strategiju kako bi maksimizirali ukupnu konačnu nagradu. No kako možemo između dvije strategije odlučiti koja strategija je bolja? Kažemo da je strategija π bolja od strategije π', ako je funkcija vrijednosti s obzirom na strategiju π za svako stanje s veća ili jednaka od funkcije vrijednosti s obzirom na strategiju π' za svako stanje s, odnosno $\pi \geq \pi' \;\; if \;\; v_\pi(s) \geq v_{\pi'}(s), \forall s$.
Prema tome, optimalna strategija je ona strategija koja rezultira optimalnom funkcijom vrijednosti, odnosno $v_*(s) = \max_\pi v_\pi(s)$.
Imajte na umu da u Markovljevom procesu odlučivanja može postojati više od jedne optimalne strategije, ali sve optimalne strategije postižu istu optimalnu funkciju vrijednosti.

U sljedećem, posljednjem poglavlju ovoga rada, posvetit ćemo se proučavanju jednog jednostavnog algoritma podržanog učenja koji se zove Q-učenje. Nakon što pokrijemo teorijske osnove ovog algoritma, na praktičnom primjeru vidjet ćemo kako ga možemo koristiti u praksi.

6 Q-učenje

6.1 Osnove Q-učenja

Q-učenje je algoritam podržanog učenja bez modela i bez strategije, odnosno algoritam podržanog učenja koji se temelji na vrijednosti. To znači da agent ne mora znati ili imati model okoline u kojoj se nalazi te agent neće slijediti danu strategiju, već će pokušati maksimizirati konačnu nagradu koristeći se pohlepnom strategijom, odnosno, u svakom stanju u kojem se nađe, agent će odabrati akciju koja će imati najveću vrijednost određenu funkcijom vrijednosti.

Još jedna stvar koju je bitno za napomenuti je to da u Q-učenju, funkcija vrijednosti koju maksimiziramo jest Q-vrijednost.
Prije nego na praktičnom primjeru vidimo kako koristiti Q-učenje u praksi, objasnimo što je to Q-vrijednost te pogledajmo neke osnovne formule i ostale pojmove vezane uz Q-učenje.

Recimo da nam je poznata nagrada koju možemo dobiti za svaku akciju a u svakom stanju s. U tom slučaju, agent želi izvršiti niz akcija koji će mu na kraju donijeti maksimalnu ukupnu nagradu. Ova ukupna nagrada naziva se još i Q-vrijednost, a formula je sljedeća:

$$Q(s, a) = R_{t+1} + \gamma \max_a Q(s', a)$$

Q-vrijednost je funkcija vrijednosti akcije koju smo spomenuli ranije u radu. Razlika između funkcije vrijednosti stanja, $V_\pi(s)$ ili kraće $V(s)$, i funkcije vrijednosti akcije $Q(s, a)$ je u tome što $Q(s, a)$ predstavlja očekivanu nagradu koju agent treba primiti ukoliko poduzme akciju a u stanju s, dok vrijednost $V(s)$ predstavlja očekivanu nagradu koju agent treba primiti s obzirom na stanje s.

Dakle, Q-vrijednost $Q(s, a)$ daje nam predviđanje ukupne nagrade, gdje, pri odabiru akcije a, koristimo pohlepnu strategiju. Slično kao i do sada, γ predstavlja faktor umanjenja koji kontrolira doprinos nagrada u budućnosti.
Nadalje, kako je formula koju smo prethodno iznijeli rekurzivna, $Q(s', a)$ ovisit će o $Q(s'', a)$ koji će tada imati koeficijent γ^2 pa zatim $Q(s'', a)$ o $Q(s''', a)$ i tako dalje.

Dakle, Q-vrijednost ovisi o Q-vrijednostima budućih stanja, tj.:

$$Q(s,a) \rightarrow \gamma Q(s',a) + \gamma^2 Q(s'',a) + \cdots + \gamma^n Q(s^n,a)$$

Napomenimo kako vrijednost γ biramo sami, tako da će podešavanje te vrijednosti utjecati na algoritam i smanjiti ili povećati doprinos budućim nagradama.

Budući da je ovo rekurzivna jednadžba, početne Q-vrijednosti biramo proizvoljno (uglavnom im početne vrijednosti postavimo oko 0, kao u većini algoritama strojnog učenja). S vremenom, kako agent bude stjecao iskustva i ažurirao Q-vrijednosti, one će se približavati optimalnim vrijednostima. U praksi, to ažuriranje provodimo na sljedeći način:

$$Q(s,a) \leftarrow Q(s,a) + \alpha^1 \left[R_{t+1} + \gamma \max_a Q(s',a) - Q(s,a) \right]$$

odnosno

$$Q^{\text{new}}(s,a) \leftarrow (1-\alpha) \underbrace{Q(s,a)}_{\textit{old value}} + \underbrace{\alpha}_{\textit{learning rate}} \cdot \overbrace{\left(\underbrace{R_{t+1}}_{\textit{reward}} + \underbrace{\gamma}_{\textit{discount factor}} \cdot \underbrace{\max_a Q(s',a)}_{\textit{estimate of optimal future value}} \right)}^{\textit{learned value}}$$

Kako bi stvorili proces iz kojega će agent učiti, potrebno je formulirati Q-tablicu. Q-tablica sadrži sve Q-vrijednosti, odnosno Q-vrijednosti za sve kombinacije stanja i akcija. Sve odluke koje agent donosi, donosi ih na temelju te tablice. Primjer jedne jednostavne Q-tablice možemo vidjeti na sljedećoj slici.

