Playing Ultimate Tic-Tac-Toe using DeepQ and Monte Carlo Tree search

Petar Đorđević 353/2020 Milica Živković 132/2018

Jun 2022

Sadržaj

1 Uvod

Cilj projekta je napraviti neuronsku mrežu koja je sposobna da igra Ultimativni iks-oks [1] sa proširenim skupom pravila (igranje mini table koja je pobeđena nije dozvoljeno iako ima slobodnih polja). Ovo pravilo se uvodi da bi se izbeglo formiranje modela koji bi stalno igrao rizičnu taktiku kojom bi pobedio bilo koga ko igra prvi put ali bi gubio od svakoga ko iole zna da igra. Rad se radi po uzoru na AlphaZero [2].

2 Opis problema

Svaka 3×3 iks oks tabla naziva se mini tablom, dok se 3×3 raspored tih mini tabli zove globalna tabla. Igru počinje X i dozvoljeno mu je da igra na bilo koje od 81 praznih polja. Svaki potez određuje protivniku koju mini tablu mora da igra (npr. ako igrate [1,2] polje unutar mini table, time šaljete protivnika na [1,2] mini tablu). Ako pošaljete protivnika na mini tablu koja nema više slobodnih polja ili na mini tablu koja je pobeđena, protivnik sme da igra slobodno polje bilo koje nezavršene mini table. Mini tabla je pobeđena kao u običnom iks oksu, a igra se pobeđuje tako što pobedite 3 mini table u istoj koloni, vrsti ili dijagonali. Ukoliko niko od igrača nije ostvario uslov za pobedu, a ne postoji više validnih poteza za igranje, igra je nerešena.

Gruba ocena broja različitih stanja u igri je 81^3 (svako polje može biti X, O ili prazno) pomnoženo sa 9 (prošli igrač nas je poslao na jednu od 9 mini tabli), što je približno 3.99e+39, a eksperimentalno se pokazuje da je broj akcija iz jednog stanja u proseku između 6 i 7, što čini igranje igre nekim algoritmom pretrage nemoguće. Takođe, ova igra spada u grupu nerešenih igara (nije pronađen algoritam koji deterministički dovodi do toga da neki igrač uvek pobedi).

Pošto je u pitanju naizmenična igra strategije bez skrivenih stanja (npr. u pokeru ne znamo koje karte drži protivnik), igra je dovoljno slična šahu, šogiju i gou da se nad njom može primeniti AlphaZero.

3 Naša implementacija

3.1 Model

Pošto je neuronska mreža koju AlphaZero koristi zahtevna za treniranje a naša igra je jednostavnija od do sada obrađenih, odlučili smo da koristimo jednostavniji model koji će se sastojati iz nekoliko konvolutivnih slojeva praćenih gusto povezanom mrežom u nadi da će i dalje biti sposoban da nauči igru.

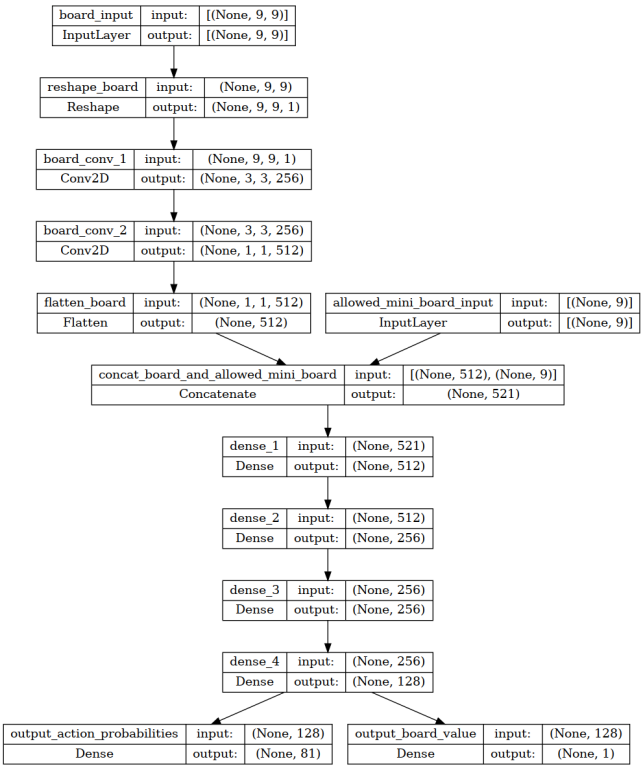
Stanje igre jednoznačno je određeno sa 2 parametra: tabla za igru T_{ij} i mini tabla t_i na koju nas je protivnik poslao. T_{ij} se predstavlja sa 9×9 matricom čija polja mogu imati vrednosti 0 (prazno), 1 (X) i -1 (O), dok je parametar t_i ceo broj od 0 do 8 koji se kategoričkim enkodiranjem svede na niz od 9 elemenata u kojem je tacno jedno polje 1 a ostala su 0. Ova dva parametra je moguće kombinovati u jedan ali je ideja bila da model sam nauči koji su mu dozvoljeni

