Treniranje neuronske mreže za igranje igrice

Uroš Milenković

Uvod

- Izazov je naučiti neuronsku mrežu da pokreće agenta na osnovu senzora iz okruženja.

- Okruženje - svet koji ima svoje zakonitosti
- Senzori - daju nam informacije o okruženju
- Agent - zadaje akcije kojima menja stanje okruženja

Primena na tRex

- Okruženje - igrica
- Senzori - vrednosti koje dobijamo iz igrice (screenshot, game over, itd.)
- Agent - softer koji reaguje na vrednosti senzora

Mašinsko učenje

Algoritme mašinskog učenja možemo podeliti u 2 grupe:

1. **Nadgledano učenje:** Dostupan je skup *(x, y)* pri čemu su *x* ulazni podaci a *y* ciljna vrednost. Cilj učenja je pronaći vezu h među njima, *y = h(x)*.
2. **Nenadgledano učenje:** Dostupni su samo podaci *x* a potrebno je pronaći pravilnosti u podacima, npr. grupisati slične podatke.

Ojačano učenje (eng. Reinforcement learning)

- Ojačano učenje je oblast mašinskog učenja inspirisana *biheviorizmom*
- Biheviorizam je grana psihologije, potiče od engleske reči *behavior*
- Daje odgovor na pitanje “koje akcije treba da preduzme softverski agent u okruženju kako bi povećao ukupnu nagradu?”
- Spada i u nadgledano i u nenadgledano učenje
- Dok u nadgledanom učenju imamo ciljnu vrednost a u nenadgledanom nemamo, u ojačanom učenju imamo šture i vremenski odložene vrednosti - nagrade
- Samo na osnovu ovih nagrada agent mora da nauči kako da se ponaša u okruženju

Model ojačanog učenja

Interpreter prosleđuje stanje s_1 okruženja agentu

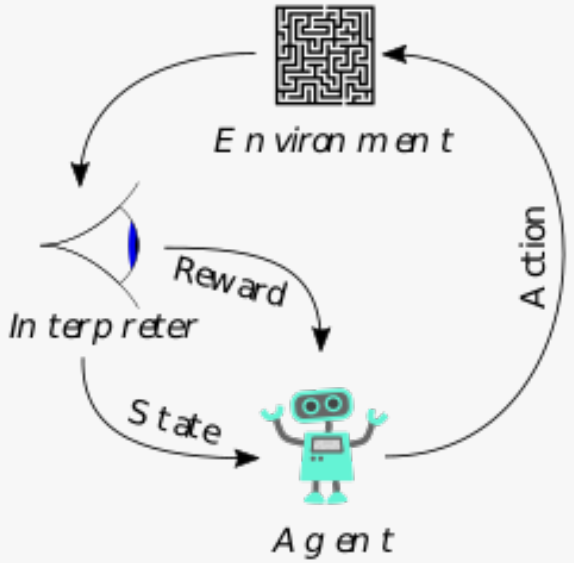
- Screenshot, vrednosti o brzini, pozicija T-Rex-a…

Agent na osnovu stanja preduzima akciju a koju izvršava u okruženju

- Skok, čučanj ili trčanje

Interpreter posmatra novo stanje s_2 i računa nagradu r koje prosleđuje agentu

- Nagradu dobija kad preskoči prepreku ili udari u nju
- Primetimo da nagrada može biti i negativna



Markovljev problem odlučivanja (eng. Markov Decision Problem - MDP)

- Skup stanja i akcija, zajedno sa pravilima prelaska iz jednog u drugo stanje, predstavljaju Markovljev problem odlučivanja.
- Jednu epizodu ovog procesa predstavlja konačna sekvenca stanja, akcija i nagrada:
  - s_0 , a_0 , r_1 , s_1 , a_1 , r_2 , s_2 , …, s_{n-1}, a_{n-1}, r_n, s_n
  - s_n zovemo terminalno stanje
- U MDP se pretpostavlja da verovatnoća s_{i+1} zavisi samo od trenutnog stanja s_i i akcije a_i, a ne od prethodnih stanja

