

Koroška cesta 46  
2000 Maribor, Slovenija

**Krmiljenje semaforjev z okrepitvenim učenjem**

Seminarska naloga pri predmetu   
Umetna Inteligenca

Celje, maj 2020 Žiga Sušin

**Kazalo vsebine**

[1 Uvod 3](#_Toc41599890)

[2 Izdelava programa 4](#_Toc41599891)

[**2.1 O temi 4**](#_Toc41599892)

[**2.2 Priprava okolja 4**](#_Toc41599893)

[**2.3 Predstavitev izdelanega programa 4**](#_Toc41599894)

[**2.3.1 Razred DQNAgent 5**](#_Toc41599895)

[**2.3.2 Razred Sumo 6**](#_Toc41599896)

[**2.3.3 Funkcija Main 6**](#_Toc41599897)

[3 Simulacija in rezultati 7](#_Toc41599898)

[4 Literatura 9](#_Toc41599899)

**Kazalo slik**

**Slika 1: GUI programa SUMO 4**

**Slika 2: aktivacijska funkcija ReLU 5**

**Slika 3: križišče v poskusu 6**

**Kazalo grafov**

**Graf 1: Nagrada v odvisnosti od epizode 7**

**Graf 2: Skupen čas čakanja vozil v odvisnosti od epizode** **8**

# Uvod

Okrepitveno učenje je področje strojnega učenja, pri katerem poskušamo program oz. agenta, priučiti kako se odločati v določenih situacijah. To dosežemo s pomočjo sistema nagrajevanja oz kaznovanja.  
Takšne agente uporabljamo v različnih disciplinah (teorija iger, teorija nadzora, teorija informacij, stimulacijsko podprta optimizacija …)  
Vsem postopkom okrepitvenega učenja je skupno, da se agent mora odločati na podlagi interakcije z okoljem (problemskim prostorom), tako da zaporedje njegovih odločitev doseže ciljno stanje. Cilj RL postopka je maksimizacija kumulativne vsote nagrad in s tem optimalni vedenjski vzorec v danih pogojih.

Okrepitveno učenje je sestavljeno iz več elementov:

* Agent: je element, ki uporablja metodo okrepitvenega učenja in se odloča
* Stanje okolja: trenutni opis okolja, na podlagi katerega se agent odloča. Ponavadi je podano v obliki polja števil. [0,1,2,3]. Pri tem števila opisujejo stanje. Npr. slika 10x10 pikslov bo pretvorjana v polje dolžine 100, kjer vsako mesto predstavlja 1 pixel.
* Akcije: Nabor akcij, ki jih lahko agent v vsaki ponovitvi sprejme.
* »Nagrada«: Agent je po vsaki izvedeni akciji pozitivno ali negativno nagrajen. Na podlagi te nagrade agent oz. program ve, ali je bila akcija, ki jo je izvedel v danem trenutnu optimalna.
* Strategija: Cilj okrepitvenega učenja je najti vedenjsko strategijo, ki za vsako stanje in akcijo maksimizira nagrado. Ta strategija se med učenjem spreminja, glede na količino prejetnih nagrad.

Okrepitveno učenje je opredeljeno kot MDP (Markov decission process).   
Model je določen kot skupek stanj, ki posredujejo informacije o okolju, ki ga modelirajo.  
MDP se zapiše kot

* S : Množica vseh stanj
* A(s): Množica možnih dejanj v stanju s
* : verjetnost prehoda v stanje s', če se nahajamo v stanju s in izvedemo dejanje a
* : Nagrada

# Izdelava programa

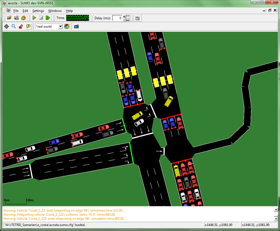
## O temi

Število avtomobilov na cestah hitro narašča, zato se pogosto, zlasti v prometnih središčih, srečujemo s prometnimi zastoji. V večini križišč poteka upravljanje z uporabo svetlobnih naprav (semaforjev). Problemi, kot so zamude potnikov in večje onesnaževanje zraka, nastanejo zaradi neučinkovitega upravljanja teh naprav. Večina obstoječih semaforjev je upravljana s pomočjo programov, ki ne upoštevajo prometa v realnem času. Taki programi nastavijo semaforjem enake čase v vsakem ciklu. To deluje v primeru, da je promet konstanten iz vseh strani. V primeru nepredvidljivih prometnih situacij (dogodki v bližini) se promet poveča in takrat so pomanjkljivosti takšnih krmilnih sistemov še bolj opazne. Te probleme bi se dalo odpraviti z implementacijo sistema, ki se sproti uči in posodablja.

## Priprava okolja

Program je v celoti napisan v programskem jeziku Python 3.7 in z uporabo nekaterih dodatnih knjižnic:   
Tensorflow 2.2 (odprtokodna knjižnica za strojno učenje),   
NumPy (Knjižnica, ki omogoča delo z večdimenzionalnimi polji),  
Keras (Knjižnica iz tensorflowa. Omogoča enostavno ustvarjanje nevronskih mrež).  
Traci (Knjižnica, preko katere upravljamo z programom SUMO)

Za simulacijo okolja je uporabljen program SUMO – Simulator of Urban Mobility.  
Sumo je dokaj preprost, a zelo uporaben simulator prometa. V njem preprosto ustvarimo ceste oz. povezave, križišča, semaforje. V posebnem dokumentu nato definiramo ves promet, vrsto avtomobilov lahko ustvarimo tudi sami (jim določimo hitrost, velikost).



Slika : GUI programa SUMO

## Predstavitev izdelanega programa

V tem delu naloge bom predstavil delovanje programa.   
Program je sestavljen iz 2 razredov, DQNAgent, kjer je nastavljena vsa logika agenta. Tu so določeni vsi parametri in definirana nevronska mreža. Razred Sumo, ki skrbi za generiranje prometa in spremljanje stanja prometa.  
Q-funkcija izgleda takole   
Nagrada je po tej formuli vedno enaka trenutni nagradi + največji možni nagradi za stanje s', akcijo a'.

Za popravljanje uteži uporabljamo RMSprop algoritem.

### Razred DQNAgent

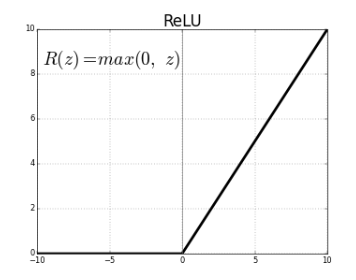
Agent ima na začetku definirane vse parametre, ti so:

**Gamma:** določa pomembnost prihodnjih nagrad. Pri 0 ne »gleda« naprej in upošteva samo trenutno nagrado, pri večjih vrednostih bolj upošteva ostale nagrade.

**Epsilon:** določa razmerje med raziskovanjem in izkoriščanjem že priučenega. Na začetku je epsilon 1 nato pa se po epizodah zmanjšuje.

**Stopnja učenja:** Določa kolikšen delež novo pridobljenih informacij (uteži), zamenja stare. Pri faktorju 1, agent uporablja samo znanje, ki ga je pridobil nazadnje.

**Model:** ustvari model nevronske mreže. Nevronska mreža kot vhod sprejme 3 polja. 1 vhod je polje 12x12. To polje vsebuje pozicije vozil v celicah na cesti. 2 vhod je polje 12x12. To polje vsebuje hitrosti vozil v posameznih celicah. 3 vhod je polje 2x1 in vsebuje stanje semaforja.  
Notranji plasti imajo 128 in 64 nevronov. Vse te plasti uporabljajo aktivacijsko funkcijo ReLU (Rectified Linear Unit).

Funkcija ReLU je najbolj uporabljena aktivacijska funkcija. Odstrani delo z negativnimi vrednostmi, pozitivna pa ohrani   


Slika : aktivacijska funkcija ReLU

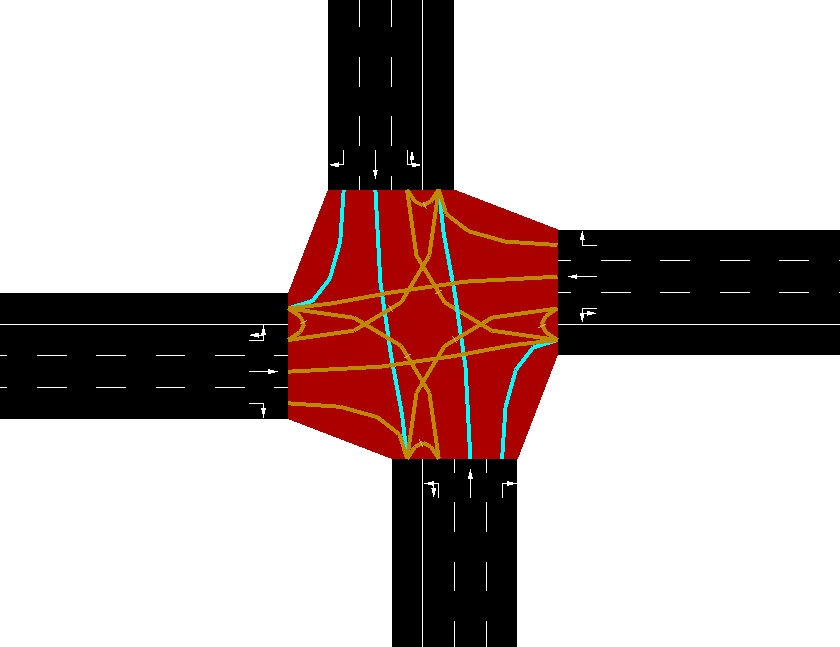
Razred DQNAgent ima še nekaj drugih spremenljivk, npr. memory, ki hrani podatke o utežeh. Te se potem serializirajo v zunanjo datoteko. Spremenljivka allActions pa hrani vsoto vseh mogočih akcij tj. 2.

Imamo še funkcije **save()** in **load()**, ki shranita oz. naložita uteži v memory iz datoteke.  
Funkcija **act(size)**, ki izbere naslednjo akcijo. Na podlagi epsilona se najprej odloči ali bo šlo za »exploration« ali »exploitation«. V primeru raziskovanja, naključno izbere naslednjo akcijo, v nasprotnem primeru pa model izračuna in vrne najbolj primerno akcijo.  
Funkcija **lrn(batch\_size)** pa nauči model z trenutnim stanjem in znanjem v memory.  
Funkcija **remember(state, action, reward, next\_state, done)** v memory zapiše vse vhodne parametre.

### Razred Sumo

Razred Sumo služi kot pomoč pri generiranju datoteke za simulacijo.   
Prva funkcija **generate\_sumo()**, se pokliče na začetku izvajanja programa. Najprej generira vse 4 krake ceste, potem pa ustvari še avtomobile. Ti so ustvarjeni v naključnem vrstnem redu. Vsi avtomobili pa imajo enake lastnosti. (hitrost, dolžina, pospešek).  
Druga funkcija, ki jo razred vsebuje pa je **get\_state().** To je funkcija, ki se kliče v začetku in koncu vsake epizode. Vrne nam stanje prometa v obliki polja na nasleden način   
[položaj vozil, hitrost vozil, luči na semaforjih].

Ceste in križišče so v poskusu generirane na naslednji način:



Slika : križišče v poskusu

### Funkcija Main

Funkcija main je funkcija, ki se kliče ko začenemo program. Vsebuje naslednje parametre:

Episodes: določa število epizod, ki se v vsakem učenju izvede.

Epsilondecay: Določa spremembo spremenljivke epsilon, po vsaki epizodi.

Ta funkcija gre v zanko tolikokrat, kot določimo z spremenljivko episodes.   
Na začetk vsake epizode najprej zaženemo SuMO in nastavimo semafor na začetno fazo. Potem pa gremo v zanko while kje ostanemo, dokler simulacije ne zapusti zadnji avtomobil. Na začetku vsakega koraka pogledamo stanje na križišči, potem izvedemo akcijo in preštejemo nagrado. Potem damo agentu v »spomin« stanje, izvedeno akcijo, nagrado in naslednje stanje. Na koncu pa s podatki iz spremenljivke memory agenta »učimo«.

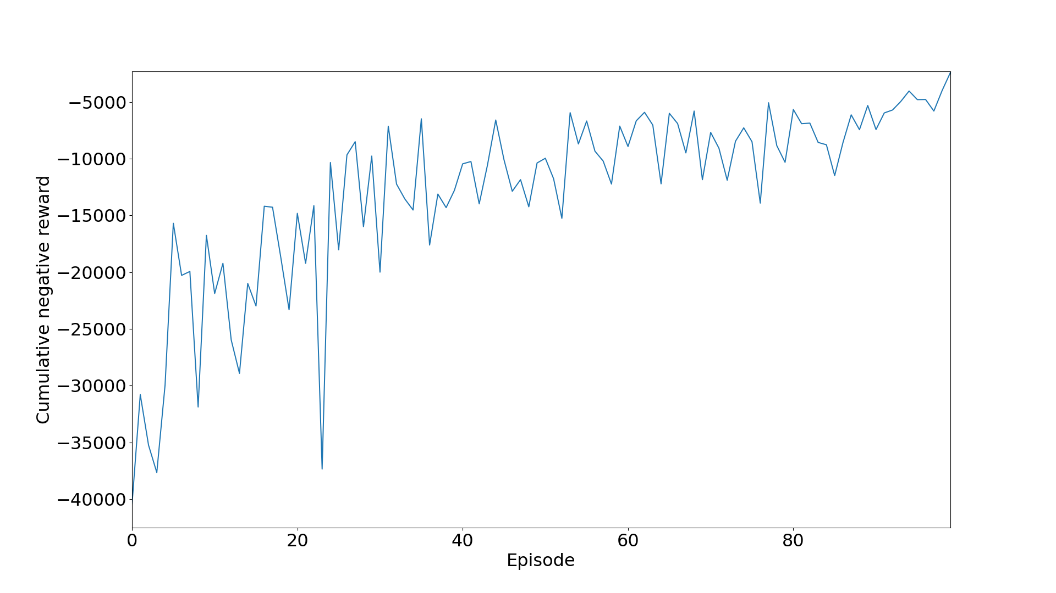
# Simulacija in rezultati

V zadnjem delu naloge sem problem večkrat simuliral z različnimi parametri in na tak način probal priti do najboljše rešitve. Rezultati so predstavljeni s 3 grafi.  
1. graf nagrade v odvisnosti od epizode.  
2. graf povprečnega časa čakanja vozil.

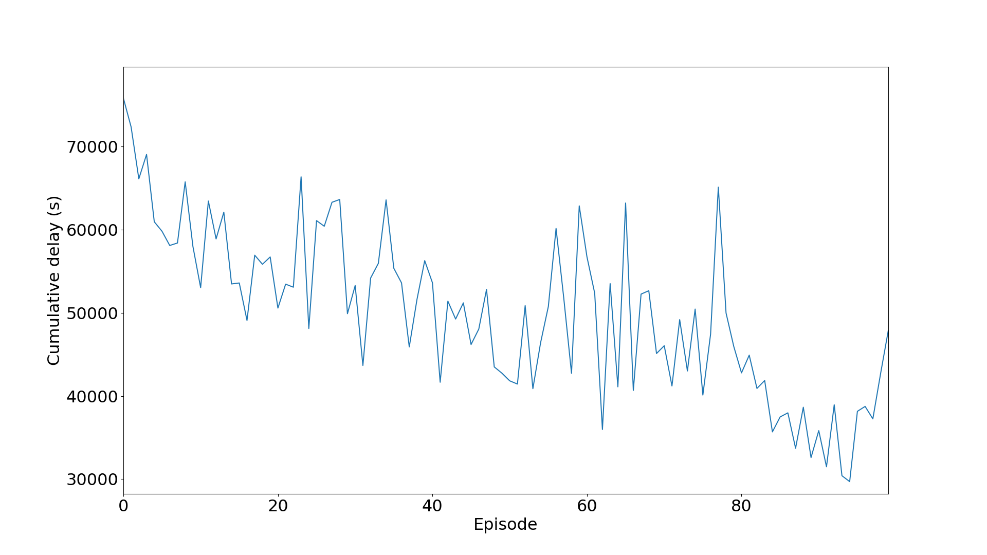
Parametri, ki sem jih spreminjal so: gamma, stopnja učenja.   
Epsilon sem v večini primerov zmanjševal procentualno glede na število epizod.

Pri čemer je e = zaporedna vrednost trenutne epizode.

Najboljši rezultat je bil dosežen z naslednjimi nastavitvami:

gamma = 0,95  
stopnja učenja = 0,0002

Graf : Nagrada v odvisnosti od epizode



Graf : Skupen čas čakanja vozil v odvisnosti od epizode

V grafu te simulacije je jasno razvidno zviševanje nagrade skozi epizode. V drugem grafu pa upadanje časa čakanja avtomobilov.

# Literatura

1. T. Patel, R. Cook, A Q-Learning Approach to Traffic Light Signal Control, (2018)

2. L., Xiaoyuan, D. Xusheng, W. Guiling, F. Z. Han, Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks(2018)

3. S. S. Mousavi, M. Schukat, Traffic Light Control Using Deep Policy-Gradient and Value-Function Based Reinforcement Learning (2017)

4. Reinforcement Q-Learning from Scratch in Python with OpenAI Gym, LearnDataSci.com, <https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/>