potezi na osnovu podataka koje bi i igrač imao, bez enkodiranja dozvoljenih poteza u tablu.

Neophodno je imati dva izlaza iz mreže: niz od 81 verovatnoća za svaku moguću akciju p_i gde veća verovatnoća predstavlja bolji potez, i vrednost trenutnog stanja v kao realnu vrednost između -1 i 1. Dok je P očigledno neophodna, v se ne koristi direktno u odlučivanju igranja poteza već samo kao pomoć za navigiranje Monte Karlo algoritma u izboru stanja za dublju pretragu.

U konvolucioni sloj bi ulazilo samo T. Koristi se informacija da je T podeljeno na 3×3 mini table radi pametnijeg određivanja koraka i filtera. Da bi se 9×9 tabla svela na 3×3 potrebno je koristiti korak (3, 3) i veličinu filtera 3×3. Taj rezultat se opet propusti kroz konvoluciju filtera 3×3 da bi se na kraju dobio niz različitih osobina table. Dovoljno je kao aktivacionu funkciju koristiti $ReLu$.

Niz osobina nadoveže se na t i novonastali niz se propusti kroz 4 gusto povezana sloja sa $tanh$ aktivacijom u nadi da će model naučiti da rezonuje o tim informacijama, i iz njih izlaze oba izlazna sloja, P sa softmax aktivacionom funkcijom da bi suma svih verovatnoća bila 1, a v sa $tahn$ aktivacijom da bi se svelo na željeni interval. Funkcija gubitka za P bi bila kategorička krosentropija, a za v srednje kvadratna greška. Optimizacioni algoritam bi bio $Adam$.



Slika 1: Grafički prikaz modela.

3.2 Monte Karlo

Zbog kompleksnosti igre običan MinMaks algoritam za dubinsku pretragu nije primenljiv, pa se odabir prostora pretrage mora pametnije vršiti. Za to koristimo Monte Karlo algoritam pretrage stabla igre. Pri određivanju sledećeg poteza u igri Monte Karlo algoritam izvrši veliki broj simulacija (silazaka niz stablo igre). Broj simulacija unapred se zadaje, u AlphaZero algoritmu je korišćeno 800 ali smo mi zbog performansi smanjili broj simulacija na 600. Na kraju simulacije rezultati se uproseče i vraća se optimizovani niz p_i.

Svaka simulacija se započinje iz trenutnog stanja koje je prosledjeno algoritmu. Koristeći izlaz iz modela računa se supremum poverenja (upper confidence bound) svake akcije.

$$U_{sa} = Q_{sa} + c \cdot P_{sa} \cdot \frac{\sqrt{\sum_b N_{sb}}}{1 + N_{sa}}$$

gde je Q_{sa} do sada izračunata vrednost stanja s nakon igranja akcije a, P_{sa} je verovatnoća iz modela da se iz stanja s odigra akcija a, N_s je koliko puta smo ušli u stanje s i N_{sa} je koliko puta smo iz stanja s odabrali da odigramo akciju a. Konstantom c se određuje koliko je algoritam sklon istraživanju novih stanja (tj koliko se preferira pretraga u širinu u odnosu na pretragu u dubinu) koju smo mi postavili na 2.

Jedan silazak se završava kada naidjemo na stanje koje do sada nismo obišli. Ukoliko je stanje završno, vraća se nagrada iz okruženja, a ukoliko nije vraća se v predviđeno modelom. Zbog toga Q_{sa} možemo gledati kao prosek svih nagrada dobijenih iz tog stanja tom akcijom.

Pozivi algoritma su potpuno nezavisni, vrednosti se ne čuvaju već računaju ispočetka.

3.3 Agent

Agent predstavlja klasu koja kombinovanjem neuronske mreže i Monte Karlo algoritma pretrage igra igru i uči o njoj. Akciju koja će se odigrati u okruženju bira tako što Monte Karlo algoritmom dobije p_i i onda izabere neku akciju koja je ϵ udaljena od najverovatnije akcije. Ovo se uvodi da bi se izbeglo preprilagođavanje kada više akcija imaju istu verovatnoću.

Treniranje se vrši tako što jedan agent igra sam protiv sebe veliki broj puta, pamti sve rezultate Monte Karlo pretrage i nakon određenog broja partija pozove fit nad modelom koristeći neki nasumični uzorak sačuvanih poteza. Na ovaj način model ne uči igru direktno, već vremenom uči da bolje aproksimira rezultat Monte Karlo pretrage što onda pri samoj pretrazi pruži verodostojnije podatke. Pošto je igra simetrična agent uvek igra kao X (kada igra kao O samo se T pomnoži sa -1 pre predviđanja poteza) da bi učio manje parametara igre. Agent može igrati i bez pamćenja prethodnih poteza kada ne trenira.

4 Rezultati

Nakon svakih 100 partija igre model smo puštali da se 10 partija bori protiv trenutnog najboljeg modela i beležili rezultate. Red *score* se računa kao broj pobeda minus broj gubitaka. Svaki model igra jednak broj puta kao X i kao O.

Timestamp	score	is current best
2022-05-18 00:16:24	0	1
2022-05-21 17:49:21	-3	0
2022-05-23 10:15:08	-4	0
2022-05-25 04:27:39	2	1
2022-05-28 04:20:12	-5	0
2022-05-29 22:49:26	-5	0
2022-05-31 18:06:20	0	0

Tabela 1: Log treniranja.

Radi boljeg testiranja napravili smo agenta koji koristi samo Monte Karlo algoritam bez neuronske mreže. Naš najnoviji model je protiv njega imao 6 pobeda, 3 nerešena i 1 gubitak, što definitivno ukazuje na to da je model uspeo da nauči nešto. Napravili smo i grafičko okruženje pomoću kojeg možemo mi igrati protiv modela, ali model još uvek nije dovoljno istreniran da pobedi osobu koja zna da igra.

Važno je naglasiti da je za 700 partija samo igre bilo potrebno 2 nedelje neprekidnog treniranja. Problem koji imamo (koji je konzistentan sa Deep Mind-ovim nalazima) je da je sam algoritam najviše usporen jer se podaci stalno prebacuju sa procesora na grafičku karticu i obratno. Koristeći numba biblioteku za JIT kompilaciju uspeli smo da optimizujemo program tako da je 97% njegovog izvršavanja procesuiranje neuronske mreže i prenos podataka.

5 Zaključak

Iako je rezultat veoma obećavajuć, i očigledno se postižu poboljšanja, algoritam nije praktično upotrebljiv za nekoga ko nema odgovarajući hardver i resurse. Bilo je potrebno da napravimo server u dnevnoj sobi sa NVidia grafičkom karticom i da ga pustimo da radi 2 nedelje bez prestanka da bi dobili bilo kakve rezultate, a čak i to je daleko više nego što prosečan student ima. Potencijalno bismo eksperimenta radi nekada pustili algoritam da trenira duže čisto da bi videli kakvi rezultati mogu da se postignu.

Literatura

[1] Wikipedia: Ultimate Tic-Tac-Toe

[2] Deep Mind: A general reinforcement learning algorithm that masters chess, shogi and Go through self-play