Cautare in arbore de tip Monte Carlo folosita ca strategie in jocurile de masa

Sabau Andrei-Mircea

Scopul prezentarii

Scopul acestei lucrari este de a prezenta informatii despre un subcapitol al inteligentei artificiale si documentarea construirii unui bot cu ajutorul cautarii in arbore de tip Monte Carlo (MCTS – Monte Carlo Tree Search). De asemenea, aceasta prezentare contine avantajele si dezavantajele folosirii acestei metode, rezultatele obtinute prin aplicarea ei, solutii pentru imbunatatirea timpului de executie, reducerea memoriei folosite si concluzii personale.

Motivatia alegerii temei

Datorita faptului ca tot ce ne inconjoara incepe sa fie automatizat intr-o mai mica sau mare masura, (indiferent unde ne aflam, suntem serviti de anumite instrumente independente care ne fac viata mai simpla, fie ele controlate prin simple instructiuni de cod sau prin inteligenta artificiala), oamenii incep sa aiba o atractie fata de aceasta autmoatizare si o integreaza uneori inconstient si mai mult in viata lor de zi cu zi.

Traim intr-o lume in care nu trebuie sa te chinui sa scrii de mana saptamanal o lista de cumparaturi, ai o aplicatie si aceasta iti reaminteste sa cumperi un anumit produs pentru ca pe baza achizitiilor tale din ultimele 3 luni, ai cumparat produsul X in fiecare saptamana.

La fel si in cazul cautarii in arbore de tip Monte Carlo, beneficiile sunt extraordinare. Industria gaming-ului este revolutionata datorita faptului ca apar noi provocari pentru jucatorii de top, apar noi strategii care sunt preluate din domeniul inteligentei artificiale si imbunatatite, apare mai multa diversitate datorita parametrilor pe care ii suporta sistemul.

Luand in considerare faptul ca mereu am fost o persoana care incearca sa gaseasca explicatia matematica din spatele lucrurilor si mereu am fost fascinat de scalabilitatea lucrurilor, precum viteza de procesare a anumitor instructiuni, am ales sa scriu despre aceasta tema pentru ca, consider ca acesta este prezentul si viitorul.

Scurta istorie si rezultate

Metoda Monte Carlo, bazata pe esantionare aleatorie a simularilor, dateaza din anii 1940. Bruce Abramson a explorat ideea In teza de doctorat din 1987 si a spus ca metoda "se dovedeste a fi precisa, usor de estimat, eficient de calculat si independenta de domeniu".

Jocul Go

In 1992, B. Brügmann a folosit-o pentru prima data Intr-un program care juca jocul Go.

In 2008, MoGo a obtinut nivelul dan (master) la Go pe o taba 9 × 9 Go, iar programul Fuego a Inceput sa castige cu jucatori puternici de amatori la Go-ul pe o tabla 9 × 9 Go.

In ianuarie 2012, programul Zen a castigat cu 3:1 Intr-un meci de Go pe o placa de 19 × 19 cu un jucator amator de 2 dan.

Apoi a venit randul lui Google. Google Deepmind a dezvoltat programul AlphaGo, care, In octombrie 2015, a devenit primul program de Computer de Go care l-a Invins pe un jucator profesionist uman Go pe o placa de dimensiuni mari de 19x19.

In martie 2016, inteligenta artificiala a lui Google, AlphaGo a primit un titlu onorific de 9-dan (master) la Go-ul jucat pe o tabla de 19 × 19 dupa Infrangerea campionului mondial uman la Go, Lee Sedol, Intr-un meci de cinci meciuri, cu un scor final de patru jocuri la unu (4:1).

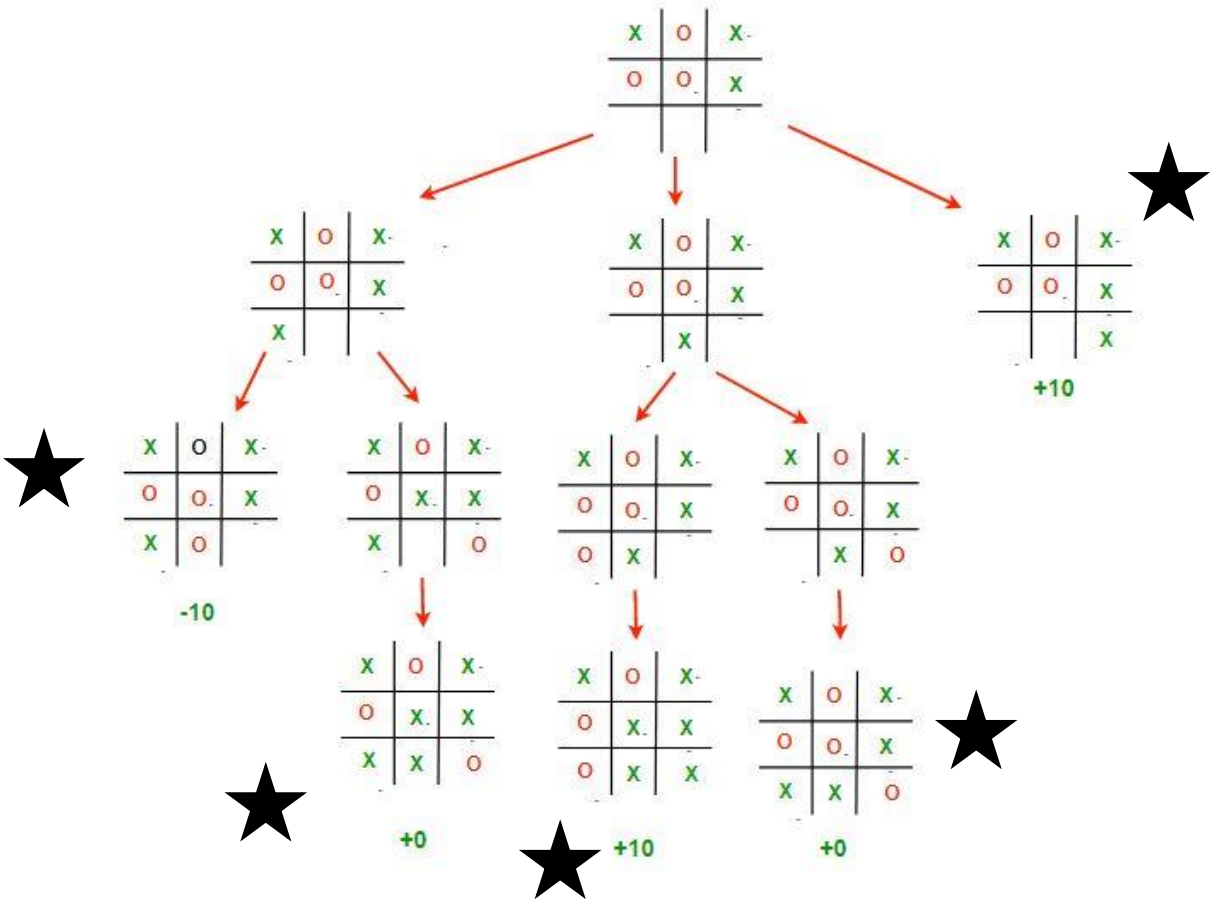
Jocul sah

Recent (In decembrie 2017) Google a anuntat ca noua sa inteligenta artificala, Alpha Zero, l-a Invins fara drept de apel pe campionul mondial la sah, ScottFish (si el o inteligenta artificiala), Intr-un meci in care din 100 de partide nu a pierdut nici una, avand 25 de victorii si 75 de remize. Inginerii de la Google au anuntat ca noua inteligenta artificala AlphaZero, foloseste tot metoda Monte Carlo combinata cu Retelele Neuronale.

Cautare in arbore de tip Monte Carlo

Aceasta metoda se bazeaza pe generarea de mutari in ordine aleatorie, fiecare mutare fiind considerata un nod intr-un arbore. Radacina acestui arbore este punctul de plecare pentru primul set de simulari ale jocului si creeare statisticilor. Luand in considerare anumite rationamente simple, ajungem la concluzia ca arborele poate simula mutari dar, nu poate sari peste mutari (ex: intr-un joc de X si 0, avand 2 mutari pe tabla, nu putem sari peste simularea celorlalte miscari si sa ajungem direct la forma finala a jocului). Dupa generarea unui set de mutari, se ia fiecare mutare si se trag anumite concluzii precum: daca jocul se poate continua din punctul respectiv, daca meciul este pierdut sau castigat, daca jucatorul pierde puncte mutand in pozitia respectiva, etc.

Evaluarea unei mutari in care jucatorul a ajuns la o stare finala a jocului se face cu o functie numita ‘fitness’ care atribuie fiecarui nod terminal (frunza) un anumit punctaj, astfel facand selectia mult mai usoara.



In imaginea de mai sus se poate vedea simularea si evaluarea mutarilor intr-un joc de X si 0. Nodul cel mai de sus reprezinta radacina si primul nivel reprezinta posibillele mutari pentru X. Pe primul nivel avem 2 stari intermediare si o stare finala. Starea finala este marcata cu un star. Starea finala este datorata faptului ca X castiga runda, nodul respectiv fiind eveluat de functia fitness si primind +10 puncte datorita castigului. O multitudine de jocuri este apoi simulata, extinzand nodurile 1 si 2 de pe primul nivel, pana se ajunge la frunze.