Vremenski odložene nagrade

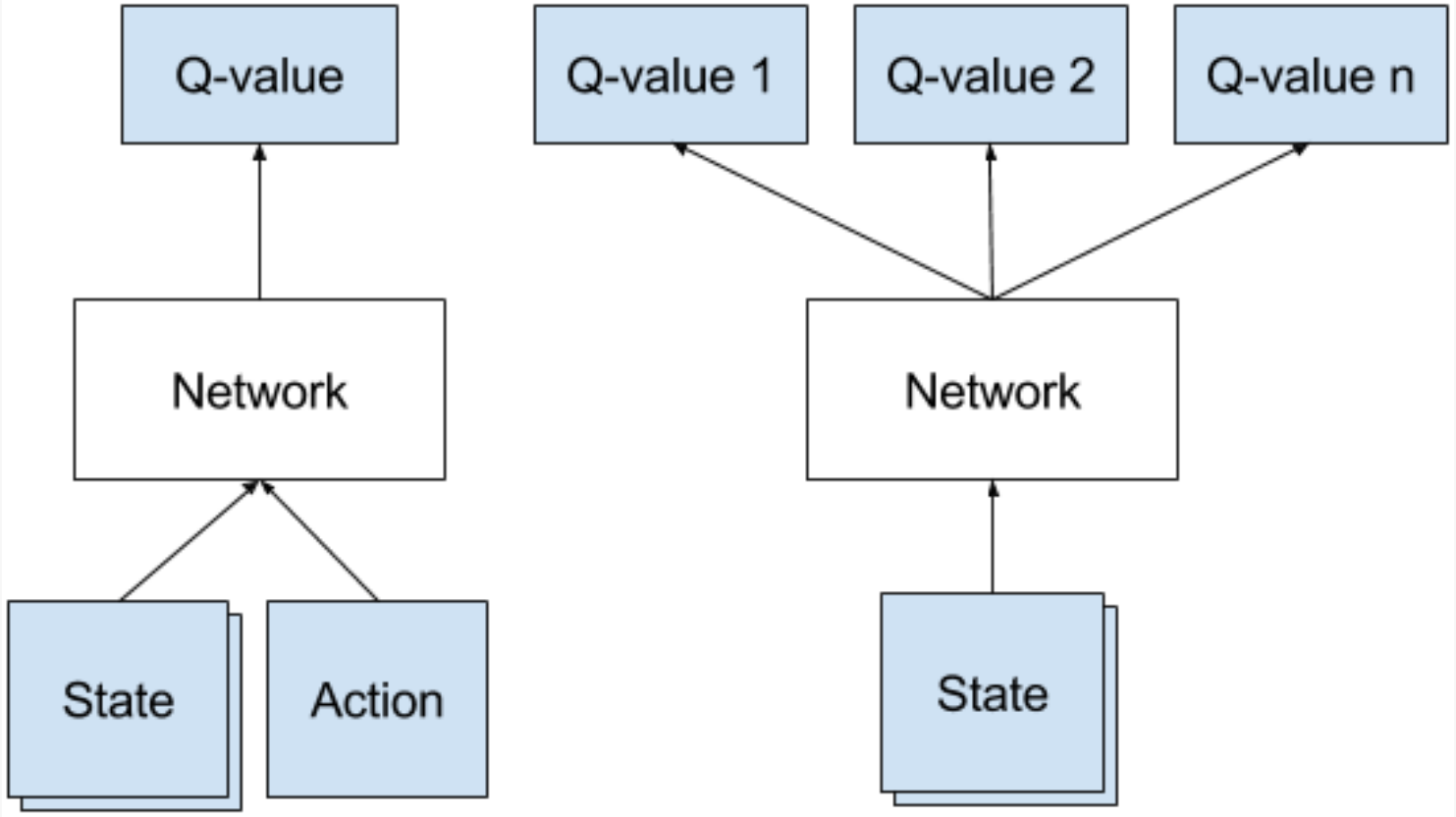
- U MDP možemo lako izračunati ukupnu nagradu:
  - $R = r_1 + r_2 + \ldots + r_n$

