Autonomous Car

Echipa: RL Overflow
Membri:
- Constantin Gabriel-Adrian
- Hernest Mihai
- Richiteanu Mihai-Sebastian
- Sociu Daniel

In acest proiect am tratat tema utilizarii Reinforcement Learning in dezvoltarea masinilor autonome. Pentru aceasta am utilizat ca environment CarRacing-v0, acest environment fiind o interfata GYM.

- **Environment**

CarRacing-v0 ne ofera o viziune top-down asupra environment-ului unei masini de curse aflata pe un circuit. Acesta este un task continuu in care trb sa invatam din pixeli, state-ul consistand dintr-o imagine de 96x96. Circuitul este impartit in mai multe tile-uri, episodul finalizandu-se atunci cand toate tile-urile au fost vizitate.
Conform documentatiei GYM, exista doua tipuri de reward-uri returnate de environment. Reward-urile pozitive sunt primite atunci cand vehiculul progreseaza pe traseu. Reward-ul returnat per tile este de 1000/(numarul de tile-uri totale). Reward-ul negativ este -0.1 pentru fiecare frame.
In partea de jos a ecranului se gasesc scorul, viteza de deplasare, 4 senzori ABS, pozitia volanului si giroscopul.

- **Ideea proiectului**

Avand in vedere ca avem de a face cu un environment continuu vom folosi imaginile returnate de acesta pentru a antrena un model CNN care sa decida ce actiune trebuie aplicata in fiecare frame. Astfel, ca paradigma vom folosi Deep Q-Networks, in care vom folosi ca date un istoric al actiunilor alese care antreneaza modelul folosind un algoritm bazat pe ideea de Q-Learning.

- **Metoda de rezolvare a task-ului**

Am folosit o arhitectura CNN pentru a rezolva problema de obtinere a actiunii primind state-ul ca input, acest model fiind echivalent cu ceea ce face Q-learning-ul.
In primul rand obtinem state-ul din environment (care reprezinta frame-ul curent al environment-ului CarRacing) si il prelucram, convertind-ul la grayscale si renuntand la portiunea neagra din partea de jos a frame-ului.
De asemenea actiunile sunt definite ca fiind continue, deci pentru a avea un model functional, ar trebui sa reducem spatiul actiunilor la un numar fixat. In cazul nostru spatiul va avea dimensiunea 12, continand tupluri predefinite de noi formate din: directie (stanga/inainte/dreapta), accelerare, franare.
Ca input pentru CNN am incercat doua abordari, una in care trimitem imaginea prelucrata, si una in care facem un stack de imagini pe care le trimitem. Dupa implementarea celor doua metode am observat ca varianta cu o singura imagine obtine rezultate mai bune.
Deci acest model primeste o imagine, care reprezinta state-ul, si va asocia o valoare fiecarei actiuni. Actiunea cu valoarea cea mai mare va fi considerata optima.
Pentru a antrena acest model, avem nevoie de un istoric creat dinamic al actiunilor alese si urmarile acestora (noul state, reward, solved) din care extragem un batch.
Bazandu-ne pe o idee similara TD-ului, parcurgem istoricul din state-ul cel mai vechi si evaluam deciziile luate de model prin prisma state-urilor viitoare (deja cunoscute).

De asemenea folosim si o idee similara importance sampling-ului, adica folosim 2 modele, unul pentru a prezice state-ul curent si celalalt pentru a-l prezice pe cel viitor.
Iteram prin batch; Extragand tuplul din batch (state, action_taken, reward, next_state, solved), folosim modelul temporar (acesta este folosit si la parcurgerea episodului) sa prezicem actiunile pentru state-ul curent (predictions), si modelul nostru principal il utilizam pentru a prezice next_state (future_predictions).
Vom modifica prezicerea pentru state-ul nostru astfel:

- daca este final: predictions[action_index] = reward
- daca nu este final: predictions[action_index] = reward + gamma * argmax(future_predictions)

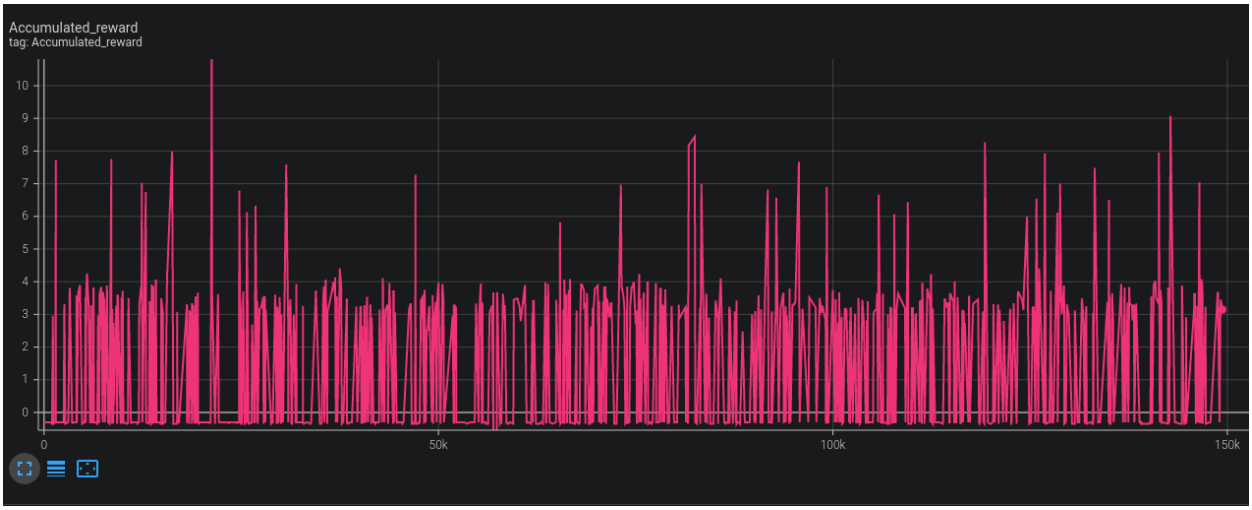
State-urile mentionate anterior vor reprezenta datele de antrenare, iar prezicerile vor constitui label-urile asociate acestora. Astfel vom antrena modelul cu aceste date peste tot batch-ul. Modelul nostru principal isi va actualiza weight-urile, preluandu-le de la modelul temporar la un anumit numar de episoade, acesta reprezentand un parametru antrenabil.

- **Rezultatele algoritmului (metrici, grafice)**

```
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(7, 7), activation='relu', strides=3, input_shape=(shape[0], shape[1], 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=128, kernel_size=(5, 5), activation="relu"))
model.add(Dropout(0.2, seed=42))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dense(len(self.action_space), activation=None))
model.compile(loss='mean_squared_error', optimizer=Adam(learning_rate=self.LR))

model.summary()  # structura modelului
```

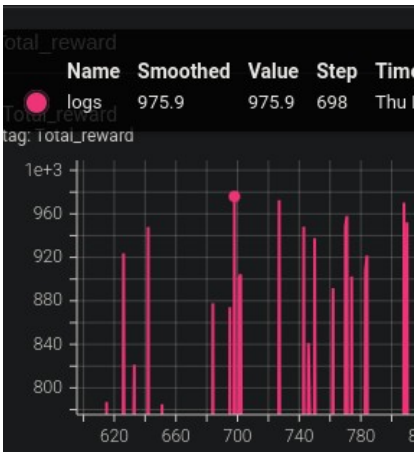
Am rulat acest model timp de 850+ episoade, insa converge in aproximativ 700 episoade. Cel mai mare total reward a fost obtinut in episodul 698 avand valoarea de 975.9. Practic modelul nostru a ratat un numar nesemnificativ de tile-uri completand cu succes intregul circuit. Mai jos atasam o serie de grafice ce ilustreaza performanta modelului nostru de-a-lungul antrenarii. Aceste grafice au fost salvate cu ajutorul tensorboard:

Istoric al reward-urilor acumulat pe fiecare frame



Reward-ul total de-a-lungul a 800 de episoade

Reward-ul total obtinut pentru cel mai bun episod



Evolutia functiei de loss de-a-lungul ultimelor episoade