Mali Mate

Projektni zadatak iz kolegija *Strojno učenje* na PMF-u, 2019./2020.

Karlo Lončarević
Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu
karlo.lonac@gmail.com

Mateja Savić
Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu
matejasavic8@gmail.com

Lovro Sindičić
Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu
lovro.sindicic@gmail.com

***Sažetak*—Učenje s podrškom omogućava agentu da optimizira svoje ponašanje iz interakcije s određenim okruženjem. Iako su razvijene neke vrlo uspješne aplikacije algoritama za učenje s podrškom, još uvijek je otvoreno pitanje kako proširiti opseg do velikih dinamičkih okruženja. U ovom ćemo radu proučiti upotrebu učenja u popularnoj arkadnoj video igri Pac-Man. Kako bi Pac-Man brzo naučio, dizajnirali smo različite algoritme poput Q-learning i Approximate Q-learning. Na kraju smo problem pokušali riješiti korištenjem konvolucijske neuronske mreže.**

Kako bismo čitatelju omogućili bolje razumijevanje problema ovaj rad pruža i osnovno znanje koje je potrebno da se razumije osnovne principe učenja, korištenja dubokog učenja, implementacije značajka i izgradnju arhitekture mreže.

***Index Terms*—Strojno učenje, učenje s podrškom, Q-learning, neuronske mreže, izbor značajki, regresija, Pac-Man, metodika, Codey Rocky**

I. Uvod i problem

Pac-Man je jedna od najpoznatijih *vintage* video igrica koju je razvila japanska kompanija Namco 1980. godine. Svoju popularnost iz ranih dana održala je i dan danas te se sve češće javljaju zanimljivi gifovi i memeovi temeljeni na Pac-Manu što dodatno govori i o tome da su mlađe generacije upoznate s ovom video igricom.

Kao budući nastavnici informatike u toku svojeg studija susreli smo se s metodičkim pristupom programiranju robota[1]. Naišli smo na problem nedostataka raznih aktivnosti i zato smo si postavili pitanje možemo li koristeći dosadašnje znanje napraviti vlastitu verziju Pac-Mana pomoću robota.

Ovaj projekt realizirat ćemo pomoću robota Codey Rocky s određenim odmakom od tradicionalnih pravila igre s obzirom na to da nam konstrukcija robota ne omogućava neke od realizacija koje su moguće u video verziji igrice (jedenje igrača i sl.), ali i za njih ćemo pronaći jednako zanimljiva i kreativna rješenja.

II. Učenje podrškom

Cilj učenja podrškom je naučiti agenta određenu akciju kroz interakciju s okolišem. Okoliš je opisan skup stanja S, dok agent može učiniti neku akciju iz skupa A. Svakim izvođenjem akcije a u stanju s agent dobiva trenutnu nagradu r i sukladno nagradi koju je dobio on prelazi u novo stanje i ponavlja se cijeli postupak do izvršenja zadataka.

Način na koji agent uči je ovisno o nagradi koja mu se da. Cilj na kraju je sakupiti što veću nagradu. $r_0 + r_1 + \dots$. Kako navedena suma u nekim primjenama može divergirati primjenjuje se maksimalna očekivana kumulativna nagrada. $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots,\ 0 \leq \gamma < 1$.

Lema 1. *Ako je skup nagrada R konačan, onda je maksimalna očekivana kumulativna nagrada realan broj.*

Dokaz. Neka je $R = \{R_1, R_2, \dots, R_n\}$ skup svih različitih nagrada koje igrač može sakupiti. Budući da je skup R konačan totalno[2] uređen skup postoji najveći element tog skupa. Označimo taj element s M.

$$\sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \leq \sum_{k=1}^{\infty} \gamma^k \cdot M = M \cdot \frac{1}{1-\gamma} < +\infty$$

□

A. Markovljev lanac

Ovaj smo problem modelirali korištenjem konačnog Markovljevog lanca pa zato za početak navedimo njegovu definiciju i neka bitna svojstva koja smo koristili u rješavanju problema.

Definicija 1. *Neka je skup S takav da vrijedi $k(S) \leq \aleph_0$. Slučajni proces $X = ({X_n}^3 : n \geq 0)$ definiran na vjerojatnosnom prostoru $(\Omega, \mathscr{F}, \mathbb{P})$ s vrijednostima u skupu S je Markovljev lanac ako vrijedi*

$$\mathbb{P}(X_{n+1} = j | X_n = i, \dots, X_0 = i_0) = \mathbb{P}(X_{n+1} = j | X_n = i) \tag{1}$$

za svaki $n \geq 0$ i za sve $i_0, \dots, i_{n-1}, i, j \in S$ za koje su obje uvjetne vjerojatnosti dobro definirane. Svojstvo u relaciji zovemo (1) naziva se Markovljevim svojstvom. Pretpostavimo da se nalazimo u vremenskom trenutku n. Tada vrijeme $n+1$ predstavlja neposrednu budućnost, dok vremena $0, 1, \dots, n-1$

[1]Prema novome kurikulumu [1] za OŠ neki od ishoda koji to obuhvaćaju su C.1.3, B.2.1, A.4.2, B.5.2, B.6.2.

[2]Za parcijalno uređen skup $(A, <)$ kažemo da je totalno uređen ako su svaka dva njegova različita elementa usporediva.

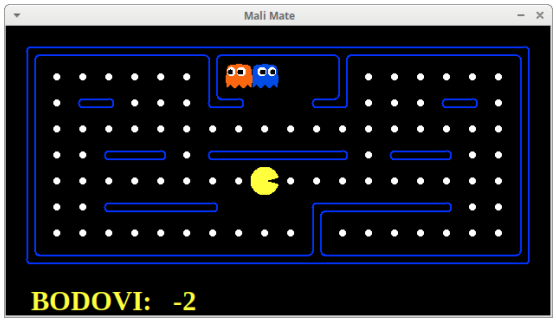
[3]Slučajnu varijablu X ćemo za naše potrebe definirati kao funkciju X: $\Omega \to \mathbb{R}$ ako vrijedi $X^{-1}(B) \in \mathscr{F}$, za svaki $B \in \mathscr{B}$, pri čemu je $\mathscr{B}$ σ -algebra Borelovih skupova.

predstavljaju prošlost. Markovljevo svojstvo nam govori da je ponašanje Markovljevog lanca u neposrednoj budućnosti, uvjetno na sadašnjost i prošlost, jednako ponašanju Markovljevog lanca u neposrednoj budućnosti, uvjetno na sadašnjost i prošlost, jednako ponašanju Markovljevog lanca u neposrednoj budućnosti, uvjetno na samo sadašnjost.
Drugi način na koji možemo iskazati Markovljevo svojstvo je sljedeće: (neposredna) budućnost i prošlost uvjetno su nezavisne uz danu sadašnjost što se može i precizno matematički dokazati.

B. Skup S

Sve metode učenja s podrškom koja ćemo implementirati biti će bazirane na emulatoru za Pac-Man igru izrađenom na sveučilištu Berkeley u Kaliforniji [4]. Pravila će biti jednaka originalnoj igri Pac-Man, izuzev toga što će agent imati samo jedan život i neće biti debelih bijelih točkica. Prisjetimo se, ako ih agent pojede, tada neometano može pojesti lovce. Za navedenu promjenu pravila odlučili smo se zbog jednostavnije implementacije cijele igre s robotima.

Agent i lovci se u toku igre kreću po diskretnoj koordinatnoj mreži po slobodnim poljima. Na donjoj slici 1 navodimo primjer vlastito izrađenog labirinta kojeg ćemo koristiti prilikom demonstracije projekta s robotima. U daljnjem tekstu naveden labirint nazivamo Sunce.



Slika 1. labirint Sunce

Svaki trenutak igre jedinstveno je opisan matricom koja sadrži zid, točke, Pac-Mana i lovce.

```
%%%%%%%%%%%%%%%%%%%%
%......% GG %......%
%.%%...%%  %%...%%.%
%..................%
%..%%%.%%%%%.%%%..%
%........P.........%
%..%%%%%   %%%%%..%
%..........%.......%
%%%%%%%%%%%%%%%%%%%%
```

Listing 1. labirint Sunce

Testiranja ćemo provoditi na većem broju različitih labirinta koji su poznati iz igre. Gornji labirint dimenzija je $20 \cdot 9$ što nam daje ukupan broj različitih situacija $92^3 \cdot 4 \cdot 2^{82} = 150620164579852218046976444433408$. S obzirom na veliki broj raznih mogućnosti trebat ćemo naći algoritme koji će iz manjeg broja naučenih primjera moći zaključiti na većem skupu koji prije nisu vidjeli.