- Vremenski odloženu nagradu od tačke t možemo da računamo:
  - $R_t = r_t + r_{t+1} + \ldots + r_n$
- Pošto je okruženje stohastičko (ima random) ne možemo da budemo sigurni da li ćemo dobiti istu nagradu kad sledeći put izvršimo istu akciju. Iz tog razloga uvodimo “popust” na narednu nagradu
  - $R_t = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots + \gamma^{n-1}\, r_n$
    $= r_t + \gamma\, R_{t+1}$

Q učenje (eng. Q-learning)

- Definišemo funkciju $Q(s, a)$ koja predstavlja maksimalnu nagradu sa ”rezervom” kada izvršimo akciju a u stanju s
  - $\mathrm{Q}(s_t, a_t) = \max R_{t+1}$
    $= r + \gamma \, max_a \, \mathrm{Q}(s_{t+1}, a_{t+1})$
- Ovom funkcijom uvek biramo sledeću akciju tako da maksimizujemo nagradu u budućnosti

Q mreža (eng. Q-network)



Ponavljanje prethodnog iskustva

- Tokom igre čuvamo torku $<s, a, r, s'>$
- Kada treniramo mrežu nasumično biramo torke iz skupa prethodno sačuvanih torki
- Te torke čine mini seriju koju propuštamo kroz mrežu

Istraživanje okruženja

- Biramo nasumično akciju sa verovatnoćom ε
- Time istražujemo okruženje i sprečavamo pojavu lokalnog minimuma
- Početno ε postavimo na 1, a vremenom ga smanjujemo do 0,1

Deep Q-learning algoritam

```
observe initial state s
repeat
   select an action a
       with probability ϵ select a random action
       otherwise select a = argmaxa’Q(s,a’)
   carry out action a
   observe reward r and new state s’
   store experience <s, a, r, s’> in replay memory D

   sample random transitions <ss, aa, rr, ss’> from replay
memory D
   calculate target for each minibatch transition
       if ss’ is terminal state then tt = rr
       otherwise tt = rr + γmaxa’Q(ss’, aa’)
   train the Q network using (tt - Q(ss, aa))^2 as loss

   s = s'
until terminated
```

Implementacija

Alati

- Selenium - webdriver
- Chrome DevTools Console
- window.Runner.instance_
- Keras

Hiper parametri

- Kontrolisanje ε parametra za istraživanje okruženja
- Veličina mini serije
- Veličina memorije za ponovljeno iskustvo
- Nagrade:
  - $+0.1$ za svaku preskočenu prepreku
  - -1.0 za izgubljeni život

Dve arhitekture

Gusta mreža

- Ulaz: 15 float vrednosti
  - Brzina, pozicija tRex-a, pozicije i veličine naredne 3 prepreke
- Mreža:

Layer	Num units	Activation
Dense	64	ReLU
Dense	64	ReLU
Dense	3	Linear

Konvolutivna mreža

- Ulaz: 4 screenshot-a
- Mreža:

Layer	Num units	Filter size	Stride	Activation
Conv2D	16	8x8	4	ReLU
Conv2D	32	4x4	2	ReLu
Dense	256			ReLU
Dense	3			Linear

Rezultati

Tehnički problemi i rešenja

- Selenium ne podržava držanje dugmeta
  - interakcija sa okruženjem prebačena na OS
- Predugačko je čekanje od 100ms da bi se dugme držalo
  - loša implementacija niti u Python-u
  - Poslati samo keyDown event a kada treba nova akcija da se izvrši poslati odgovarajući keyUp
- Skaliranje ulaznih vrednosti na interval $(-1, 1)$
- Igra posle nekog vremena “zaglupi” - vraća da je partija gotova iako je započeta nova
  - Bash skript koji započinje učenje iznova na svakih 40 epizoda

Dalja unapređenja

- Genetski algoritam
- Kontrolisanje eps na osnovu vremena
- Podela memorije za iskustvo na 2 dela u zavisnosti od toga da li postoji nagrada
- Obeležavanje stanja radi stratifikacije
- Isprobavanje nove arhitekture na osnovu ranije generisanih stanja

Hvala na pažnji!

Pitanja?

Reference

- Playing Atari with Deep Reinforcement Learning, DeepMind - https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf