Dots and Boxes

Matematički fakultet, januar 2019.

Strahinja Ivanović, 149/2015 :: Darko Veizović, 310/2015

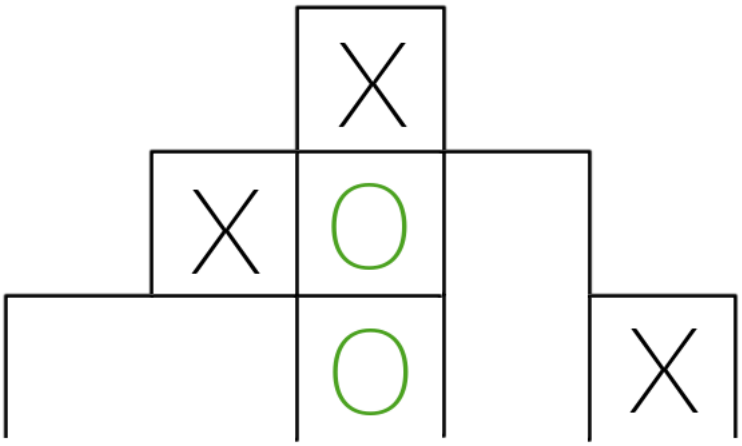
Sadržaj

1 Uvod

1.1 Abstract

Igre na tabli sa dva igrača, kao sto su **šah**, **go** ili **šogi**, najduže su izučavan domen u istoriji veštačke inteligencije. Najjači programi stvoreni da igraju ove igre bazirani su na kombinaciji sofisticiranih tehnika pretrage, adaptacija na specifičnost igre i ručno pravljenim funkcijama evaluacije, redizajniranim više od nekoliko decenija od strane ljudskih eksperata u navedenim igrama.

Tabla ove igre podseća na znak *KARO* u kartama. Sastoji se od 57 kvadrata koji mogu biti označeni kao **x** ili **o**. Svaki kvadrat formiraju 4 linije, a cela tabla ih ima ukupno 96. U svakom potezu, igrač markira jednu liniju na tabli. Ako je pritom formirao kvadrat (ili dva) ostaje na potezu i markira sledeću liniju, tako rekurzivno. Pobednik je igrač koji je zauzeo vise kvadrata. Ovako formulisana, igra **Dots** podseca na igru **go**, ali odnosi kvadrata i linija, kao i sam oblik table, prave dovoljnu razliku.



Slika 1: prikaz jednog dela table

Za potrebe upoređivanja korišćeni su sledeći objekti:

- program koji nasumično odigrava poteze
- klasični *alpha/beta* sa intuitivnom funkcijom evaluacije
- ljudski ekspert

Program opisan ovde dosegao je nadljudske performanse, pobedivši ubedljivo sva tri objekta. Nasuprot opisanim metodama iz uvodnog dela, program ne zna ništa o igri, sem njenih pravila.

2 Metode

2.1 Anatomija klasičnog Dots programa

Stanje pozicije određeno je binarnim brojem od 96 cifara, gde **1** predstavlja upisanu liniju, brojem polja obeležnih kao **x** ili **o** i znakom igrača na potezu (1 ili -1).
Evaluacija pozicije dobija se pravljenjem stabla, kod koga se iz svake pozicije potezima dolazi do nove pozicije. Dubina pretrage, odnosno visina stabla, zavisi od pozicije, jer se na primer, iz početne pozicije u narednih 5 poteza može dobiti više od 7.000.000.000 mogućih stanja na tabli, dok se pri kraju igre dubina pretrage može značajno uvećati zbog suženog izbora poteza, pa tako u poziciji koja je 10 poteza pred kraj, možemo razviti stablo do listova i tako napraviti nešto više od 3.500.000 čvorova.

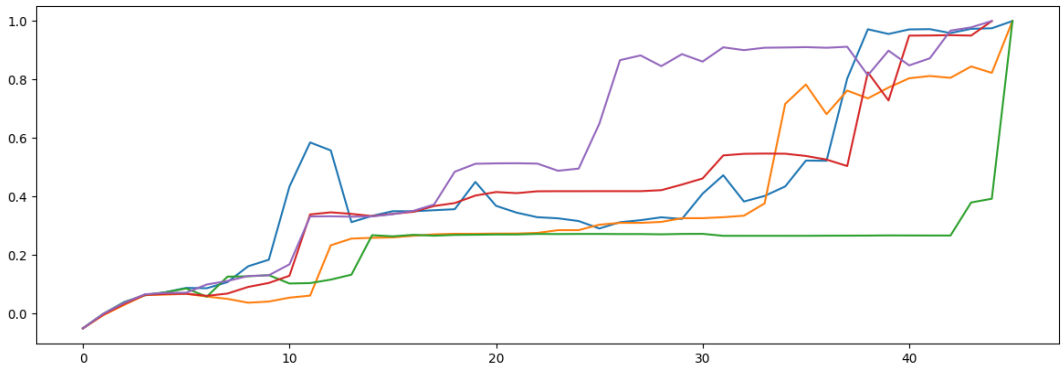
Dodatno, *alpha/beta* srezivanjem izbegavamo grananje stabla u delove gde je jedna strana verovatno potpuno dominirana drugom. Kao **funkcija evaluacije** svakog od listova (ili pseudo listova) koristi se obična razlika između obeleženih polja. Konačna evaluacija dobija se *minmax* pretragom, gde u svakom nivou uzimamo *maximum* ili *miminum* u zavisnosi od strane na potezu u tom nivou *(pozitivne vrednosti su dobre za **x**, negativne za **o**)*.
Niti jedna od opisanih tehnika se ne koristi u ovom programu, iako bi neke od njih verovatno dovele do boljih performansi. Bez obzira na to, program je fokusiran na čistom učenju iz igranja protiv samog sebe.

2.2 Dots

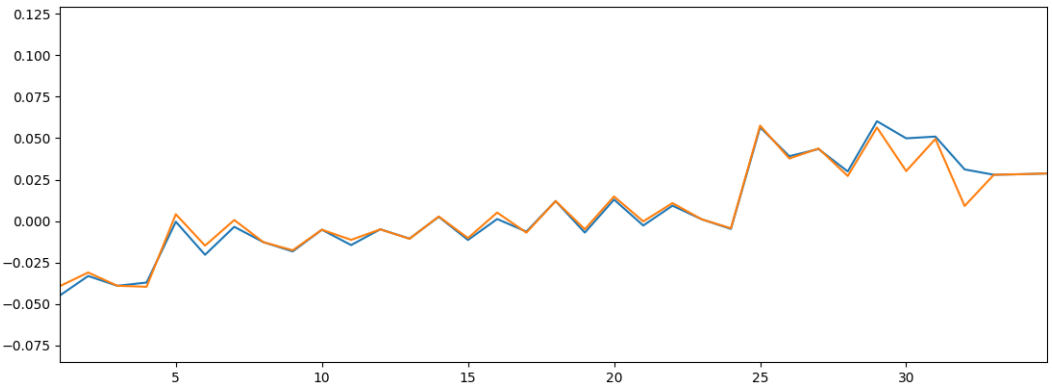
Umesto *alpha/beta* pretrage, u ovom programu koristi se **Monte Carlo Tree Search**. Svaki od čvorova predstavljen je svojom pozijom na tabli, brojem obilaska i sumom estimacija, kao i inicijalnom estimacijom neuronske mreže koja se pamti zbog kasnijeg podešavanja same mreže.
MTCS implementiran u programu je nalik klasičnom, s tim da se umesto rollout-a, gde se kao evaluacije pozicija uzimaju srednje vrednosti dobijene kao rezultat simulacija iz date pozicije, evaluacija dobija pomoću neuronske mreže koja vraća šanse za pobedu iz te pozicije, **od -1 do 1**, gde **1** znaci da **x** sigurno dobija, dok **-1** donosi sigurnu pobedu za **o** stranu. Prilikom formiranja stabla, *USB1 algoritmom* pravi se balans između istraživanja novih grana i dubljeg pretraživanja obećavajućih. Na kraju pretrage program pohlepno bira najjači potez u odnosu na broj obilaska potencijalnih poteza i njihovih šansi za donošenje pobede.
Po završetku igre, unutrašnja stanja neuronske mreže se azuriraju u skladu sa ishodom igre, kao i odnosa inicijalne estimacije pozicije između bliskih čvorova. Mreža se modifikuje u cilju smanjivanja razlika između estimacije čvora i srednjih vredosti estimacija dva čvora posle njega, kao i razliku inicijalne estimacije verovatnoće i verovatnoće dobijene MTCS pretragom. Ako estimacije pri svakom odigranom potezu nasumično inicijalizovane mreže predstavimo u grafu, možemo očekivati da dobijemo isprekidanu *cik-cak* liniju koja ukazuje na to da program nema predstavu šta se desava, u jednom potezu je siguran da pobeđuje, već u sledecem da gubi. Ažuriranje neuronske mreže na ovaj način izglađuje taj grafik nakon svake iteracije, što reflektuje **shvatanje** pozicije. Za resavanje ove

igre determinističkim metodama bio bi potreban obilazak *96!* pozicije, sto je, $9.916779e + 149$.
Ovakvom metodom za svaku poziciju imali bismo konačnu evaluaciju. **MTCS** pristupom, uz mnogo manje obilazaka, verovatnoća ishoda konvergira ka toj evaluaciji.

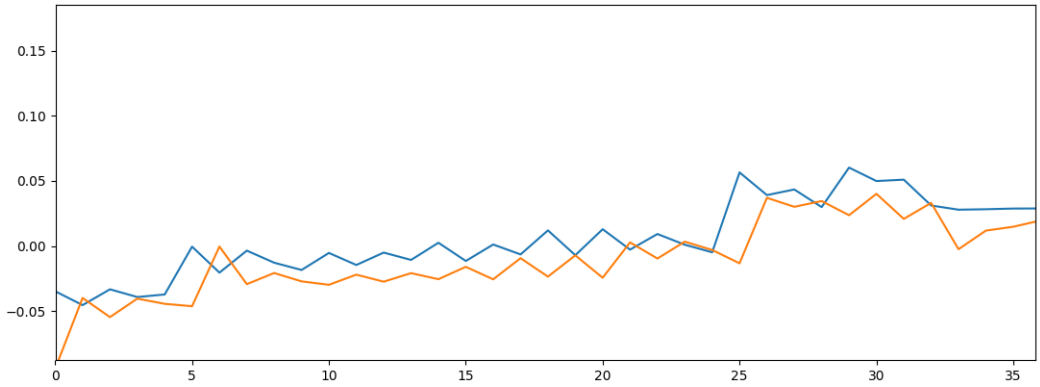
2.3 Grafovi



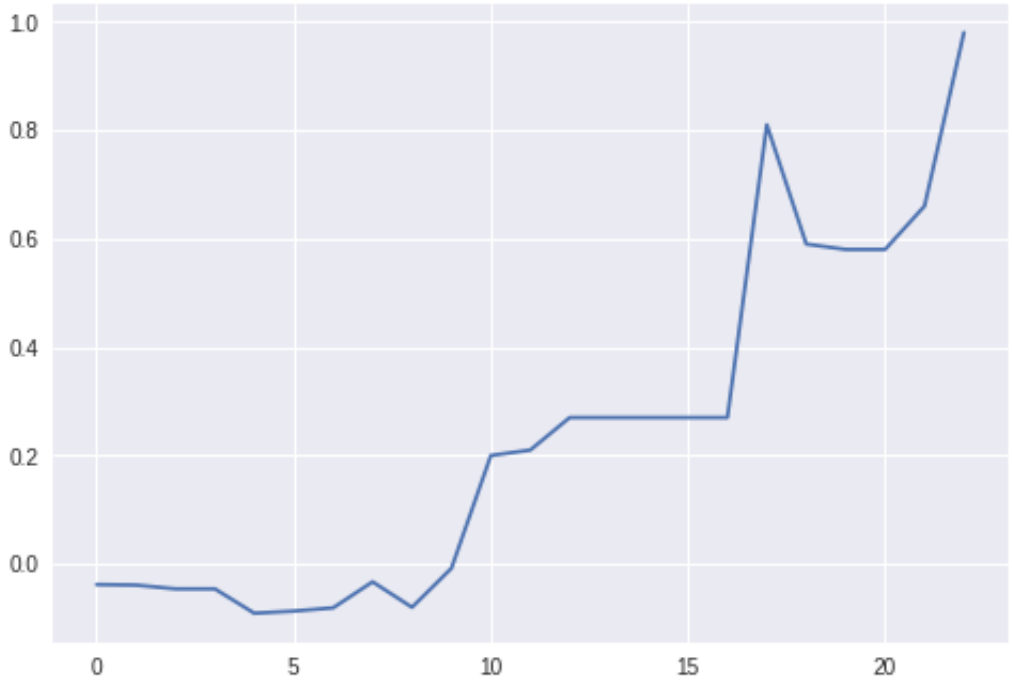
Slika 2: tok partije protiv nasumičnih poteza



Slika 3: razlika u verovatnoćama MTS i estimacijom neuronske mreže



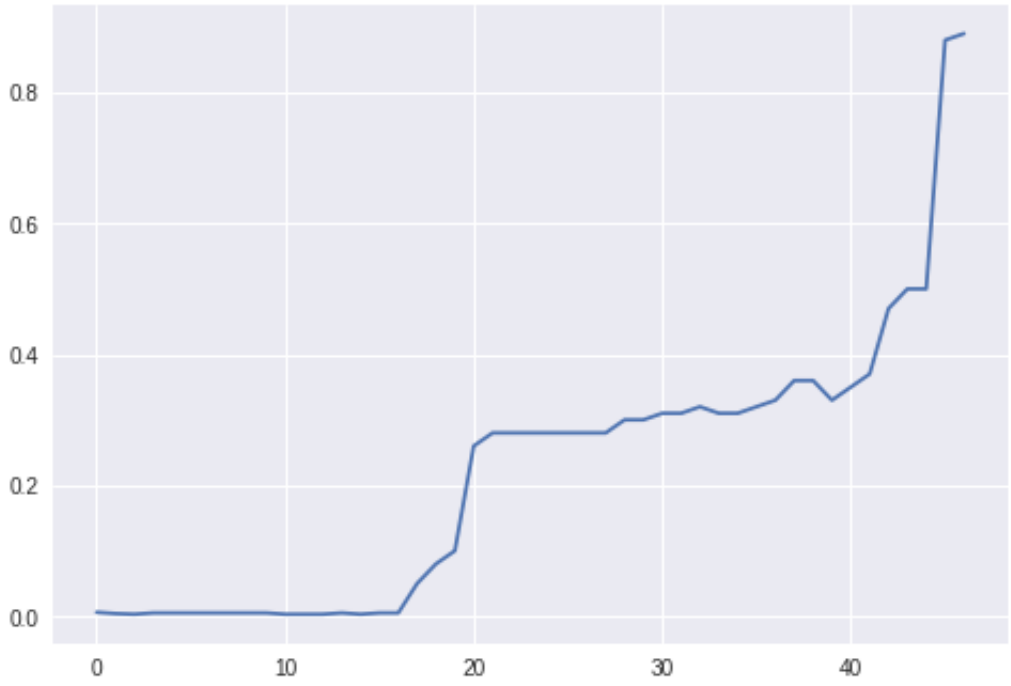
Slika 4: ažuriranje neuronske mreže



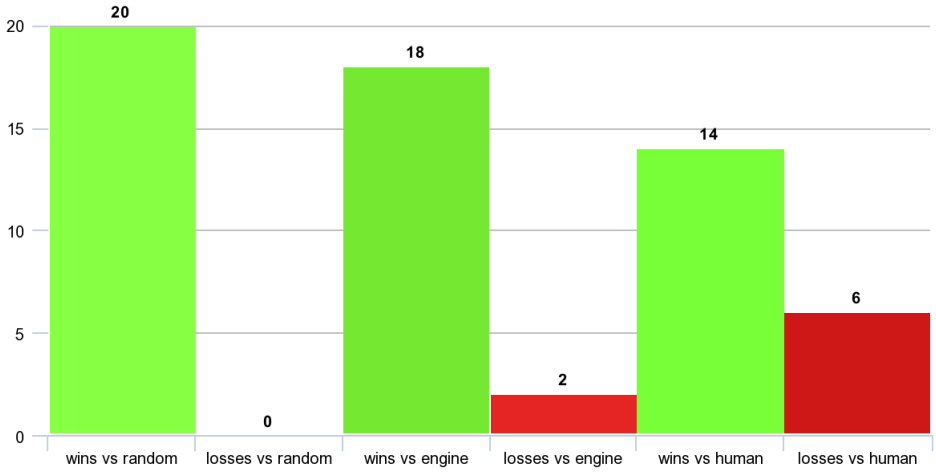
Slika 5: partija iz rane faze treniranja - los

Specifikacija hardvera na kojoj je mreža trenirana:

- GPU: 1xTesla K80, 2496 CUDA cores, 12GB GDDR5 VRAM
- CPU: 1xsingle core hyper threaded Xeon Processors @2.3Ghz, 45MB Cache



Slika 6: partija iz napredne generacije - win



Slika 7: odnos pobeda i poraza

3 Osvrt i predlozi poboljšanja

Kao prostor za poboljšanje efikasnosti programa, kao i brzine njegovog izvršavanja i treniranja neuronske mreže, možemo kao očigledan početak uzeti izbor jezika u kome je program pisan. Bez sumnje, možemo reći da je ***python*** spor, pa

bi se implementacijom istog programa u npr. programskom jeziku ***C*** postiglo ogromno poboljšanje performansi. Ipak, neuronska mreža je trenirana na Google Colab, koji za sada podržava samo ***python***. Sa druge strane, sa ***python*** bibliotekama za rad sa veštackom inteligencijom poput: Tensor Flow i Keras u zamenu za perormanse dobijate udobnost pri implementaciji i lakoću prenošenja ideja u kod.
Kao konkretna poboljšanja koda mogu se dodati pomenute heuristike (alpha/beta...). Jedna od stvari koja je mogla biti iskorišćena je binarna priroda ishoda igre. Sa 57 mogućih polja, jasno je da se igra moze završiti samo porazom ili pobedom, pa se predstavljanjem verovatnoća ishoda moglo bolje predstaviti u odnosu na pobedu jedne strane. Samim tim verovatnoće bi se kretale od 0 do 1, što bi omogućilo primenu drugačije funkcije aktivacije (sigmoid umesto tanh) koja bi verovatno doprinela bržem konvergiranju optimumu.

Takođe, još jedna stvar specifična za igru koja se mogla iskoristiti je simetričnost table. Znajući to, pre vršenja estimacije, tabla kao ulazni parametar u neuronsku mrežu bi bila proizvoljno rotirana, čime bi se smanjila mogućnost *overfitovanja* i generalno ubrzao postupak treniranja.

4 Literatura

Veliki deo ideja preuzet je iz Alpha Zero papira koji je objavila kompanija Deep Mind. Takođe, kao smernica je poslužio Lc0, a pomoć u shvatanju pojedinih koncepata je dobijena od strane ljudi sa Leela Chess Zero Chat.