[1] α je stopa učenja koja određuje u kojoj mjeri novostečene informacije nadmašuju stare informacije. $\alpha = 0$ - agent ne nauči ništa (iskorištava isključivo prethodna znanja), $\alpha = 1$ - agent razmatra samo najnovije informacije (zanemaruje prethodno znanje)

Q(s, a)	Akcija (a)			
Stanje (s)	0	1	2	...
0	0.21772	0.09723	0.03119	0.04508
1	0.00168	0.01841	0.09021	0.03007
2	0.09021	0.03901	0.08233	0.06260
3	0.07745	0.06769	0.09672	0.09747
4	0.01272	0.05147	0.09482	0.08170
...	0.02254	0.02488	0.08215	0.03041

Slika 9: Primjer Q-tablice

6.2 Praktični primjer

Nakon što smo se upoznali s osnovama Q-učenja, bitno je da pogledamo i jedan praktičan primjer kako bi nam te osnove bile jasnije. Primjer ćemo proučavati u programskom jeziku Pythonu, točnije u Jupyter notebook-u, zajedno s pripadnim tekstom. Kako bi prikazali primjer koji ćemo proučavati, koristit ćemo gym library, alat za razvijanje algoritama podržanog učenja.

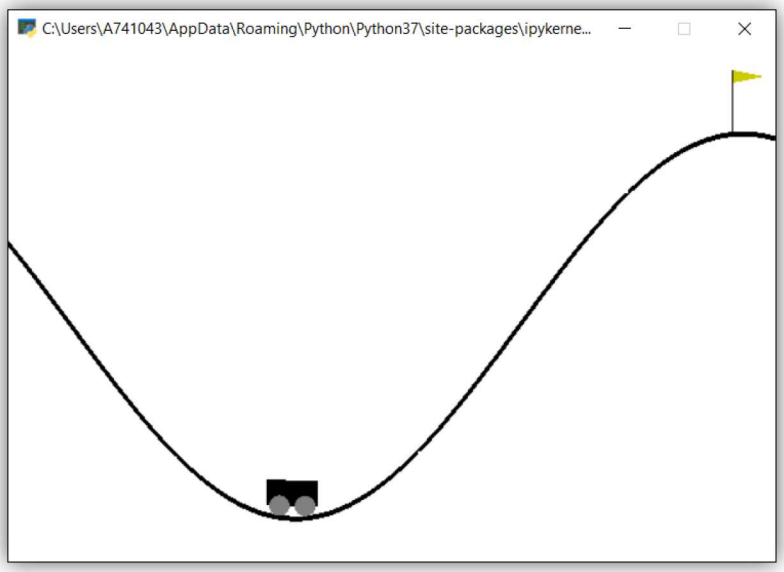
Gym sadrži mnogo okolina koje možemo koristiti u algoritmima podržanog učenja. U našem primjeru, koristit ćemo okolinu koja se zove MountainCar-v0.

6.2.1 Opis problema

U MountainCar okolini, agent je auto koji se nalazi na dnu brda. Na vrhu brda, s desne strane, nalazi se žuta zastavica, do koje agent mora doći. Dakle, agent mora naučiti kako se popeti na brdo i doći do zastavice, naravno, s ciljem da ukupna nagrada koju će zaraditi bude što veća.

Iako agent ne mora ništa znati o okolini u kojoj se nalazi (jer je algoritam Q-učenja algoritam bez modela), mi ćemo tu okolinu malo proučiti kako bi nama osobno bilo lakše shvatiti ovaj primjer Q-učenja.

Prikaz agenta i okoline možemo vidjeti na sljedećoj slici:



Slika 10: MountainCar okolina

Pogledajmo za početak, kako inicijalizirati okolinu i reći agentu da poduzima slučajno odabrane akcije u njoj.

```
import gym

env = gym.make('MountainCar-v0') # inicijalizacija okoline
env.reset()
print(env.reset()) # početno stanje

for i in range(500):
    env.render()
    env.step(env.action_space.sample()) # poduzimanje nasumično odabrane akcije
env.close()
```

```
[-0.4294818  0.       ]
```

Kod 1: Inicijalizacija okoline

Pomoću naredbe *gym.make(MountainCar-v0)* kreiramo okolinu MountainCar. Proces započinje pozivanjem naredbe *env.reset()* koja vraća početno stanje okoline. Možemo vidjeti kako nam početno stanje $[-0.4294818, 0]$ daje dvije vrijednosti. Prvi element daje nam poziciju auta, s obzirom na x-os, dok nam druga vrijednost označava brzinu auta koja je na početku 0 jer se auto još ne kreće. Nakon što smo inicijalizirali okolinu i dobili njeno početno stanje, ulazimo u petlju od 500 koraka, gdje u svakom koraku prikažemo okolinu (*env.render()*) te napravimo slučajno odabranu akciju

(*env.step(env.action_space.sample())*).
Time dobijemo animaciju okoline i agenta koji poduzima 500 nasumično odabranih akcija. (1 korak = 1 akcija)

Proučimo sada koje sve akcije i stanja, u ovoj okolini, možemo imati. Od pomoći će nam biti sljedeće naredbe:

```
#izvlacenje informacija iz okoline

# akcije koje možemo poduzeti
print(env.action_space)
print(env.action_space.n)
# dimenzija stanja
print(env.observation_space)
# najviša granica stanja
print(env.observation_space.high)
# najniža granica stanja
print(env.observation_space.low)
```

```
Discrete(3)
3
Box(2,)
[0.6  0.07]
[-1.2  -0.07]
```

Kod 2: Informacije o okolini

Pomoću naredbe *action_space* možemo vidjeti koliko akcija imamo na raspolaganju (nenegativni brojevi od 0 pa nadalje). Tako vidimo da u našoj okolini, možemo poduzeti 3 akcije, 0, 1 i 2, gdje 0 znači 'idi lijevo', 1='stoj na mjestu' i 2='idi desno'. Naredba *observation_space* vraća nam dimenziju prostora stanja, a kao što smo već mogli vidjeti u početnom primjeru, dimenzija je 2, odnosno svako stanje se sastoji od dvije varijable, pozicije auta i brzine auta. Naredbe *observation_space.high* i *observation_space.low* vraćaju nam najvišu, odnosno najnižu granicu prostora stanja pa, zahvaljujući tome, možemo vidjeti koje su rubne vrijednosti pozicije i brzine.

Pogledajmo na još jednom, malo detaljnijem primjeru, kako izgledaju nagrade u ovoj okolini. No, ovoga puta, nećemo poduzimati nasumične akcije, već ćemo u narednom primjeru, tjerati auto da se stalno kreće u desno (akcija 2) te vidjeti hoće li se na taj način auto malo više približiti zastavici. Također, pokušat ćemo saznati koliko dugo će auto pokušavati doći do cilja, tj. koji je maksimalan broj koraka moguć, nakon kojega će auto završiti jednu epizodu ("epizoda" = 1 pokušaj izvođenja, možemo gledati i kao 1 život u igrici).

```python
import gym

env = gym.make('MountainCar-v0')
state = env.reset()

done=False
i=0
while not done:
    env.render()
    action = 2
    new_state, reward, done, _ = env.step(action)
    print(reward, new_state)
    i=i+1
    if done:
        print("Episode finished after {} timesteps".format(i))
        break

env.close()
```

```
-1.0 [-0.27551177  0.00726513]
-1.0 [-0.26894021  0.00657156]
-1.0 [-0.26309815  0.00584206]
-1.0 [-0.25801695  0.0050812 ]
-1.0 [-0.25372346  0.00429349]
-1.0 [-0.25024005  0.00348341]
-1.0 [-0.24758463  0.00265542]
-1.0 [-0.24577074  0.0018139 ]
-1.0 [-0.24480753  0.0009632 ]
-1.0 [-2.44699878e-01  1.07656518e-04]
-1.0 [-0.24544831 -0.00074843]
-1.0 [-0.24704906 -0.00160075]
-1.0 [-0.24949404 -0.00244499]
-1.0 [-0.25277083 -0.00327679]
-1.0 [-0.25686262 -0.00409179]
-1.0 [-0.26174816 -0.00488554]
-1.0 [-0.26740174 -0.00565358]
-1.0 [-0.27379313 -0.00639139]
Episode finished after 200 timesteps
```

Kod 3: Neprestano kretanje u desno

Na primjeru možemo vidjeti kako je agent dobio nagradu u vrijednosti -1 za svaki korak u epizodi, odnosno za svaku akciju koju je napravio. Nagrada će iznositi 0 samo kada agent dođe u stanje gdje se nalazi zastavica, tj. kada dođe u ciljno stanje. Također, možemo vidjeti kako je u toj epizodi agent izvršio 200 koraka, što nam je defaultni maksimalan broj koraka po epizodi, nakon čega, ukoliko agent već ranije nije došao do cilja, epizoda se završava.

Još jedna stvar koja nam je ostala za objasniti u prethodnom primjeru jest što nam sve naredba *env.step* vraća. Već smo spomenuli u prvom primjeru kako nam ta naredba služi za izvršavanje određene akcije, no nismo još rekli što sve dobijemo kao rezultat izvršavanja te naredbe.

env.step vraća nam četiri vrijednosti. *new_state* je novo stanje koje nam daje okolina, nakon što poduzmemo određenu akciju. *reward* nam daje nagradu koju dobijemo za prelazak iz jednog stanja u drugo, odnosno za dolazak u novo stanje. *done* nam govori da li je vrijeme za resetirati okolinu, odnosno je li epizoda završila, dok nam zadnji parametar vraća neke informacije korisne za uklanjanje pogrešaka u kodu.

U oba primjera, kada agent poduzima slučajno odabrane akcije i kada agent neprestano poduzima akciju desno, na animaciji možemo vidjeti kako agent ne uspjeva niti se približiti zastavici, a kamoli doći do nje. Poduzimanje ovih nasumičnih akcija ili forsiranje kretanja udesno, ne pomaže nam doći do cilja. Razlog tomu je to što agent nema dovoljno snage, odnosno nema dovoljan zalet kako bi se uspio popeti na brdo. Mi već sada možemo zaključiti kako je rješenje to da se agent na neki način zanjiše, krene desno, pa se vrati lijevo pa ponovno desno, sve dok ne dobije dovoljno ubrzanje i dovoljan zalet kako bi se uspio popeti na brdo.

Ovaj problem ćemo riješiti koristeći Q-učenje. Kako smo već više puta spomenuli, Q-učenje je algoritam koji se ne temelji na modelu, tako da agentu nisu potrebne informacije o tome kako izgleda okolina i slično. Ovo što smo do sada proučavali, bilo je isključivo za nas, kako bi nama bilo lakše shvatiti ovaj primjer.
Iako smo mi razvili ideju o tome kako bi ovaj problem trebali riješiti, pustit ćemo agenta da sam isprobava situacije i proba sam naći rješenje ovog problema.

6.2.2 Primjena Q-učenja

Kada krećemo s primjenom Q-učenja, prvo i najbitnije pitanje koje nam se postavlja je kako kreirati Q-tablicu? Znamo kako se Q-tablica sastoji od vrijednosti svih mogućih kombinacija stanja i akcija. No, sada, u ovome primjeru, dolazimo do jednog problema. Mogli smo vidjeti u prethodnim primjerima kako vrijednosti stanja koja nam okolina daje, odnosno pozicija i brzina, imaju čak do 8 decimala. To je jednostavno previše informacija i kada bi radili sve kombinacije za do 8 decimala, Q-tablica bila bi ogromna i zauzimala mnogo memorije, a uostalom, nije nam niti potrebna tolika preciznost.
Kako bi smanjili toliku količinu nepotrebnih informacija, podatke ćemo grupirati u nešto čime ćemo moći lakše upravljati. Grupirat ćemo ih u 20 diskretnih skupova, kako bi smanjili Q-tablicu (ovaj broj možemo odabrati proizvoljno).

Sljedeći kod služi nam kako bi razdijelili stanja u diskretne skupove:

```
size = [20,20]
group_size = (env.observation_space.high - env.observation_space.low)/size

print(group_size) #veličina svake grupe
```

```
[0.09  0.007]
```

Kod 4: Podjela u diskretne skupove

group_size označava nam veličinu svakog pojedinog skupa. Sada kada smo raspodijelili stanja na 20 diskretnih podskupova, možemo definirati početnu Q-tablicu koju ćemo svakim novim korakom ažurirati. Q-tablicu definiramo na sljedeći način:

```
import numpy as np

Q_table = np.random.uniform(low=-1, high=1, size=(size + [env.action_space.n]))
print(Q_table.shape)
print(Q_table)
```

```
(20, 20, 3)
[[[-0.27712351  0.94733084  0.8957068 ]
  [-0.56161997  0.95546111  0.91654723]
  [ 0.9920337   0.44869542  0.01146692]
  ...
```

Kod 5: Kreiranje Q-tablice

Q-tablica je dimenzije 20x20x3, 20x20 za kombinacije pozicija i brzina, a x3 jer imamo na izboru 3 akcije. Tablicu smo popunili slučajno odabranim vrijednostima između -1 i 1. Vrijednosti pomoću kojih ćemo popuniti početnu Q-tablicu također možemo birati, ali obično uzimamo vrijednosti oko 0.

Sada, kada smo kreirali Q-tablicu i razdijelili stanja u 20 diskretnih skupova, trebat će nam pomoćna metoda koja će svako stanje koje dobijemo od okoline, prilagoditi tim diskretnim skupovima i pretvoriti ih u cjelobrojna stanja kako bi nakon toga jednostavno pogledali u Q-tablicu i za tu kombinaciju pozicije i brzine, odabrali akciju koja nam daje najveću vrijednost.

Pomoćna metoda koju ćemo koristiti je sljedeća:

```python
def get_state(state):
    state = (state - env.observation_space.low)/group_size
    print(state)
    return tuple(state.astype(np.int))

print(env.reset())
state = get_state(env.reset())
print(state)
print(Q_table[state])
print(np.argmax(Q_table[state])) #take this action
```

```
[-0.42188478  0.        ]
[ 6.96812838 10.        ]
(6, 10)
[-0.25956522 -0.25033506 -0.0379708 ]
2
```

Kod 6: Pomoćna funkcija

Sada, nakon što smo kreirali Q-tablicu i napisali pomoćnu funkciju, preostalo nam je još napisati u Pythonu formulu pomoću koje ćemo ažurirati Q-vrijednosti. Navedenu formulu smo već ranije spomenuli:

$$Q^{\text{new}}(s,a) \leftarrow (1-\alpha)\underbrace{Q(s,a)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s',a)}_{\text{estimate of optimal future value}}\right)}^{\text{learned value}},$$

a u Pythonu ćemo je zapisati na sljedeći način:

```python
new_q = (1 - alpha) * current_q + alpha * (reward + gamma * max_future_q)
```

Kod 7: Ažuriranje Q-vrijednosti

gdje je *max_future_q = np.max(Q_table[new_state])*.

Postoji još jedna vrlo bitna stvar kod treniranja modela podržanog učenja. Rekli smo već kako se podržano učenje bazire na metodi pokušaja i pogreške jer ne znamo kako izgleda okolina te je na taj način pokušavamo shvatiti i otkriti koja stanja su dobra, koja loša i slično. Problem je što, prilikom istraživanja okoline, ukoliko previše "lutamo", putem možemo gubiti nagrade. Kako bi agent istovremeno i istraživao i učio, kombiniramo dvije metode: istraživanje odnosno odabir gubljenja nagrada kako bi istražili okolinu i dobili malo više informacija o njoj, npr. istražiti nove putove kojima možemo ići (eng. exploration) i eksploataciju to jest istraživanje puta koji nam je već poznat kako bi maksimizirali nagradu (eng. explotation). U podržanom

učenju, vrlo je bitno koristiti i istraživanje i eksploataciju.

