**Kvantni križić-kružić i integracija RapidMinera s drugim programskim rješenjima**

Uvodni opis problema

Tijekom akademske godine godine 2010./2012. kao projekt na kolegiju "Umjetna inteligencija" radio sam na implementaciji igre kvantni križić kružić (<https://github.com/stanko-arbutina/ml-qttt/tree/master/ui> ). Igru možemo zamišljati kao više paralelnih igara klasične varijante. Postoji posebna vrsta poteza (*measurment*, odn. mjerenje) kojim igrač odlučuje koja od mogućih varijanti igre se stvarno desila. Više o igri: <http://en.wikipedia.org/wiki/Quantum_tic_tac_toe>

U implementaciji smo koristili min-max algoritam s alfa-beta podrezivanjem. Svaka igra završava u manje od 13 poteza, ali se stablo pretraživanja širi eksponencijalno, te je svako pretraživanje veće od dubine 3-4 trajalo neprihvatljivo dugo na prosječnom računalu. Zbog toga smo u vrednovanju krajnjih čvorova pretraživanja koristili fiksnu heuristiku koja nam se intuitivno učinila najpogodnija. Iako bi se daljnjom optimizacijom algoritma (npr. boljim prepoznavanjem ekvivalentnih stanja), te korištenjem paralelizacije (preko QUDAe) vremenski prihvatljiva dubina mogla značajno povećati, već tada se pojavila ideja da bi se sama heuristika mogla učiti nekom od metoda strojnog učenja.

Također, valja imati na umu da ako želimo igru učiniti široko pristopačnom, ona bi se trebala moći izvoditi u web browseru i na mobilnim platformama - stoga oslanjanje na procesnu moć i pristup sistemskim resursima niskog nivoa nije praktičan smjer razmišljanja.

Osnovna ideja za projekt je reimplementacija igre, ali ovaj put koristivši neku od metoda strojnog učenja. Implementaciju (sa strane klijenta) bih pisao u javascriptu jer je to front-end tehnologija koja je najraširenija.

Kako će igra biti dostupna na webu, očiti izvor podataka za učenje su odigrane partije. Naravno, jer ne mogu računati da će puno ljudi igrati igru, a kako bih mogao završiti projekt, trebati ću napraviti i generator eksperimenata (igre računala protiv sebe).

Cilj i hipoteze istraživanja problema

Ciljna funkcija nam mora dati odgovor na pitanje što slijedeće odigrati. Kao ulaz bi mogla dobivati niz do sada odigranih poteza, ili trenutno stanje ploče. Zbog toga što daljnji nastavak igre ovisi isključivo o trenutnom stanju (a ne o načinu kako smo došli do njega), jasno je da nam druga varijanta bolje opisuje problem. Kao izlaz bismo mogli dobiti fiksan potez (najbolji potez iz trenutnog stanja za igrača koji je na redu) ili broj koji reprezentira vrijednost određenog stanja (stanja koja vjerojatnije vode do pobjede imaju veću vrijednost) – slično primjeru u knjizi T. Mitchella s klasičnom varijantom igre. Smatram da je u ovom slučaju bolje koristiti vrijednost stanja iz nekoliko praktičnih razloga:

* iz vrijednosti jednostavno dobijemo najbolji potez (onaj koji vodi u stanje s najvećom vrijednošću)
* vrijednost nam daje više informacija – osim znanja o tome je li neki potez najbolji ili nije, dobivamo i informacije je li neki potez bolji od nekog drugog. Dakako, funkcije s kojima ćemo raditi su aproksimacije, ali vjerujem da bi nam funkcija koja vraća vrijednost omogućila da brže naučimo igrati poteze koji, ako nisu najbolji, nisu ni loši. Razlog je intuitivan, lakše(i brže) je zaključiti da npr. “među (aproksimativno) ekvivalentnim potezime, potezi u kojima gubim kontrolu nad pločom su u principu lošiji od onih u kojima ne” nego “koji je najbolji potez”.
* u eventualnom daljnjem razvoju igre, takvu funkciju možemo lako integrirati u npr. min-max pretraživanje (početna ideja), te kombiniranim pristupom dobiti jačeg igrača
* dobivamo veću fleksibilnost – bez da promijenimo model, možemo npr. smanjiti težinu igre tako da igramo drugi najbolji potez, ili možemo procijeniti snagu ljudskog igrača (vidimo koliko su njegovi potezi “jaki”)

Sama reprezentacija ciljne funkcije (i prostor hipoteza) ovisi o korištenoj metodi učenja (glavni konkretni rezultat projekta bi i bilo okruženje u kojem možemo relativno jednostavno eksperimentirati s raznim metodama), ali napomenuo bih da “stanje ploče” koje ciljna funkcija dobiva kao ulazni parametar, možemo opisati na razne načine. Osim samih sirovih podataka (koji znak se nalazi na kojim koordinatama), lako je zamislivo da postoje razna izvedena svojstva tih podataka koja su korisna. Npr. u šahu, opis stanja “na potezu smo, protivnik ima samo kralja, a mi kralja i kraljicu” pokriva velik broj broj rasporeda figura, ali zadržava sve bitne informacije o njihovoj vrijednosti.

Pregled dosadašnjih istraživanja

Tijekom prošle godine pojavila se implementacija igre na <http://www.qtpy.mobi/>. To je, koliko sam uspio pronaći, jedina javno dostupna verzija u kojoj se može igrati protiv računala. Također, bot se trenira jednim fiksnim algoritmom strojnog učenja. Sam algoritam učenja je pisan u Pythonu, i javno dostupan na githubu. Ukratko, server za određenu poziciju vraća slijedeći potez kružića (bot igra samo kao kružić). U bazi su pohranjeni parovi (stanje, potez), te su dodani atributi koji za svaki par sadrže informacije o tome u koliko je igara sudjelovao s obzirom na rezultat (pobijedio, izgubio, neriješeno). Kada se bira slijedeći potez, ako postoji neki koji nije igran iz trenutne pozicije, vrati se taj (slučajno), a ako su svi igrani, onda se vraća onaj koji je prvi po broju neriješenih/dobivenih igara (ovisno o težini).

Osnovni nedostatak ove implementacije je što dosta loše igra. Konzistentno gubi protiv “brute force” igrača (minimax dubine tri), a i početnik u igri ga počne pobjeđivati nakon nekoliko odigranih partija. Mislim da je osnovni problem u tome što je zbog reprezentacije stanja (pozicije znakova na ploči) prostor za pretraživanje jednostavno prevelik, te čak i ako je mnogo ljudi igralo igru, u kasnijim fazama igre dobivamo uglavnom slučajne poteze (koji još nisu odigrani). Iz iskustva znamo da fotografsko “štrebanje” pozicija i najboljih poteza nije dobar način za učenje bilo kakve kompleksnije igre.

Također, zbog razloga koje sam već naveo, “slijedeći potez” je (pogotovo u kontekstu velikog prostora stanja) nezahvalna ciljna funkcija. Osim toga, intuitivno bih rekao da je učenje kroz ponavljanje (rote learning) efektivna metoda u situacijama u kojima je faktor grananja (barem na početku) relativno malen (pa postoje čvorovi koje često ponavljamo).

O primjeni strojnog učenja za igranje ove igre nisam našao više ništa. Postoji malo općenitiji članak o strategijama igranja jedne druge varijante (možemo postavljati znakove na proizvoljno mnogo nezauzetih mjesta, s različitim vjerojatnostima da će tamo stvarno i ostati nakon mjerenja), te ne mogu procijeniti koliko su rezultati primjenjivi u ovom slučaju. Ipak, zanimljivo je da se proširivanjem klasične igre kvantnim potezima svojstva igre bitno mijenjaju (npr. moguće je da kružić pobijedi).

Čini se općenito da izbor metode kod ovakvih igara uveliko ovisi o svojstvima same igre. Npr., metoda koja se pokazala kao relativno uspješna za go je pokušavati uočiti uzorke dijelova pozicije iz partija velemajstora i emulirati razrješavanje istih. No, u gou možemo dobivati/gubiti bodove na jednom dijelu ploče sasvim neovisno o drugom, što bitno povećava primjenjivost takve metode.

Materijali, metodologija i plan istraživanja

S obzirom na nedostatak iskustva s konkretnim problemom, bilo bi praktično kada bismo imali fleksibilno okruženje za eksperimentiranje s nekoliko različitih metoda, unutar kojeg bi lakše mogli korigirati pristup. Osnovni cilj ovog projekta je demonstracija programatskog pristupanja RapidMineru, u konkretnoj primjeni (učenje o igranju kvantnog križić-kružića).

Korisnik će u svom web pregledniku moći pratiti stanje ploče i igrati poteze kada je na redu. Skripta će (kada je na redu) slati upite za sva stanja koje slijede legalne poteze i odabrati potez koji vodi stanjem s najvećom vrijednošću. Sa strane servera će biti skripta koja pristupa RapidMiner APIu (dobivamo evaluaciju od određenog learnera, za početak možda i ručno fiksiranog). Na kraju partije pošaljemo niz stanja od kojih se sastojala (što ulazi u skup podataka za trening).

Podaci koje ćemo slati za određenu poziciju će osim samih pozicija znakova na ploči (kodiranih) sadržavati i znak igrača koji je na potezu (evaluacija pozicija nije ista za oba igrača), jednoznačno određeni znakovni na ploči (postoje situacije u kojima će bez obzira na rezultat mjerenja, ne nekoj poziciji biti uvijek isti znak), opis stanja za svaku od 8 linija na kojima je moguće ostvariti pobjedu (npr. niti jedan igrač ne može postaviti sva tri znaka, križiću fali jedan itd), redni broj poteza (u varijanti igre opisanoj u članku koji sam naveo, sugerirano je da je strategija u završnici različita od one u otvaranju), te eventualno (ako se sjetim) još neke metapodatke koji bi se mogli pokazati kao “definirajući” za stanje.

Što se tiče skupa stanja za trening, njihova vrijednost je jasno određena za završna stanja igre. Osim toga, za stanja koja jednoznačno određuju pozicije znakova na ploči nakon kojih je igra gotova (npr. još nismo napravili mjerenje, ali kako god da se napravi, imati ćemo tri križića na dijagonali – već u ovom trenutku znamo da je križić pobijedio) i za stanja koja prethode završnima (npr. odigrali smo potez nakon kojeg nas u slijedećem protivnik sigurno pobjeđuje) su vrijednosti jasno određene. Za ostala stanja nisam sasvim siguran – možemo ih jednostavno ne uključivati u training set (mislim da bi tako mogli imati overfitting za strategiju u završnici), kao vrijednost im dati postotak izgubljenih/dobivenih partija, minimum/maksimum stanja koja su slijedila iza njih – tu ću trebati još istraživati.

Server će osim evaluacije pozicije slati i podatak koliko je puta stanje odigrano (koliko se često pojavljuje u trening setu) – biti će korisno (barem kod generiranja eksperimenata) istraživati nove dijelove prostora stanja.

Moguće je da će tijekom vremena neki dijelovi prostora biti potpuno istraženi – u tom slučaju je vrijednost svih čvorova potpuno jednoznačna (kao u min-max algoritmu) – što bismo lako mogli pratiti pomoću nekoliko dodatnih atributa i imati preciznije podatke, ali ostavio bih to za eventualno proširenje.

U RapidMineru možemo usporediti nekoliko metoda učenja i izbora prostora hipoteza, te lako ocijeniti njihovu učinkovitost prema rezultatima u igri.

Ukupni plan bio bi slijedeći:

1. Napraviti web sučelje u kojem možemo igrati igru (dva ljudska igrača)
2. Napraviti 'dummy' računalnog igrača, koji slučajno odabire jedan od legalnih poteza
3. Napraviti generator eksperimenata - dva računalna igrača igraju jedan protiv drugog više puta
4. Slanje podataka o igri na server i spremanje podataka za učenje u relacijsku bazu.
5. Računalni igrač dobiva evaluaciju pozicije od konkretnog learnera iz rapidminera.
6. Analiza nekoliko algoritama i opisa stanja, s obzirom na uspješnost igranja igre.
7. \* Mogućnost parametrizacije upita - learnera i prostora stanja - možemo napraviti eksperiment u kojem jedan igrač igra samo sa podskupom atributa/određenim learnerom
8. \* Statistika raznih metoda učenja i prostora hipoteza na webu.
9. \*\* Implementacija jednoog relativno uspješnog algoritma sa strane klijenta.

Ne mogu procijeniti koliko će cijeli projekt zahtjevati posla, pa sam sa zvjezdicama označio dijelove koje smatram opcionalni(ji)ma.

Očekivani rezultati predloženog projekta

Očiti rezultat ovog projekta je implementacija igre koja bi bila na kompetitivnom nivou s ljudskim igračima (nadam se). Projekt bi također poslužio i kao primjer korištenja RapidMinera programatski, preko njegovog APIja i njegovo integriranje u šire softversko okruženje u svrhu istraživanja.

Nadalje, kako web-tehnologije sve više ulaze u razna područja ljudskog rada, prirodno se pojavljuje potreba za upotrebom metoda strojnog učenja u kontekstu istih, te bi ovaj projekt bio primjer izgradnje jednog takvog sustava.

Bilo bi idealno kada bih uspio završiti zadnju točku (implementacija samog algoritma sa strane klijenta) – osim praktičnosti (puno manje opterečenja na server) - demonstrirali bismo i metodu paralelizacije jednog algoritma na klijentska računala, čime jako povećavamo količinu resursa – što je svakako korisna stvar u strojnom učenju općenito.

Popis literature

* Tom Mitchell: Machine Learning, McGraw Hill, 1997.
* <http://en.wikipedia.org/wiki/Quantum_tic_tac_toe>
* <http://pages.cs.wisc.edu/~dyer/cs540/>
* J. N. Leaw, S. A. Cheong: Strategic Insights From Playing the Quantum
* Tic-Tac-Toe, <http://arxiv.org/abs/1007.3601>[quant-ph]
* <http://www.admin.cam.ac.uk/offices/research/documents/local/events/downloads/tm/06_ThoreGraepel.pdf>
* Lin Wu and Pierre Baldi: A Scalable Machine Learning Approach to Go, *Advances in Neural Information Processing Systems*, *19*, 1521.
* <http://rapid-i.com/wiki/index.php?title=Integrating_RapidMiner_into_your_application>
* <http://ai.arizona.edu/intranet/papers/WebMining.pdf>