C. Skup A

Skup stanja S bit će sve moguće konfiguracije igre uključujući poziciju igrača i lovaca, dok će skup mogućih akcija A biti ograničen na svega 5 mogućnosti: lijevo, desno, gore, dolje ili ostani na mjestu. Akcije koje igrač može izabrati jedino su limitirane zidovima kroz koje Pac-Man prema pravilima ne može proći.

Slika 2. Prikaz mogućih akcija u trenutnom stanju {gore, lijevo, ostani na mjestu}, $A(s_t) \subset A$

D. Skup R

Dobivene nagrade od strane agenta odgovaraju novim pravilima igre. Funkcija nagrade u ovom projektu je fiksirana. Odnosno, malom nagradom za jedenje točkica i visokom pozitivnom nagradom kada pobijedi. Gubitak igre rezultira velikom negativnom nagradom, a za svaku sekundu trajanja igre dodjeljuje se mala negativna nagrada u cilju zagovaranja brzih rješenja. U donjoj tablici pišu sve akcije, za koje našem agentu dajemo nagradu.

Opis	Nagrada
Pojeo sve točkice	+500
Pojeo jednu točkicu	+10
Kazna za svaku sekundu	-1
Pac-Mana su pojeli lovci	-500
Table1 Događaji u igri i pripadajuće nagrade.	

III. Pregled dosadašnjih istraživanja

Posljednjih desetak godina proveden je veći broj istraživanja u kojima se analizirala igra Pac-Man. Neki od tih istraživanja su [7] i [8]. Problemu su pristupali korištenjem algoritma Q-learning. Primjerice u [7] su rješavali igru Ms. Pac-Man vs. Ghosts, pa su umjesto Q tablice koristili su tzv. *case base*. Koristili su *Case-based reasoning* kojem je ideja je da slični problemi imaju slična rješenja. Što je inače česta ideja u rješavanju problema korištenjem umjetne inteligencije. Prema njihovim snimljenim rješenjima[4] zaključujemo da su uspješno rješili problem. U [8] su za rješavanje igre Ms. Pac-Man uz Q-learning algoritam koristili i skup značajki s kojima su opisivali zadani skup stanja. Prema njihovim rezultatima navedeni postupak bio je uspješan i Pac-Man je u testiranju pobijedio u svim igrama. Djelo Mhina i suradnika [11], gdje koriste dubinsko Q-učenje (DQL) za obuku igrača u Atari

[4]https://www.youtube.com/watch?v=4phEjPAGrbk

igrama. Budući da je dokazano da DQN može imati dobre performanse u Atari igrama, čak i bolje od čovječnih performansi u nekim igrama. DQN je privukao puno pažnje i predložena su mnoga poboljšanja. Nakon uspjeha s DQN u detaljnom radu [12] njemački stručnjaci uspijevaju kombinacijom neuronskih mreža i učenja s podrškom riješiti problem.
S tehničke strane stručni rad [6] u kojem se pomoću robota implementira Pac-Man je zaista impresivan. Neovisno o tome što se radi o potpuno drugačijoj robotskoj platformi gdje roboti imaju kamere, a način komunikacije ostvaruje se preko Socket APIa. Napomenimo da smo i mi također pokušali komunikaciju provesti na sličan način, ali su na Codey Rockyu nije moguće otvoriti portove koristeći Socket API[5]
Dakle naš problem je proučavan i rješavan već godinama, ali vrijedi ga pogledati iz malo više kuteva i probati nekolicinu različitih metoda rješavanja.

IV. Tehničke specifikacije robota

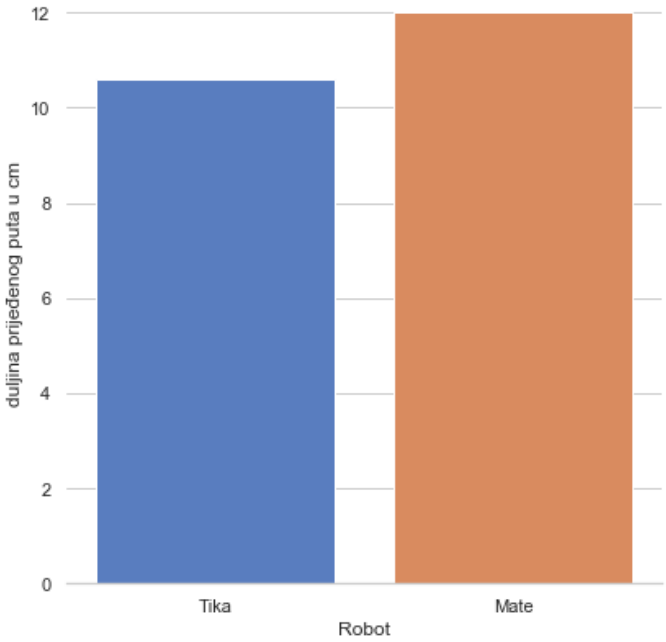
A. Opis robota

Codey Rocky proizvod je privatne kineske firme Makeblock iz 2017.
Robot je namijenjen pomoći djeci koja uče osnove programiranja i umjetne inteligencije. Sastavljen je od dva odvojiva dijela. Glava imena Codey je programirljivi kontroler koji sadrži više od 10 elektroničkih modula. Dok je tijelo odnosno, Rocky, vozilo koje može izbjeći prepreke, prepoznati boje i slijediti crte. Za svoje kretanje koristiti dva DC Geared motora, a moguće ga je programirati koristeći se blokovskim programiranjem u programu mBlock ili pisanjem koda u programskom jeziku Python.
Codey Rocky kombinira hardver sa softverom, omogućujući djeci da uče o programiranju dok se igraju i stvaraju.

Slika 3. Robot Codey Rocky

U eksploratornoj analizi smo pokazali da roboti neće davati iste rezultate za isti skup naredbi. Tako primjerice mogu prijeći različite udaljenosti ili se mogu okrenuti za različite kutove da pritom te razlike nisu male. Rezultate takvog jednog testa u kojem smo promatrali duljinu prijeđenog puta dva različita Codey Rockya se nalaze na slici 4.

[5]https://www.on-time.com/rtos-32-docs/rtip-32/reference-manual/socket-api



Slika 4. Rezultati testa - 50% snage motora u trajanju od 1 sekunde

Prema našem mišljenju nekih od mogućih razloga za to su prljavština na motorima i razina baterije. To onda može biti veliki problem prilikom demonstracije.

B. Upravljanje robotom

Naš je plan na računalu naučiti kontrolnu strategiju, a tek kasnije na naučenom modelu provesti demonstraciju pomoću robota. Sada ćemo opisati kako ćemo to napraviti. Kao što smo u ranijem odlomku naveli, robote je moguće programirati u Pythonu pa za početak navedimo popis biblioteka koje ćemo koristiti.

```
import codey, event, time
```

Listing 2. Python biblioteke koje koristiti Codey Rocky

S ugrađenim Wi-Fi-jem[6], Codey Rocky brzo se povezuje na internet, ostvarujući svoju IoT[7] funkcionalnost. Robot ima mogućnost spajanja na WiFi mrežu i prilikom spajanja na nju dobije vlastitu IP adresu.

```
@event.start
def on_start():
  codey.wifi.start('ime_WiFi_mreze', 'lozinka_mreze'
    , codey.wifi.STA)
  while not codey.wifi.is_connected():
    pass
```

Listing 3. spajanje Codey Rockya na WiFi mrežu

Zato ćemo komunikaciju računala i Codey Rockya provesti preko Wi-Fi mreže. Na računalu ćemo pokrenuti lokalni server pomoću programa XAMPP. Svaki od robota imati će svoj

[6]Wireless Fidelity
[7]Internet of things

direktoriji u kojem će biti jedna datoteka index.php koju će on čitati te ovisno o broju koji pročita provest će određenu akciju. Jedini dodatni posao simulacije koju bismo izvodili na računalu bio bi da piše u navedene datoteke pozicije na koje se mora kretati svaki od robota. Zbog vremena potrebnog da se podatak zapiše na disk i pročita s njega novi korak u igri provodili svake 4 sekunde. Dolje navodimo dio koda koji bi se izvršavao u robotima.

```
<?
    echo("2")
?>
```

Listing 4. index.php

```
while True:
  request_url = 'http://lovro-pc/mate/index.php'
  resp = requests.get(request_url)
  text = resp.text
  broj = int(text)
  if broj == 0:
    rocky.turn_left_by_degree(90)
  if broj == 1:
    rocky.turn_right_by_degree(90)
  if broj == 2:
    rocky.forward(50, 1)
  if broj == 3:
    rocky.backward(50, 1)
  if broj == 4:
    codey.speaker.play_melody('warning.wav')
          codey.emotion.dizzy()
          rocky.stop()
          codey.stop_all_scripts()
```

Listing 5. kretanje Codey Rockya

Kako je jedenje Pac-Mana jedno od pooznatijih obilježja igre potrudili smo se za to pronaći dostojnu zamjenu. Svaki Codey Rocky ima senzor trešenje. Tako da kada se lovci sudare s robotom koji glumi Pac-Mana, odnosno kada ga uhvate, pokrenut će se njegov senzor trešenje i Codey Rocky će ispustiti zvuk gašenja i ugasiti se. Dolje navodimo kod kako bismo to proveli.

```
@event.shaked
def on_shaked():
    codey.emotion.dizzy()
    rocky.stop()
    codey.stop_all_scripts()
```

Listing 6. kraj igre

V. Metode rješavanja problema

A. Q-learning

Jedan od načina kako se može naučiti kontrolna strategija je uporabom algoritma Q-learning. Osnovna ideja algoritma je koristiti Bellmanovu jednadžbu kao jednostavno ažuriranje iteracije vrijednosti korištenjem težinskog prosjeka stare vrijednosti i novih informacija.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, at) \right]$$

Dokazano je da za konačni Markovljev lanac Q-vrijednost uvijek konvergira [2]. Međutim, ovaj teorijski dokaz konvergencije ne znači da je konvergencija zagarantirana u praktičnim okruženjima, jer obično postoje ograničenja u vremenu treniranja modela. Prvi i osnovni problem ovog algoritma prilikom primjene na početni labirint je prevelik broj stanja. Primjerice u slici 5 slike A i B su nama jednake, ali algoritmu nisu i dva puta mora učiti istu stvar. Tako primjerice slike A i C razlikuje samo jedna točka i to je za algoritam jedna potpuno nova okolnost koju nije vidio i ponovno ju mora učiti.

B. Approximate Q-learning

Zato je ideja pokušati s određenim skupom značajki riješiti gore navedeni problem. Značajke su funkcije koje stanju pridružuju realne vrijednosti koje opisuju bitna svojstva stanja. Ideja je da svaka od značajki opisuje što općenitiji skup mogućih situacija, a ne da se koristi npr. značajka ušao u tunel koja bi se koristila samo u limitiranom broju slučaja. Prednost te ideje da iskustvo agenta možemo svesti u nekoliko bitnih brojeva, dok je nedostatak da određena stanja mogu dijeliti značajke, ali su u biti suštinski različita.

Dakle umjesto Q vrijednosti za svaki par akcije i stanja, ideja je koristiti vektor čije bi komponente bile značajke. Algoritam Approximate Q-learning pretpostavlja postojanje funkcije značajki $f(s, a)$ na skupu stanja i akcija koja za komponentne funkcije ima $f_1(s, a), f_2(s, a), \ldots, f_n(s, a)$. Approximate Q-learning funkcija poprima sljedeći oblik:

$$Q(s, a) = w_0 \cdot f_0(s, a) + w_1 \cdot f_1(s, a) + \cdots + w_n \cdot f_n(s, a)$$

Pri čemu je svaka težina w_i povezana s određenom značajkom f_i, a $f_0(s, a)$ je *bias* i iznosi 1. Težine se ažuriraju prema sljedećem pravilu:

$$w_i \leftarrow w_i + \alpha \cdot [r_t + \gamma \cdot \max_{a \in A}(Q(s_{t+1}, a)) - Q(s_t, a_t)] \cdot f_i(s, a)$$

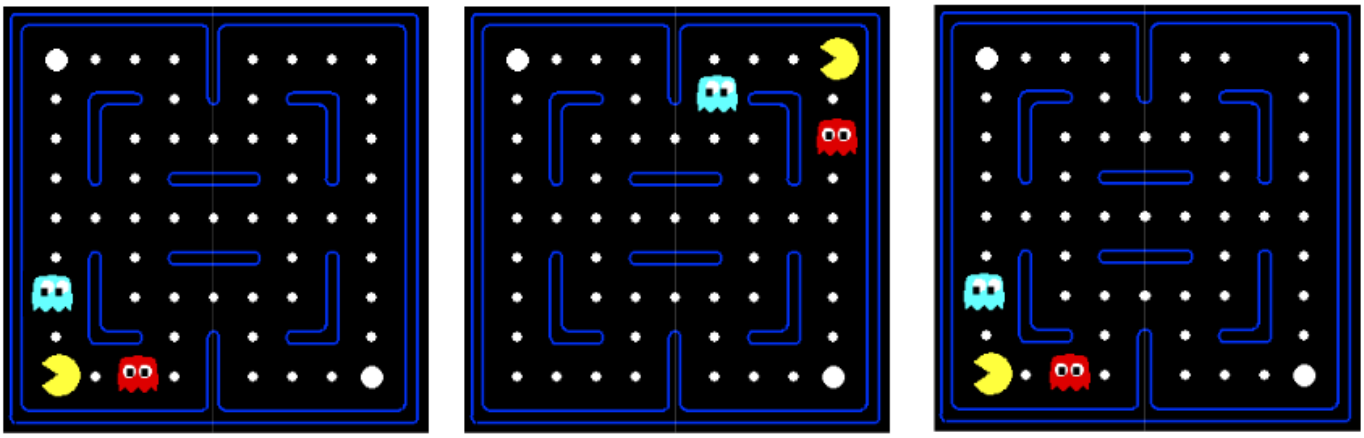
1) Izbor značajki

Metoda odabira značajki namijenjena je smanjenju broja ulaznih varijabli na samo one za koje se smatra da su najkorisnije za model u svrhu predviđanja ciljne varijable. Mnogi će modeli, posebice oni temeljeni na regresiji procijeniti parametre za svaku značajku u modelu. Zbog toga prisutnost ne informativnih varijabli može dodati neizvjesnost predviđanjima i umanjiti opću učinkovitost modela. Zato je odabir značajki prvenstveno usmjeren na uklanjanje ne informativnih ili suvišnih značajki iz modela.

Kroz rad na projektu odlučili smo da osnovni skup značajka F sadrži ove značajke

- broj lovaca na udaljenosti 1
- udaljenost od najbliže točkice

Na navedenim značajkama smo na početku rada na projektu proveli algoritam i dobili zadovoljavajuće rezultate. Također naglasimo kako smo ovaj skup nazvali osnovnim jer oduzimanjem jedne od značajki algoritam gubi u svakoj epizodi. Razmišljali smo koje značajke možemo dodati i na taj način



Slika 5. slike A, B i C

bolje opisati trenutnu situaciju u kojoj se Pac-Man nalazi. Tako smo se odlučili za sljedeći dodatni skup značajki.

- broj lovaca na udaljenosti 2
- postotak ukupno pojedenih točkica

Kada smo dodavali i micali svaku od ovih značajki skupu F nismo dobili ništa bolje niti lošije rezultate od početnih.
Udaljenosti smo implementirali pomoću algoritma A^* s heurističkom funkcijom Manhattan udaljenost[8].Opišimo onda kako navedeni algoritam funkcionira.

2) A^* algoritam

Za pojedini objekt lako je ostvariti pokret, ali pronaći putanju koja će kretanje svesti na minimum, a ostvariti optimalne rezultate je složeno. Želimo pretraživanje kojim ćemo brzo i uz minimalan broj pokreta doći do cilja.
Jedan od algoritama koji omogućuje pronalazak najkraćeg puta između dva čvora je Dijkstrin algoritam. Originalna verzija nizozemskog znanstvenika Edsgera W. Dijsktre tražila je najkraći put između dva čvora, ali češća je verzija u kojoj se traži najkraći put od početne pozicije do ostalih čvorova u grafu.
U svakom koraku proučavamo sve bridove koji spajaju trenutni vrh i sljedeći u grafu. Svaki brid ima težinu odnosno udaljenost, te zbrajamo sve težine do početnog vrha. Od svih bridova koje u jednom koraku možemo izabrati uzimamo onaj za koji je pripadni vrh, najmanje udaljen od početnog.
Ovaj algoritam omogućuje pronalazak najkraćeg puta, ali previše vremena troši na one puteve koji ne obećavaju dobar ishod. Također, u nekim problemima traženje rješenja neće biti optimalno jer nema informacije o tome gdje se točno ide i približavamo li se određenom cilju. Poboljšanje Dijkstrinog algoritma za tu informaciju je algoritam A^*.
Prema tome, algoritam A^* dolazi do cilja najkraćim mogućim putem, ali također odustaje od onih koji ne obećavaju dobar rezultat. Algoritam radi tako da dok god nije obradio ciljni čvor, uzima najbolji čvor iz liste otvorenih čvorova te iterira po njegovoj listi nasljednika. Ukoliko se nasljednik već nalazi u otvorenoj ili zatvorenoj listi provjeravamo je li nova putanja bolje od one koju imamo spremljenu. Ako je, brišemo ga iz lista te s novom cijenom putanje ubacujemo u otvorenu listu. Ukoliko je ovo prvo put da vidimo neki čvor jednostavno ga ubacimo u otvorenu listu.
Algoritam se zasniva na funkciji $f(n) = g(n)+h(n)$, pri čemu funkcija $g(n)$ predstavlja vrijednost težinu puta od početnog čvora do čvora n, a funkcija $h(n)$ procjenu minimalne težine puta od čvora n do nekog završnog čvora. Za procjenu te minimalne težine koristit ćemo heurističku funkciju.
Pseudokod koji će nam pomoći da bolje razumijemo algoritam bi izgledao ovako:

Stavi START u red (queue)

WHILE { red nije prazan
        i cilj nije postignut

DO { izračunaj novi trošak (težinu)
IF { težina novog čvora nije u listi
     ili je nova težina manja od one koji bi imali
ONDA { odredi novu težinu
       posloži red

Slika 6. Pseudokod

Dakle kod u Pythonu bi izgledao ovako:

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
  current = frontier.get()

  if current == goal:
    break

  for next in graph.neighbors(current):
    new_cost = cost_so_far[current] + graph.cost(
    current, next)
```

[8] $d(A,B) = |A_x - B_x| + |A_y - B_y| \quad A, B \in \mathbb{N} \times \mathbb{N}$

```
    if next not in cost_so_far or new_cost <
    cost_so_far[next]:
      cost_so_far[next] = new_cost
      priority = new_cost + heuristic(goal, next)
      frontier.put(next, priority)
      came_from[next] = current
```

Listing 7. Kod algoritma A^*

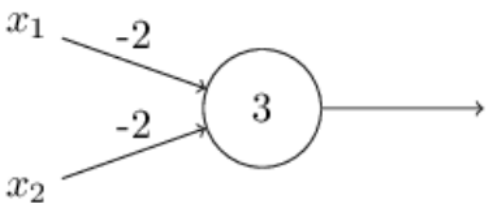
Usporedba algoritama najbolje je predočena na sljedećem primjeru:

Slika 7. Usporedba s Dijkstrinim algoritmom i Greedy pristupom

C. Deep learning

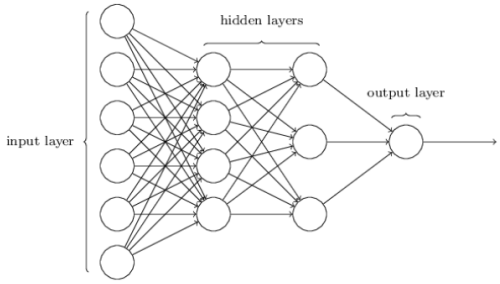
Prije samog opis rješenja problema ovim pristupom navedimo pojmove koje ćemo u nastavku rada koristiti.

Neuron je sastavna jedinica neuronske mreže. Sastoji se od ulazne vrijednosti, ulazne težine, *bias* i izlazne funkcije. Više nepovezanih neurona zovemo neuronskim slojem, dok povezane neuronske slojeve zovemo neuronskom mrežom. Neuron prikazan na slici 8 ima dvije ulazne vrijednosti x_1 i x_2. Za svaku ulaznu vrijednost ima po jednu ulaznu težinu $w_1 = -2, w_2 = -2$ dok je vrijednost *bias-a* 3. Izlazna funkcija f prima argument $x_1w_1 + x_2w_2 + bias$ i ono što funkcija vrati je izlaz neuron.



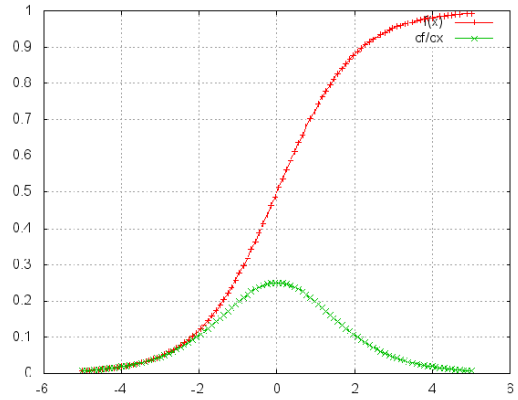
Slika 8. Prikaz neurona

Duboko učenje karakterizira mreža sa velikim brojem slojeva te sa manjim brojem neurona u slojevima. Prvi sloj se zove ulazni sloj, zadnji sloj se zove izlazni sloj, dok se ostali slojevi u sredini zovu skriveni slojevi (engl. *hidden layers*). Neuronska mreža se može poistovjetiti sa crnom kutijom (engl. *black box*). Njezin je cilj da nelinearnim transformacijama iz vektorskog prostora u vektorski prostor što točnije preslika ulazne vrijednost u izlazne vrijednosti iz skupa za učenje.



Slika 9. Prikaz neuronske mreže

Neuronska mreža kako bi se iz iteracije u iteraciju poboljšavala, mora svoje parametre podešavati sa ciljem smanjenja iznosa funkcije pogreške. Danas najpopularniji način je algoritam propagacije pogreške unatrag (engl. backpropagation). Metoda backrpropagation poznata je već od 1986 [15], dok duboko učenje tek u današnjem desetljeću dobiva na popularnosti. Razlog zašto duboko učenje nije dobilo na važnosti više od 20 godina je zbog toga što su mreže sa velikim brojem slojeva davale jako loše rezultate. Razlog tomu je što bi gradijent isčeznuo (engl. *Vanishing gradient*) prije nego što bi došao do početnih slojeva ili drugim riječima početni slojevi uopće ne bi učili. Tada se za izlaznu funkciju koristila sigmoid funkcija[9]. Njezina karakteristika je da joj je derivacija simetrična oko 0 i da joj limes u $\pm\infty$ teži prema 0 kao što je prikazano na slici 10. Upravo ova karakteristika priječila je učenje dubokih mreža. Problem isčeznuća gradijenta može se izbjeći promjenom arhitekture mreže, no najlakši način je da izlazna funkcija neurona postane ReLu funkcija. To je pokazano tek 2011. [16] i od tada je ReLu funckija postala standardom za izlazne funkcije.



Slika 10. Prikaz funkcije sigmoid i njezine derivacije

Derivacija ReLu funckije je ili 0 ili 1. Ukoliko je $x > 0$ onda povratni signal neće slabiti pri prolasku kroz mrežu, a

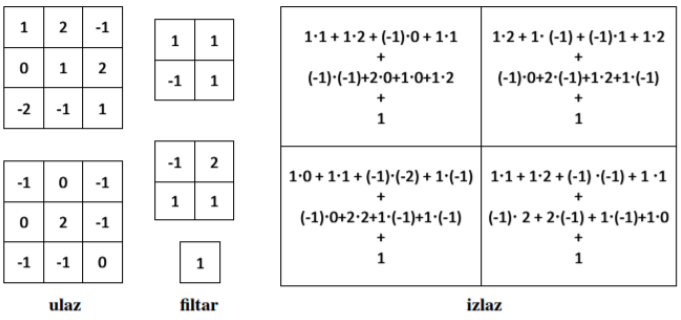
[9] $f : \mathbb{R} \to \mathbb{R},\ f(x) = \frac{1}{1+e^{-x}}$

ako je $x < 0$ onda će se signal zaustaviti jer je derivacija jednaka 0.

Jedna od prednosti DQN je u tome što Q- vrijednost za par stanja i akcija mogu biti interpolirane konvolucijskom neuronskom mrežom umjesto uobičajenog korištenja tablica podataka za spremanje Q-vrijednosti.
S obzirom da je ulaz u mrežu, matrica ili bilo koja topološka struktura, onda se koristi konvolucijska mreža. Ukoliko bi se koristila obična neuronska mreža, njezin bi zadatak bio da prvo nauči topologiju strukture zajedno sa preslikavanjem, dok je zadatak konvolucijskoj mreži naučiti samo preslikavanje. Zašto je topologija bitna, prezentira nam iduća situacija. Ukoliko se do Pac-Mana nalazi lovac, onda je izuzetno bitno da mreža zna takvu informaciju, a ona to može znati samo ako smo joj dali topološku strukturu reprezentacije trenutnog stanja. Budući da smo ovdje koristili riječ konvolucijska mreža opišimo kratko u nastavku što je to.

1) Konvolucijska mreža

Konvolucijske neuronske mreže mogu se opisati kao nadogradnja nad običnim višeslojnim unaprijednim mrežama. Konvolucijska, kao i obična, neuronska mreža sastoji se od jednog ulaznog, jednog izlaznog i jednog ili više skrivenih slojeva. Kod konvolucijskih neuronskih mreža specificni su konvolucijski slojevi i slojevi sažimanja. Konvolucijski sloj jedan je od glavnih dijelova svake konvolucijske neuronske mreže. Svaki konvolucijski sloj sastoji se od filtara koje sadrže težine koje je potrebno naučiti kako bi mreža davala dobre rezultate. Filtri su najčešće manjih prostornih dimenzija od ulaza, no uvijek su jednake dubine kao i ulaz. Pri unaprijednoj fazi potrebno je konvoluirati filtar s ulazom. Rezultat konvolucije je dvodimenzionalna aktivacijska mapa koja predstavlja odziv filtra na svakoj prostornoj poziciji. Zapravo mreža će naučiti težine unutar filtra kako bi se filtar aktivirao na mjestima gdje prepoznaje određena slikovna svojstva, kao na primjer određene vrste rubova ili slično. Kako bi objasnili rad konvolucijskog sloja potrebno je definirati dvije dvodimenzionalne matrice koje predstavljaju ulaz u konvolucijsku mrežu s dvije komponente (dubine 2). Kako se ulaz sastoji od dvije komponente, onda i dubina filtra mora biti jednaka 2. Kao i kod jednostavne neuronske mreže gdje je svaki neuron imao jedan dodatan ulaz koji je uvijek bio jedan i za kojeg je bila definirana dodatna težina (prag), tako i kod filtra postoji jedna dodatna težina za prag. Opisan primjer prikazan je na slici 11.



Slika 11. Primjer rada konvolucijskog sloja

U DQN ćemo koristi dva tipa neuronskih mreža: $Q-$mrežu i *target network*. Svaku trening iteraciju $Q-$ mreža svoje vrijednosti promjeni korištenjem ranije objašnjene metode back-propagation. Opišimo onda sada zašto nam treba target network.

2) Target network

Razlika između Q-learninga i DQN-a je u tome što se zamijeni funkcija točne vrijednosti s aproksimiranom funkcijom. Pomoću Q-learninga ažurira se točno jedna vrijednost stanja / akcije u svakom vremenskom koraku, dok s DQN-om ažurira više njih. Problem koji ovo uzrokuje je taj što možete utjecati na vrijednosti djelovanja za naredno stanje u kojem ćete se nalaziti, umjesto da im se garantira da su stabilnost kakva postoji u Q-učenju.

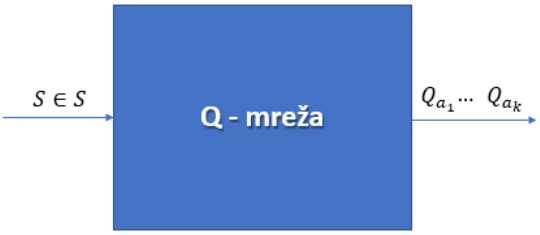
To se događa stalno sa DQN-om kada se koristi standardna dubinska mreža (hrpa slojeva iste veličine u potpunosti povezanih). Učinak koji se občino vidi kod ovoga naziva se katastrofalnim zaboravljanjem[10] i može biti poprilično spektakularan. Primjerice ako gledamo igru poput Pac-Mana s standardnom mrežom i pratimo prosječni rezultat tijekom posljednjih 100 igara, vjerojatno ćemo vidjeti lijepu krivulju rezultata, i tada sve odjednom to potpuno iskače i počinje opet donositi loše odluke, čak i kad α postane dovoljno malen. Krivulja učenja će postati skokovita i ovaj će se ciklus neprekidno nastaviti bez obzira koliko dugo pustili agenta da uči. Korištenje stabilne ciljane mreže kao mjere pogreške jedan je način borbe protiv tog učinka. Konceptualno to znači: „Imam ideju kako to dobro odigrati, isprobati ću ga malo dok ne nađem nešto bolje", za razliku od riječi: „Idem prekvalificirati sebe kako ću igrati nakon svakog poteza". Dajući mreži više vremena za razmatranje više radnji koje su se nedavno dogodile, umjesto da se stalno ažuriraju, nadamo se pronaći bolji model prije nego što ga počne koristiti za poduzimanje akcija.

3) Experience Replay

Praksa je pokazala da je bolje spremati iskustva agenta i onda slučajnim odabirom trenirati mrežu, nego trenirati mrežu na uzastopnim primjerima. Zbog toga mreže vide previše uzoraka jedne vrste i zaboravljaju ostale. Na primjer može se dogoditi da Pac-Man zaboravi naučeni početni dio igre kada dođe do završnog. Tako se onda slučajnim odabirom malo uči početni, središnji i završni dio igre. Spremanje cjelokupnog iskustva u međuspremniku omogućuje nam da treniramo na neovisnijim uzorcima. Ključni razlog korištenja memorije je prekid povezanosti uzastopnih uzoraka. Mi samo izvlačimo hrpu prijelaza iz međuspremnika nasumično i treniramo na tome. To pomaže razbiti vremensku povezanost uzoraka treninga.

Sada kada smo opisali sve bitne elemente mreže, prikažimo arhitekturu neuronske mreže za koju smo se odlučili.

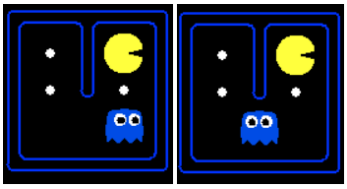
[10]Engleski izraz koji se koristi u struci je catastrophic forgetting.



Slika 12. Arhitektura neuronske mreže

D. Reprezentacija stanja

Kako smo ranije naveli u (1) Markovljevo svojstvo u Markovljevom lancu da je neposredna budućnost i prošlost uvjetno su nezavisne uz danu sadašnjost. Ako bi to primjenili u Pac-Man igri, to znači da ne trebamo u memoriji spremati sve prošle trenutke u igri. Već ćemo zato samo pohraniti zadnja dva stanja u igri. Svako stanje u igri jedinstveno je repezentirano je s pozicijom zida, hrane, lovca i Pac-Mana. Zato smo se odlučili da u svakoj matrici 0, odnosno 1, izražava postojanje elementa u odgovarajućoj matrici. Kao posljedica toga, svaki okvir sadrži lokacije svih elemenata igre prikazanih u tenzoru $A \times B \times 4$, pri čemu su A i B odgovarajuća širina i visina mreže u kojoj se Pac-Man kreće.
Na kraju je stanje predstavljeno s dva tenzora koji zajedno predstavljaju posljednja dva okvira, što onda rezultira ulaznom dimenzijom $A \times B \times 4 \times 2$. Pokažimo onda na primjeru slike 13. kako bi to izgledalo.



Slika 13. Reprezentacija stanja - primjer

$\begin{pmatrix} 1&1&1&1&1 \\ 1&0&1&0&1 \\ 1&0&1&0&1 \\ 1&0&0&0&1 \\ 1&1&1&1&1 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0 \\ 0&1&0&0&0 \\ 0&1&0&1&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&0 \end{pmatrix}$
zid	točke	Pac-Man	lovci
$\begin{pmatrix} 1&1&1&1&1 \\ 1&0&1&0&1 \\ 1&0&1&0&1 \\ 1&0&0&0&1 \\ 1&1&1&1&1 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0 \\ 0&1&0&0&0 \\ 0&1&0&1&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{pmatrix}$	$\begin{pmatrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&0&0 \end{pmatrix}$
zid	točke	Pac-Man	lovci

E. Implementacija

Q-mrežu i target network sadrže dva konvolucijska sloja koji se nadovezuju na dva potpuno povezana konvolucijska sloja. Prvi sloj koristi ranije navedenu aktivacijsku funkciju ReLu, dok zadnji potpuno povezani koristi linearnu aktivacijsku funkciju. Slojevi za objedinjavanje nisu potrebni zbog činjenice da razlučivost ulaza nije previše velika. Štoviše, objedinjavanje slojeva dovelo bi do gubitka lokacijskih podataka koji su, vjerojatno, ključni za odabir usmjerenih radnji u Pac-manu.
Za optimizaciju gradijentnog spuštanja koristili smo Adam optimizacijski algoritam opisan u [17]. Budući da Python unutar svoje standarne biblioteke nema implementaciju matrica, koristili smo poznatu Python biblioteku NumPy.
Za implementaciju CNN koristili smo Pythonovu biblioteku TensorFlow 2.2 [5], a zbog velikog broja stanja treniranje mreže smo pokrenuli na GPU[11] pomoću Google Colaba. Budući da preko Google Colaba nismo mogli pokrenuti naučeni model, svakih 1000 epizoda spremali bi naučeni model u tzv. checkpoint datoteku preko koje smo onda kasnije na računalu pokrenuli model i snimili dobivene rezultate. Programiranje na grafičkoj kartici zahtjeva vrlo veliku paralelizaciju, odnosno pokretanje velikog broja dretvi koje rade na vrlo sličnom zadatku. Operacije poput konvolucija ili množenja matrica najčešći su primjeri u kojima karakteristike grafičkog sklopovlja dolaze do izražaja i mogu značajno ubrzati izračun u odnosu na procesorsko sklopovlje.
Sada ćemo u sljedećem poglavlju raspraviti standardni problem u RL. Navedeno rješenje smo promijenili i u algoritmu Approximate Q-learning.

F. Problem - exploration vs exploitation

Već je spomenuto da u svakom stanju osim završnog stanja agent mora odabrati akciju. Nekoliko je načina na koji agent možete odlučiti koje će akcije poduzeti. Najjednostavniji od njih je pohlepni odabir: agent uvijek bira radnju koja ima najveću vrijednost.

$$a_t = \max_{a \in A} Q_t(Q, a)$$

Ova metoda je čista eksploatacija, ali postoji pojedini problem ako bi primijenili odmah na početku učenja. Naime može se dogoditi da agent se zavrti u petlji ili da uopće ne posjeti pojedini dio okoliša.
Zato je standardna ideja primijeniti tzv. ϵ-*greedy* strategiju. Na početku se fiksira $0 \leq \epsilon < 1$ i agent s vjerojatnošću ϵ izabrati slučajnu akciju koja možda ne mora nužno imati najveću vrijednost i na taj način agent upoznaje stanja koja do sada možda nije vidio. Analogno s vjerojatnošću $1 - \epsilon$ agent izabire najbolju moguću akciju. Mi se odlučili za početnu vrijednost $\epsilon = 1$ i spuštanjem dok agent uči sve dok ϵ ne padne do 0.1. Odlučili smo se da ϵ pada za 0.05 svaki put kada u treningu Pac-Man prođe 800 koraka. Uz napomenu da su brojevi rađeni za labirint Sunce, te da bi za možda veće labirinte trebalo odrediti nove brojke za koje bi smatrali da je agentu dovoljno da obiđe cijeli labirint. Naravno prilikom testiranja naučenog modela ϵ smo stavili na nulu.

1) Problemi

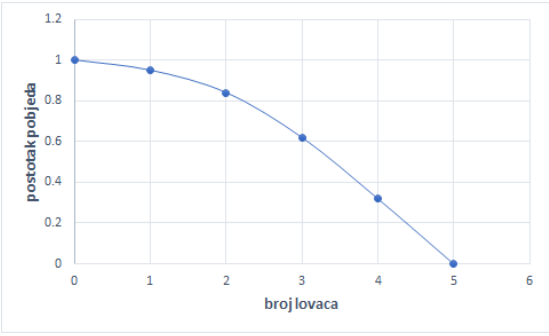
Nakon nešto više od tri tisuće epizoda, agent djeluje dobro na gore zadanom labirintu. Međutim, vidljiva je jasna razlika između ponašanja treniranog modela na početku igre i ponašanja u krajnjoj fazi igre. Model posebno ima bolje rezultate pri početku igranja. Zapanjujuće, nedostatak performansi

[11] Graphics processing unit - Grafička procesorska jedinica

u krajnjoj igri odnosi se i na zadaće koji se pojavljuju i na početku igre, kao i na kraju igre, kao što je izbjegavanje sudara s lovcima. Da bi riješili problem pada performansi tijekom igre, testirali smo nekoliko hipoteza da bismo razumjeli ove rezultate. Važno je napomenuti da je glavni značaj scenarija završetka igre to što je malen broj točkica u labirintu.
Dakle može se dogoditi da je zbog funkcije nagrade koju smo definirali na početku da je Pac-Manu na kraju igre isplativije zabiti se u lovca nego ići po zadnju točkicu u labirintu. Da bi testirali ovu hipotezu trenirani model smo postavili u labirint istih dimenzija, ali s raspršenim i manjim brojem točkica. Pac-Man je tada postigao značajnije manje rezultate (kretanje se čini više slučajnim) u situacijama krajnje igre u usporedbi sa situacijama na početku igre. Međutim, nije bilo značajnih dokaza da je namjerno išao u lovca.
Još jedan od problema koji smo promatrali je nedostatak promatranih scenarija krajnje igre koji možda je otežao donošenje odluka. Naime Pac-Man tijekom svojeg učenja se puno češće susreće s početnim nego završnim dijelom igre. Da bi provjerili je li to slučaj, vrijeme treninga značajno smo produljili i pokazalo se da veći broj trening epizoda pozitivno utječe na poboljšanje izvedbe završnog dijela igre. Ovaj je rezultat je u većoj mjeri u skladu s navedenom hipotezom da dulji trening osigurava da se algoritam promatra više završnih scenarija igre. Dakle moguće rješenje problema može biti u duljem treniranju modela.

G. Generalizacija modela

Proveli smo eksperimente za procjenu performansi modela kada se stavi u situacije koje nije se susretao tijekom treninga. Naša pretpostavka je bila da model neće dobro generalizirati na nove aspekte igre, poput različitog broja lovaca i različitih labirinta.
Dodavanje više lovaca u labirint dodatno otežava problem jer se povećavanjem broj duhova povećava se i broj radnji Pac-Mana koje mogu rezultirati porazom. Model koji je treniran s 2 duha ima dobre rezultate kada je suočen s većim brojem duhova. Ovo je očekivano ponašanje jer se težina problema povećava kad se doda još duhova. Unatoč tim nižim stopama dobitka, graf pokazuje da se može generalizirati u situacijama s više duhovi pobijedivši još uvijek značajnu količinu igara. Rezultati su prikazani na slici 14.



Slika 14. Graf koji pokazuje uspješnost s modela s različitim brojem lovaca

Za očekivati je da je broj lovaca 4 i više prevelik za dimenzije labirinta u kojem smo provodili testiranje, ali vidimo da je postotak na tri lovca ostao na visokoj razini.
Model koji je treniran na jednoj labirintu ne može generalizirati na drugim labirintima koje nikada nije vidio. Prilikom testiranja modela na novim labirintima, radnje koje obavlja agent blizu su slučajnih. Kao rezultat toga, Pac-Man ne uspijeva osvojiti nagrade i na taj način ne uspijeva pobijediti. U daljnjem istraživanju bilo bi zanimljivo istražiti može li se model generalizirati na više labirinta.

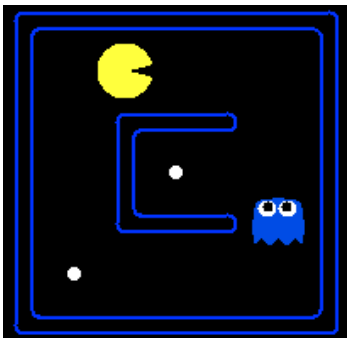
Zato smo problemu pokušali pristupiti na sljedeći način kojeg u nastavku opisujemo. Željeli smo koristeći stečena znanja u *Deep learningu* napraviti model koji bi se lakše mogao generalizarati na neviđene labirinte.

Zato smo uzeli sljedeći skup značajki
$\text{ת} = \left(\frac{1}{\text{udaljenost od lovca}}, \frac{1}{\text{udaljenost od točke}}, \text{postotak sakupljenih točaka}\right)$
Preciznije računa se recipročna vrijednost udaljenosti od najbližeg lovca i hrane zajedno s postotkom sakupljene hrane. Dakle napravili smo posebnu neuronsku mrežu kojoj je zadaća da izračunati Q vrijednosti stanja u koja Pac-Man može sljedeća otići. Mreža se sastojala od 2 sloja s linearnom i ReLu aktivacijskom funkcijom, dok smo težine mreže inicijalizirali nasumično iz skupa $\langle 0, 1]$. Navedeni su se rezultati pokazali jednako uspješnima kao i u slučaju Approximate Q-learninga.

VI. Rezultati

A. Q-learning

Algoritam smo proveli na manjem labirintu na kojem smo dobili rezultate koje smo i očekivali. Sa sve većim brojem epizoda model je na malom labirintu bio sve uspješniji i na kraju je uspješno savlado dani problem.
Algoritam smo pokrenuli i na labirinitu Sunce (slika 1), ali nakon samo 200 epizoda učenja program se srušio jer je potrošio dostupnu memoriju. Budući da detaljan prikaz rezultata



Slika 15. Manji labirint na kojem je uspješno proveden Q-learning

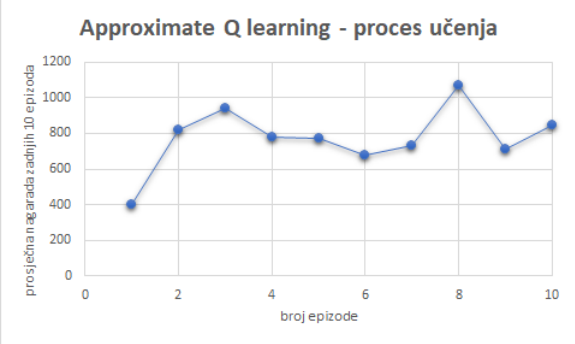
na malom labirintu nije u centru našeg projekta preskočit ćemo ga.

B. Approximate Q-learning

Iz donjeg grafa vidimo kako je primjena ovog algoritma na našem početnom labirintu Sunce puno učinkovitija nakon

samo 100 epizoda učenja. Rezultati prikazuju rast prosječnih nagrada svakih 10 epizoda i prikazujemo rezultate rješenja koja primijenjena na osnovnom skupu značajki F jer kao što smo ranije rekli nismo primijetili jasne razlike s dodavanjem drugih značajki.

Takvo drastično poboljšanje performansi pokazuje prednosti

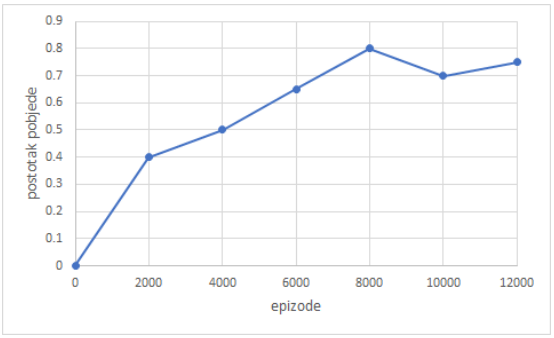


Slika 16. Rezultati na labirintu Sunce

definiranja eksplicitnih značajki. To bi mogli objasniti zbog činjenice da mnoga stanja mogu dijeliti mnoge značajke, što omogućava generaliziranje na neviđena stanja i čini ponašanje boljim. U suprotnom, agent mora istražiti svako stanje posebno igre tijekom treniranja prije nego što može dobro proći u testnoj fazi. Kako se prostor stanja eksponencijalno povećava u pogledu složenosti igre, veličina postavljenog treninga također mora eksponencijalno rasti prije nego što agent počne učiti kako pravilno igrati.

C. Deep learning

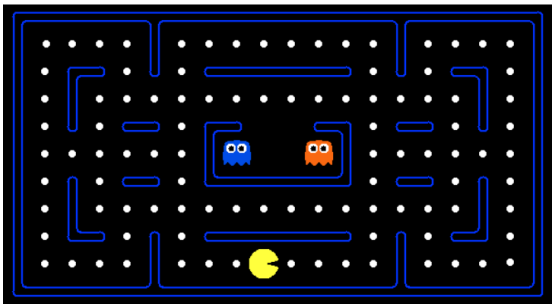
Na donjem grafu prikazujemo testiranje modela pomoću CNN na labirintu Sunce.



Slika 17. Graf testiranja modela CNN na labirintu Sunce

Pac-Man počinje pobjeđivati nakon otprilike 2000 epizoda treninga. Nastavkom treniranja modela nakon 2000 epizoda, polako se povećava postotak pobjeda.

Neuronsku mrežu koju smo razvili da nauči na jednom labirintu, a testiramo na drugom smo trenirali na labirintu Sunce, a testirali na labirintu koji se nalazi na slici 18. Nakon samo 100 epizoda model s 75% uspješnosti uspijeva naučenu kontrolnu strategiju primijeniti na drugom labirintu.



Slika 18. Labirint na kojem smo testirali ranije naučeni model

Za kraj ovog poglavlja s rezultatima navodimo video u kojem se pokazuje kako agent uči. Prvo se u video pokazuje rezultati Approximate Q-learninga, a onda i od CNN. Link na video rješenje.

VII. Mogući nastavak istraživanja

Jedno od bitnih stvari koje bi željeli završiti na projektu je dodati mogućnost vizualne prezentacije igre Pac-Man pomoću robota Codey Rocky. Da bi taj problem riješili trebamo kalibrirati robote i njihovo kretanje na jednoj podlozi. Nadamo se da ćemo navedeni problem stići riješiti kako bi ga sljedeće akademske godine mogli prezentirati.

Od različitih ideja koja se isplati implementirati u projekt u budućnosti zasigurno je dodavanje prioritetnog experience replaya[12], odnosno odabirom uzoraka iz kojih agent može najviše naučiti.

U labirintima u kojima je manji i raspršeniji broj točaka vrijedi razmisliti o drugoj funkciji nagrade. Naime ako su točkice daleko od Pac-Mana pitanje koliko će mu dugo trebati da shvati da je jedenje točkica to što treba raditi. Zato bi onda bila ideja davati nagrade već pri samom približavanju točkici, a ne samo prilikom jedenja iste.

Ideja je projekt u budućnosti iskoristiti kao pokazni primjer u nastavi informatike u osnovnoj ili srednjoj školi prilikom obrade nastavnih jedinica programiranja.

VIII. Zaključak

Za ovu izradu ovog rada proučavali samo potrebna znanja iz područja učenja podrškom i dubokog učenja te kako bih pristupili neuronskim mrežama kao alternativi klasičnim regresijskim i klasifikacijskim algoritmima. Sigurno postoji više različitih regresijskih metoda od linearne aproksimacije funkcija, ali onda bilo bi potrebno duže vrijeme i više stručnog znanja za njihovu implementaciju. Umjetne neuronske mreže ili bolje rečeno njihove biblioteke su poput kutija s gotovim alatima, i nažalost moguće je izgraditi mrežu bez pretpostavke poznavanja bitnih detalja. To je donijelo veće mogućnosti za razvoj struke, ali nažalost omogućilo ulazak ljudi koji bez potrebe za razumijevanjem što koriste rješavaju svakodnevno probleme. Nama je izazov u korištenju takvih mreža bio dovoljno razumjeti njihove komponente i steći dovoljno iskustva

[12]Engleski izraz koji se koristi u struci je prioritized experience replay.

za određivanje veličine mreže, odabir pravih komponenti za rješavanje pojedinog problema.

Iako su dobiveni pozitivni rezultati s neuronskim mrežama, postojali su tehnički problemi s kojima smo se susreli. Njihov veliki broj različitih komponenti i veličina onemogućili su nam uklanjanje pogrešaka. Ako bi se mreža ponašala čudno ili ne bi učila ispravno, sve što smo mogli učiniti bilo je promatranje ponašanja i stvaranje pretpostavki što možda nije u redu. Ovaj pristup pokušaja i pogreški, ovisno o složenosti sustava oduzelo nam je mnogo vremena. Za niz eksperimenata, odnosno potrebno vrijeme jednog treninga koji se odvijao bilo je potrebno od četiri do gotovo osam sati, ovisno o količini epizoda. Iz tog razloga bih istaknuli da znanje u ovom području još uvijek napreduje velikim koracima i da ćemo možda u budućnosti bolje moći iskoristi znanje stečeno u izradi ovog projekta.

Za kraj se samo želimo zahvaliti profesoru Goranu Igalyu na pomoći oko projekta i kolegici Ani Dugandžić na pomoći oko provođenja eksploratorne analize.

DODATAK

Kako smo u uvodnom dijelu naglasili da smo naišli na manjak aktivnosti ovdje navodimo primjer nastavne aktivnosti koja bi se mogla provesti na satu informatike i fizike u srednjoj školi.

A. Glavni cilj nastavne aktivnosti

Učenici će analizirati jednoliko pravocrtno gibanje.

B. Očekivana učenička postignuća

Učenici će:

- odrediti i objasniti srednju brzinu, pomak, put i vrijeme,
- matematički opisati jednoliko pravocrtno gibanje,
- grafički prikazati jednoliko pravocrtno gibanje s-t i v-t grafovima,
- iz grafičkog prikaza interpretirati podatke,
- izraditi projektnu dokumentaciju.

C. Međupredmetne teme

- osr C.3.2. Prepoznaje važnost odgovornosti pojedinca u društvu. Odgovorno se ponaša u zajedničkim aktivnostima.
- osr B.4.2. Suradnički uči i radi u timu. Objašnjava i služi se vještinama korisnima za timski rad.
- uku D.4/5.2. 2. Suradnja s drugima. Učenik ostvaruje dobru komunikaciju s drugima, uspješno surađuje u različitim situacijama i spreman je zatražiti i ponuditi pomoć. Može preuzeti različite uloge u skupini, dijeli ideje, sudjeluje u postavljanju ciljeva i njihovu ostvarivanju.
- pod C.4.1. i 4.2. Sudjeluje u projektu ili proizvodnji od ideje do realizacije (nadovezuje se i uključuje elemente očekivanja iz 3. ciklusa).
- ikt C.4.3. Učenik samostalno kritički procjenjuje proces, izvore i rezultate pretraživanja, odabire potrebne informacije.
- ikt C.4.4. Učenik samostalno i odgovorno upravlja prikupljenim informacijama.
- ikt D.4.1. Učenik samostalno ili u suradnji s drugima stvara nove sadržaje i ideje ili preoblikuje postojeća digitalna rješenja primjenjujući različite načine za poticanje kreativnosti.

D. Nastavni oblici

Diferencirana nastava u obliku rada u timu i frontalna nastava.

E. Nastavne metode

Prema izvorima znanja: metoda dijaloga, metoda demonstracije, metoda rada na zadacima.
Prema oblicima zaključivanja: metoda analize i sinteze, metoda analogije, heuristička metoda.

F. Nastavna sredstva

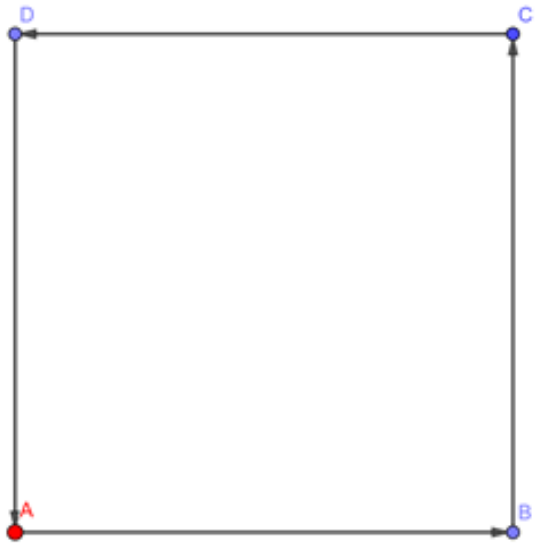
Codey Rocky, metar, štoperica, pisaći pribor, radni listić, 4 oznake u boji.

G. Tijek aktivnosti

Aktivnost je namijenjena za nastavu fizike u prvom razredu srednje škole kada se radi jednoliko pravocrtno gibanje kao aktivnost primjene teorije odrađene na prethodnim satovima. Tema se obrađuje prije nego su učenici upoznati s otporom podloge i sličnih dodatnih uvjeta koji mogu utjecati na brzinu. No u ovom slučaju učenici će uspoređivati svoje podatke dobivene na temelju različitih postotaka baterije Codey Rockyja pa će i rezultati biti različiti. Svaki Codey Rocky će biti programiran tako da put završi u položaju iz kojeg je krenuo, a put će opisati kao kvadrat s jednakim stranicama. Kao što je prikazano na slici Codey Rocky kreće iz točke A i opisuje put kao na donjoj slici. Učenici će dobiti radni listić koji će ih navoditi na potrebna mjerenja i poslužiti kao uvod u izradu projektne dokumentacije. Prvi zadatak za učenike je provjeriti jesu li dobili potrebna sredstva za izradu projektnog zadatka, nakon potvrde da imaju sav potrebni materijal može se krenuti na projektni zadatak. Postaviti Codey Rockyja u početni položaj koji označe jednim od papirnih oznaka. Pokrenuti robot i postaviti preostale oznake. Oznake imenovati. Učenici će izmjeriti vremena na intervalima, duljine stranica kvadrata. Odrediti put i pomak, srednje brzine itd... Na kraju će sve to prikazati pomoću s-t i v-t grafa.

H. Radni listić

Provjerite potrebni materijal za izradu projektnog zadatka: Codey Rocky, 4 papirnate oznake u boji, metar, štoperica, pisaći pribor.
Zadatak 1. Postavite jednu od papirnih oznaka na mjesto koje ćete koristiti kao početni položaj Codey Rockyja te ga postavite na tu oznaku tako da mu je sredina tijela točno iznad oznake. Pokrenite Codey Rockyja. Na mjestima gdje se rotira za 90° postavite i ostale 3 oznake. Oznake imenovati



Slika 19. Mapa kretanja Codey Rockya

slovima A, B, C i D redom krenuvši od početne.
Napomena: pazite na mjerne jedinice.

Zadatak 2. Ponovite prethodni postupak da biste potvrdili pravilnost oznaka te koristeći štopericu kojom ćete odrediti sljedeće intervale: t – potrebno vrijeme od kretanja iz početnog položaja pa do povratka u početni položaj, t_1 – vrijeme potrebno za prelazak puta od točke A do točke B, t_2 – vrijeme potrebno za rotaciju u točki B, t_3 – vrijeme potrebno za prelazak puta od točke B do točke C, t_4 – vrijeme potrebno za rotaciju u točki C, t_5 – vrijeme potrebno za prelazak puta od točke C do točke D, t_6 – vrijeme potrebno za rotaciju u točki D, t_7 – vrijeme potrebno za prelazak puta od točke D do točke A, t_8 – vrijeme potrebno za rotaciju u točki A. Ovdje ispišite redom vremena:

Zadatak 3. Codey Rocky je opisao put u obliku kvadrata. Zato izmjerite udaljenost od točke A do točke B, a drugim udaljenostima pridružite izmjerenu udaljenost, jer su točke A, B, C i D vrhovi kvadrata. Nacrtajte taj kvadrat ovdje:

Zadatak 4. Odredite prijeđeni put i pomak u svakom vrhu kvadrata.

Zadatak 5. Odredite srednje brzine $\overline{v_1}, \ldots, \overline{v_8}$ Codey Rockyja u vremenskim intervalima t_1, …, t_8. Za put na intervalima koristite izmjerene udaljenosti u prethodnom koraku, tamo gdje se Codey Rocky rotira put iznosi 0m.

Zadatak 6. Odredite srednju brzinu $\overline{v}$ Cody Rockyja u vremenskom intervalu t.

Zadatak 7. Nacrtajte s-t graf.

Zadatak 8. Nacrtajte v-t graf.

LITERATURA

[1] Ministarstvo znanosti i obrazovanja, *Odluka o donošenju kurikuluma za nastavni predmet Informatike za osnovne škole i gimnazije u Republici Hrvatskoj*, NN, Zagreb, 2019
[2] C. J. Watkins, P. Dayan, *Machine learning*, 1992
[3] C. J. Watkins, Doktorski rad: *Learning from Delayed Rewards*, King's College, 1989
[4] Berkeley Pacman project, `http://ai.berkeley.edu/project_overview.h`
[5] Google, Tensorflow, `https://www.tensorflow.org/`, 2019
[6] R. Madhav, *An Implementation of Pacman game using robots*, Indian Journal of Computer Science and Engineering. 2011
[7] F. Dominguez-Estevez, *Training Pac-Man bots using Reinforcement Learning and Case-based Reasoning* ,Complutense University of Madrid, 2017
[8] L. Bom, R. Henken, M. Wiering, *Reinforcement learning to train ms. pac-man using higher-order action-relative inputs. In: ADPRL. pp. 156–163. IEEE*, 2013
[9] S. Šegović, materijali za kolegiji Duboko učenje, `http://www.zemris.fer.hr/~ssegvic/du/`
[10] A. Krizhevsky, I. Sutskever, G. E. Hinton, *Advances in neural information processing systems*, 2012
[11] V. Mnih, K. Kavukcuoglu, D. Silver i dr., Playing atari with deep reinforcement learning, 2013
[12] J. Peters, R. Calandra, G. Neumann, *Deep Learning for Reinforcement Learning in Pacman*, 2014
[13] D. Kopljar, *Konvolucijske neuronske mreže, završni rad*, FER, 2016
[14] Z. Vondraček, *Markovljevi lanci, predavanja*, PMF, 2012
[15] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J., "Learning representations by back-propagating errors", 1986
[16] X. Glorot, A. Bordes and Y. Bengio, Deep sparse rectifier neural networks (PDF). AISTATS. "Rectifier and softplus activation functions. The second one is a smooth version of the first.". 2011
[17] D. P. Kingma i J. Ba,*Adam: A Method for Stochastic Optimization*, 2015.
[18] Vladimir Paar i suradnici, *Fizika oko nas 1*, udžbenik fizike u prvom razredu gimnazije, Školska knjiga, 2019.