Ca si urmatoare mutare, AI-ul v-a alege sa castige jocul printr-o singura mutare, considerand distanta de la radacina la frunza respectiva ca fiind cea mai mica, aceasta eliminand posibilele pierderi.

Modalitati de joc cand folosim MCTS:

Agresiva:

1. Facem o mutare care ne va asigura castigul
2. Alegem o mutare care ne va aduce mai aproape de castig

Defensiva:

1. Facem o mutare care impiedica oponentul sa castige in urmatoarea runda
2. Facem o mutare care impiedica oponentul sa ajunca la un castig in urmatoarele runde

Posibile imbunatatiri ale algoritmului

Aceasta metoda poate fi imbunatatita prin reducerea spatiului de cautare in arbore. Considerand un joc de Sah pe o tabla de 8x8, fiecare nivel in arbore creste exponential si suntem limitati de memoria calculatorului si puterea lui de procesare. Doar pentru primul nivel din arbore am avea de simulat toate mutarile posibile pentru fiecare piesa de pe tabla de sah, care sunt extrem de multe. Facand o abstractie banala, sa zicem ca la fiecare runda avem alte 20 de mutari posibile (in realitate fiind mult mai multe), considerand 5 nivele de arbore, ajungem in final sa avem aproximativ 20^5 = 3.200.000 simulari ale jocului, despre care avem nevoie de configuratia tablei, statistici, etc. Considerand o implementare proasta a tablei pe o matrice de string-uri de 8x8 avem o tabla de 64 bytes necesara pentru stocarea datelor. Considerand ca si statisticile si alte variabile ajung la o marime de 36 bytes pentru fiecare tabla, avem in final 100bytes necesari pentru fiecare simulare. Daca propagarea datelor prin arbore nu este facuta eficient, si reducerea spatiului de cautare nu este eficientizat la maxim, la nivelul 5 avem :

100bytes + 100*20 bytes + ... 100*20^5 bytes = aproximativ 336.800.000 byes

= 328.947 MB

= 321 GB memorie RAM

necesara pentru a rula 5 nivele din tabla de sah, doar 5 nivele.

Din aceste motive, reducerea spatiului de cautare cu posibile mutari vulnerabile atat pentru jucator cat si pentru oponent, incetarea simularii jocului in cazul unu sah mat, valorificarea pierderii pieselor importante ar putea creste exponential eficienta. Pe langa asta, implementarea folosind-use mai putina memorie: piesele sa fie tinute in perechi de forma (x, y, tip piesa) si renuntarea la tabla ar aduce un beneficiu mare.

Opinie personala

Consider ca aceasta metoda este pe cat de simpla pe atat de pretentioasa cand vine vorba de oprimizari si pe atat de puternica. Ca si avantaje as subilinia faptul ca este usor de implementat, refolosibila in alte jocuri, deoarece nu necesita cunostinte despre domeniu, algoritmul trage concluzii pe baza functiei fitness care poate fi rescrisa in functie de domeniu, codul MCTS-ului ramanand la fel. Pe langa asta, prin eficientizarea cautarii in arbore, putem cauta in sectiunea cea mai interesanta a arborelui (ex: sectiunea care ne ofera cele mai multe puncte fitness in drumul cel mai scurt de la radacina la nod). In plus, in functie de memoria disponibila, putem sa oprim cautarea in arbore oricand, folosindu-ne de rezultatele pana in acel moment.

Ca si dezavantaje as subilina faptul ca timpul pentru gasirea unei strategii bune, creste direct proportional cu numarul de simulari pe care arborele le produce. Pe langa asta, fara o functie buna de fitness cautarea in arbore este inutila pentru ca o strategie buna nu se poate distinge de una proasta si exista cazul reducerii spatiului de cautare pentru eventuale mutari de success.

Concluzie

Metoda de cautare in arbore de tip Monte Carlo are avantaje si dezavantaje. Din rezultatele oficiale, pana in acest moment, aceasta dovedeste a avea mai multe avantaje decat dezavantaje. Consider ca aceasta metoda exploatata la maxim, cu o functie de fitness buna, cu optimizari facute pe spatiul de cautare si pe timpul de executie, este o metoda aproape indestructibila in ceea ce priveste industria de gaming. Pe langa asta, aplicabilitatea ei pe jocuri care nu au mai mult de 20-30 de mutari si mai putin de 10 posibile mutari pe runda, este incredibila.