Osvrnimo se na naš primjer. U Q-učenju koristimo pohlepnu strategiju te će naš model, uvijek odabirati maksimalnu Q-vrijednost, što nam ovdje predstavlja eksploataciju. No, kako smo na početku Q-tablicu popunili nasumičnim vrijednostima, one su nam na početku nebitne jer ne predstavljaju prave vrijednosti koje agent želi uzeti. Ono što želimo jest da agent na početku treniranja, istražuje okolinu kako bi naučio nove putove i ažurirao Q-vrijednosti iz naučenog. Iz tog razloga, kako bi natjerali agenta da na početku više istražuje, uvodimo novi parametar epsilon:

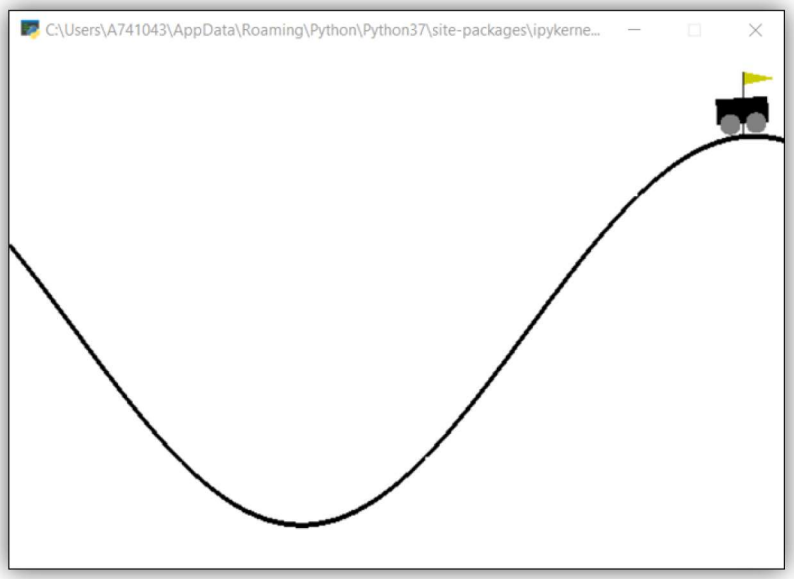
```
epsilon = 0.8
min_eps = 0
...

# računanje epizodne redukcije epsilona
reduction = (epsilon - min_eps)/(episodes//2)
...

while not done:
    # određivanje sljedeće akcije - epsilon pohlepna strategija
    if np.random.random() > epsilon:
        # poduzimanje akcije iz Q-tablice
        action = np.argmax(Q[state_adj[0], state_adj[1]])
    else:
        # poduzimanje random akcije
        action = np.random.randint(0, env.action_space.n)

...
# redukcija epsilona
if epsilon > min_eps:
    epsilon -= reduction
```

Kod 8: Istraživanje okoline

Na slici možemo vidjeti kako smo uveli parametar *epsilon* te nakon svake epizode smanjivali njegovu vrijednost sve dok nije došao do 0. Epsilon smo koristili tako da smo na početku učenja većinom poduzimali nasumične akcije, dok kasnije, kako smanjujemo vrijednost epsilona, koristimo se eksploatacijom i pohlepnom strategijom, te poduzimamo akcije na temelju Q-tablice.

Sada imamo sve parametre i funkcije potrebne kako bi agent mogao učiti te možemo početi trenirati agenta. U prilogu 1 nalazi se python kod koji smo koristili za treniranje agenta.

Nakon nekog vremena, kada agent već pomalo nauči kako izgleda okolina i ažurira Q-tablicu, uspjeva dostići svoj cilj i doći do zastavice na brdu:



Slika 11: MountainCar s Q-učenjem

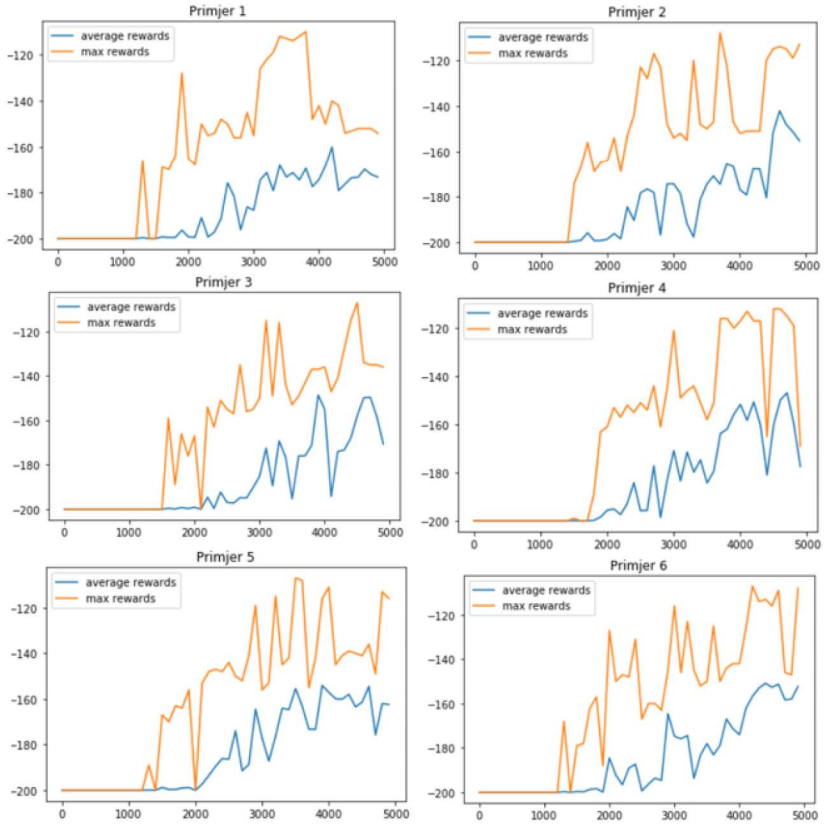
6.2.3 Usporedba rezultata

Prethodno smo vidjeli kako izgleda implementacija Q-učenja na ovom primjeru, a sada je vrijeme za "igranje" s parametrima i usporedbu rezultata.
Na nekoliko primjera s različitim vrijednostima parametara, pogledat ćemo je li agent uspio naći put do cilja te koliku je maksimalnu nagradu uspio dobiti. Agenta ćemo trenirati kroz 5000 epizoda. Iako i ovaj parametar možemo mijenjati, koristit ćemo fiksan broj kako bi nam bilo lakše vidjeti dobivene rezultate.

Nakon što smo kroz šest primjera trenirali agenta, na sljedećoj stranici možemo vidjeti dobivene rezultate u tablici te slike koje nam prikazuju prosječne i maksimalne nagrade koje je agent uspio ostvariti u svakom od navedenih primjera.
U prilogu 2 nalazi se kod pomoću kojega smo prikazali rezultate agentovog treniranja.

	size	alpha	gamma	epsilon	Max reward	Ep with max reward	First succ ep	Number of succ eps
Primjer 1	[20,20]	0.2	0.9	0.8	-110	3782	1226	2235
Primjer 2	[20,20]	0.1	0.95	0.9	-108	3682	1444	2157
Primjer 3	[20,20]	0.1	0.8	1	-107	4431	1575	1728
Primjer 4	[30,30]	0.1	0.95	1	-112	4479	1481	2140
Primjer 5	[40,40]	0.2	0.9	0.8	-107	3491	1271	2388
Primjer 6	[40,40]	0.1	0.95	0.9	-107	4124	1298	2119

Tablica 1: Usporedba rezultata



Slika 12: Nagrade

Navedimo sada neke zaključke koje možemo izvesti iz prikazanih rezultata.
U svakom od šest isprobanih primjera, agent je naučio kako doći do cilja. Na temelju tablice i slika možemo vidjeti kako se rezultati treniranja nisu drastično mijenjali. Maksimalna nagrada kroz 5000 epizoda u svakom primjeru iznosi oko -110. Agent je na početku treniranja istraživao okolinu te je, nakon otprilike 1400 epizoda, uspio pronaći uspješan put koji ga je doveo do zastavice. Broj uspješnih epizoda u kojima je agent došao do cilja, u svakom primjeru, iznosi oko 2000.

Na kraju ovoga primjera možemo zaključiti kako ima mnogo parametara koje možemo mijenjati i isprobavati kako bi vidjeli postoji li neka bolja kombinacija kojom će agent još brže i bolje naučiti kako doći do cilja i zaraditi još veću nagradu.

Mi, u ovih šest primjera, nismo drastično eksperimentirali s parametrima, samo ih malo mijenjali. Također, primjetimo i to kako nismo mijenjali broj epizoda kroz koje će agent učiti, što također otvara vrata drugim i možda boljim kombinacijama. No to ćemo ostaviti na volju čitateljima da isprobavaju i igraju se s parametrima kako žele te pronađu najuspješniju kombinaciju.

Zaključak

Nakon što smo prošli kroz osnove podržanog učenja i Q-učenja, na kraju ovoga rada možemo donijeti neke bitne zaključke o podržanom učenju.

Zašto koristiti podržano učenje?
Vrlo jednostavno, iz razloga što nam podržano učenje pomaže odlučiti koju akciju poduzeti u određenoj situaciji, odnosno koje akcije će nam donijeti najveću nagradu. Može se primjeniti na razne situacije i probleme, od igara pa sve do robotike ili regulacije prometa.

Kada ne trebamo koristiti podržano učenje?
Ukoliko smo u mogućnosti problem riješiti pomoću nadziranog učenja, ne trebamo koristiti podržano učenje. Razlog tomu je to što je podržano učenje dosta zahtjevan posao, kako za nas, tako i za računala pa njegova primjena oduzima puno vremena. Stoga, ako problem možemo riješiti na lakši način, preporuka je da odaberemo taj lakši način.

Glavni izazovi podržanog učenja
Postoji mnogo izazova s kojima se susrećemo pri primjenjivanju podržanog učenja. Neki od glavnih problema su osmišljavanje nagrada, postavljanje parametara (dobro odabrane vrijednosti parametara mogu ubrzati proces učenja), realistične okoline mogu biti djelomično vidljive te stvari u realističnim okolinama mogu biti nestacionarne (promjenjive).

Kao što smo vidjeli u ovome radu, podržano učenje se primjenjuje u raznim znanstvenim područjima. Posebice unazad par godina, znanstvenici mu posvećuju sve više i više pažnje. No, podržano učenje i dalje ima puno problema i nije ga nimalo lako koristiti. Ipak, sve dok se ulaže više napora u rješavanje problema, primjena podržanog učenja može nam donijeti svijetlu budućnost i riješiti te probleme.

Literatura

[1] Nandy, A., Biswas, M. (2018). *Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python*. Apress

[2] A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python
URL: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

[3] AI - Agents & Environments
URL: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_agents_and_environments.htm

[4] Applications of Reinforcement Learning in Real World
URL: https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

[5] Components of a Reinforcement Learning Agent and it's application on SNAKE
URL: https://becominghuman.ai/components-of-an-rl-agent-and-its-application-on-snake-1b3b6c8e1de5

[6] Gym Documentation
URL: http://gym.openai.com/docs/

[7] Introduction to Various Reinforcement Learning Algorithms
URL: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

[8] Machine Learning Definition and Application Examples
URL: https://www.spotlightmetal.com/machine-learning--definition-and-application-examples-a-746226/?cmp=go-aw-art-trf-SLM_DSA-20180820&gclid=EAIaIQobChMI5Y-3z-3x5AIVhOJ3Ch0GDgfPEAAYASAAEgJIpPD_BwE

[9] Machine Learning For Beginners
URL: https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab

[10] Q-learning for solving the OpenAI Gym Mountain Car problem
URL: https://gist.github.com/gkhayes/3d154e0505e31d6367be22ed3da2e955

[11] Reinforcement Learning by Georgia Tech (Udacity course)
URL: https://www.udacity.com/course/reinforcement-learning--ud600

[12] Reinforcement Learning Isomorphisms
URL: http://www.sashasheng.com/blog/2018-1-6-reinforcement-learning-taxonomy

[13] Reinforcement Learning Python Tutorial
URL: https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/

[14] Reinforcement Learning: What is, Algorithms, Applications, Example
URL: https://www.guru99.com/reinforcement-learning-tutorial.html

[15] Reinforcement Q-Learning from Scratch in Python with OpenAI Gym
URL: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

[16] RL Course by David Silver, *URL:*
https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT

[17] Simple Beginner's guide to Reinforcement Learning & its implementation
URL: https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/

[18] Uvod u strojno učenje
URL: http://degiorgi.math.hr/~singer/ui/ui_1415/ch_18a.pdf

Prilog 1

Prilog sadrži kompletan python kod koji smo koristili za treniranje agenta pomoću Q-učenja.

```
class QLearning:

    def __init__(self, alpha, gamma, epsilon, min_eps, episodes,
    ↪  size):

        self.alpha = alpha
        self.gamma = gamma
        self.eps = epsilon
        self.min_eps = min_eps
        self.episodes = episodes
        self.size = size

        self.env = gym.make('MountainCar-v0')

        self.Q_table = np.random.uniform(low=-1, high=1, size=(size +
        ↪  [self.env.action_space.n]))

        # računanje epizodne redukcije epsilona
        self.reduction = (self.eps - self.min_eps)/(self.episodes//2)

        # praćenje nagrada
        self.episodes_rewards = []
        self.rewards = {'ep': [], 'avg': [], 'max': []}
        self.succ_episodes = {'ep': [], 'reward': []}
        self.STATS_EVERY = 100

    def get_state(self, state):

        group_size = (self.env.observation_space.high -
        ↪  self.env.observation_space.low)/self.size
        state = (state - self.env.observation_space.low)/group_size
        return tuple(state.astype(np.int))

    def select_action(self, state):
```

```
        if np.random.random() > self.eps:
            # poduzimanje akcije iz Q-tablice
            action = np.argmax(self.Q_table[state])
        else:
            # poduzimanje random akcije
            action = np.random.randint(0, self.env.action_space.n)
        return action

    def update_q_table(self, done, new_state, state, action, reward):

        # ukoliko je postignut cilj, ažuriraj Q-vrijednost s
        ↪ nagradom 0
        if done and new_state[0] >= self.env.goal_position:
            self.Q_table[state][action] = reward
        # ako simulacija još traje, ažuriraj Q-vrijednost pomoću
        ↪ formule
        else:
            # maksimalna Q-vrijednost za novo stanje
            max_future_q =
            ↪ np.max(self.Q_table[self.get_state(new_state)])
            current_q = self.Q_table[state][action] # trenutna
            ↪ q-vrijednost (za trenutno stanje i odabranu akciju)
            # formula za ažuriranje q-vrijednosti
            new_q = (1 - self.alpha) * current_q + self.alpha *
            ↪ (reward + self.gamma * max_future_q)
            # ažuriraj Q-tablicu s novom q-vrijednosti
            self.Q_table[state][action] = new_q

    def train(self):

        for episode in range(self.episodes):
            state = self.get_state(self.env.reset())
            done = False
            episode_reward = 0

            while not done:
                # prikaži zadnjih 10 epizoda
                if episode >= (episodes - 10):
                    self.env.render()
```

```
                action = self.select_action(state)
                # poduzmi akciju
                new_state, reward, done, _ = self.env.step(action)

                # spremi nagradu
                episode_reward += reward

                # ažuriraj Q-tablicu
                self.update_q_table(done, new_state, state, action,
                ↪  reward)

                # ažuriraj stanje
                state = self.get_state(new_state)

                # ukoliko je postignut cilj, spremi uspješnu epizodu
                if done and new_state[0] >= self.env.goal_position:
                    self.succ_episodes['ep'].append(episode+1)

                    ↪  self.succ_episodes['reward'].append(episode_reward)

            # redukcija epsilona
            if self.eps > self.min_eps:
                self.eps -= self.reduction

            # spremi nagrade
            self.save_rewards(episode, episode_reward)

        self.env.close()

        return (self.rewards, self.succ_episodes)

    def save_rewards(self, episode, episode_reward):

        # spremi nagradu od svake epizode
        self.episodes_rewards.append(episode_reward)

        # statistika
        if episode % self.STATS_EVERY == 0:
            average_reward =
            ↪  np.mean(self.episodes_rewards[-self.STATS_EVERY:])
```

```
max_reward =
↪  np.max(self.episodes_rewards[-self.STATS_EVERY:])
self.rewards['ep'].append(episode)
self.rewards['avg'].append(average_reward)
self.rewards['max'].append(max_reward)
print('Episode {}, Average Reward: {}, Max Reward: {},
↪  Current eps: {}'.format(episode, average_reward,
↪  max_reward, round(self.eps,2)))
```

Prilog 2

Ovaj prilog sadrži python funkciju pomoću koje prikazujemo agentove rezultate učenja.

```
def plot_rewards(rewards, succ_episodes):

    plt.plot(rewards['ep'], rewards['avg'], label="average rewards")
    plt.plot(rewards['ep'], rewards['max'], label="max rewards")
    plt.title("Primjer 1")
    plt.legend(loc=2)
    plt.show()

    if succ_episodes['ep'] == []:
        print("There are no successful episodes.")
    else:
        max_reward = np.max(succ_episodes['reward'])
        ep_max =
        ↪  succ_episodes['ep'][np.argmax(succ_episodes['reward'])]
        succ_ep = len(succ_episodes['ep'])
        first_succ = succ_episodes['ep'][0]
        print('Max reward {} in episode {}, Number of succ episodes:
        ↪  {}, First succ ep: {}'.format(max_reward, ep_max, succ_ep,
        ↪  first_succ))
```

Sažetak

Sažetak. Ovaj diplomski rad proučava podskupinu strojnog učenja koja se naziva podržano učenje. U radu pokrivamo osnove podržanog učenja i proučavamo jedan algoritam podržanog učenja, Q-učenje, te uz pomoć Python programskog jezika, na praktičnom primjeru, primjenjujemo taj algoritam.
Prvo poglavlje ovoga rada sadrži uvod u kojem čitatelju dajemo do znanja kako se, u današnje vrijeme, umjetna inteligencija i strojno učenje sve više i više razvijaju i primjenjuju u svakodnevnom životu. Uvodimo neke osnovne probleme kojima se strojno učenje bavi kako bi na samome početku dobili osnovnu percepciju o ovom području računalne znanosti.
U drugom poglavlju ovoga rada upoznajemo se s osnovnim konceptima strojnog učenja te trima vrstama strojnog učenja: nadzirano, nenadzirano i podržano učenje.
Treće poglavlje namjenjeno je isključivo proučavanju podržanog učenja. U tom poglavlju možemo vidjeti gdje se sve primjenjuje podržano učenje te po čemu se ono razlikuje od ostalih vrsta strojnog učenja. Nakon toga, proučavamo osnovne karakteristike podržanog učenja te osnovne komponente. Upoznajemo se sa scenarijem podržanog učenja i proučavamo tko je agent, što je okolina te što su stanja, akcije i nagrade.
Nakon što smo smo se upoznali s osnovama podržanog učenja, u četvrtom poglavlju možemo vidjeti na koji način dijelimo algoritme podržanog učenja.
Peto poglavlje posvećeno je Markovljevom procesu odlučivanja gdje možemo vidjeti kako matematički formulirati problem podržanog učenja.
Na samome kraju, u šestom poglavlju, proučavamo jedan algoritam podržanog učenja koji se naziva Q-učenje te na jednom jednostavnom primjeru gledamo kako primjeniti taj algoritam u praksi.

ključne riječi: strojno učenje, podržano učenje, agent, okolina, stanje, akcija, nagrada, Markovljev proces odlučivanja, Q-učenje.

Summary

Summary. This paper examines a subset of machine learning called reinforcement learning. In this paper we cover the basics of reinforcement learning and study one of reinforcement learning algorithms, Q-learning, and with the help of Python programming language, we apply this algorithm on a practical example.
The first chapter of this paper contains an introduction in which we inform the reader how, nowadays, artificial intelligence and machine learning are increasingly being developed and applied in everyday life. We introduce some basic problems that machine learning deals with in order to get a basic perception of this area of computer science.
In the second chapter of this paper we are introduced to the basic concepts of machine learning and the three types of machine learning: supervised, unsupervised and reinforcement learning.
The third chapter is intended exclusively for the study of reinforcement learning. In this chapter we can see where all the reinforcement learning is applied and how it differs from other types of machine learning. After that, we study the characteristics of reinforcement learning and basic components. We learn about typical reinforcement learning scenario and study who the agent is, what the environment is, and what the states, actions and rewards are.
After learning basics of reinforcement learning, in Chapter 4, we study about reinforcement learning algorithms.
The fifth chapter is devoted to Markov Decision Process, where we can see how to mathematically formulate the problem of reinforcement learning.
Finally, in Chapter Six, we study a certain reinforcement learning algorithm called Q-learning, and in one simple example, we can see how to implement that algorithm.

keywords: machine learning, reinforcement learning, agent, environment, state, action, reward, Markov Decision Process, Q-Learning.

Životopis

Lucija Blažević rođena je 28.12.1991. godine u Virovitici. U rodnom gradu, pohađala je osnovnu školu Vladimir Nazor te Jezičnu Gimnaziju Petra Preradovića. Godine 2012. upisuje preddiplomski studij matematike na Odjelu za matematiku u Osijeku. Nakon završenog preddiplomskog studija i stečenog zvanja prvostupnika matematike, upisuje sveučilišni diplomski studij matematike i računarstva na istoimenom fakultetu.
Na drugoj godini diplomskog studija odrađuje praksu u tvrtki Atos, s kojom nastavlja daljnju suradnju sve do danas.