Monte Carlo Tree Search for å spille GO

av Erik Borgeteien Hansen og Oda Alida Fønstelien Hjelljord

Takk til Markus Johannes Pedersen for all hjelpen gjennom prosjektet. Takk til Ole Christian Eidheim, Donn Morrison, Jonathan Jørgensen for undervisning som vi brukte i dette prosjektet og for utsettelse som tillot oss å fullføre det.

Abstract

I dette prosjektet ble det utviklet en implementasjon av Monte Carlo Tree Search (MCTS) som kunne benytte et konvolusjonelt nevralt nettverk (CNN) til å gjøre simuleringer av spill.

Kjøringer av MCTS ble brukt til å utforske spill og disse spillene ble brukt til å trene CNN. For teste treningen, ble modeller med forskjellige mengder trening spilt mot hverandre i en turnering.

En feil i MCTS implementeringen førte til at CNN-modellene ble mer flinke som hvit-spiller enn svart-spiller.

[TL:DR for resultater]

Oppgavebeskrivelse

I denne oppgaven var målet å implementere en kombinasjon av Monte Carlo Tree Search (MCTS) og et nevralt nettverk for å spille Go. Aigagrors GymGo [1] var oppgitt som miljø for Go. Oppgaven var relativt fri i forhold til hvordan det nevrale nettverket skulle implementeres.

MCTS kombinert med et nevralt nettverk er grunnlaget for AlphaGo. Oppgaven var i stor grad lagt opp til å lage en enkel, nedrustet versjon implementasjon av denne.

Tidligere relevant arbeid

Det mest relevante tidligere arbeidet for denne oppgaven er AlphaGo. Grunnlaget til AlphaGo er også MCTS kombinert med et nevralt nettverk. AlphaGo er kanskje verdens mest kjente spill AI og det første programmet til å slå en profesjonell Go-spiller. Den bruker både avanserte søketrær og dyp læring til å spille Go bedre enn noen tidligere algoritme. [2]

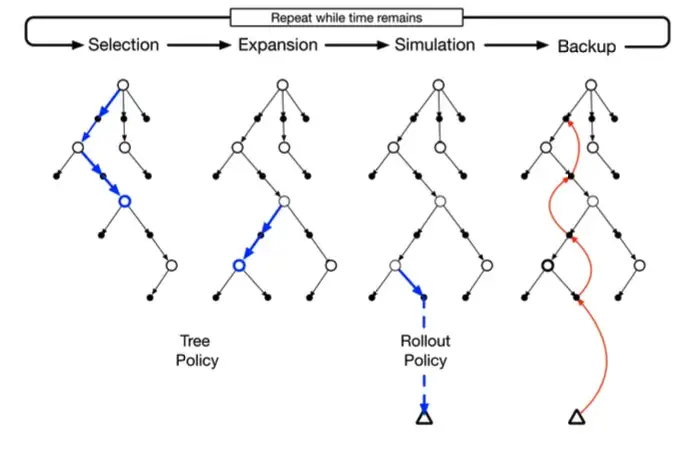
Teori

Go

Go er et abstrakt strategi brettspill for to spillere. Det ble funnet opp i Asia for mer enn 2500 år siden og er sannsynligvis det eldste brettspillet fortsatt i bruk. [3] Det spilles med svarte og hvite steiner som plasseres på åpne plasser på brettet. Utplasserte steiner kan ikke flyttes, men kan fjernes ved at de blir omringet av motstander steiner (fanget). Dersom en spiller ikke ønsker å legge ut en stein kan de velge å stå over turen sin. Spillet slutter når begge spillere står over turen sin. [4] Selv om reglene i Go er relativt enkle, er Go et ekstremt komplekst spill. Sammenlignet med sjakk har Go et større brett, lengre spill og, i gjennomsnitt, mange fler mulige trekk per tur. Antallet mulige posisjoner på et standard 19x19 brett har blitt kalkulert til omtrent 2.1×10170.[5]

Monte Carlo Tree Search

MCTS er en algoritme som legger grunn for mange suksesser i maskinlæring, som for eksempel AlphaGo. Den bruker simulering til å gi en forventningsverdi til nodene i treet og fokuserer på mer lovende noder. På denne måten unngår man å teste alle trekkene som er mulig, noe som ville vært uholdbart tidkrevende for Go. [6] Algoritmen består av fire hovedsteg; seleksjon, ekspansjon, simulering og tilbake-propagasjon. Gjennom disse blir de mest lovende grenene i treet utforsket i ønsket antall iterasjoner eller til et komplett spill blir oppdaget. Algoritmen går gjennom stegene som vist i illustrasjonen under:

[Bilde: 6]

I seleksjons steget bruker vi en seleksjons policy til å velge det mest lovende barnet til hver node. Når en løvnode som enten ikke er besøkt før eller den noden som er mest lovende i forhold til UCB1 formelen. UCB1 formelen er seleksjons policyen bruk i denne oppgaven, det er også policyen brukt av AlphaGo som ble nevn tidligere. UCB regnes ut på følgende måte.

UCB1(node) = node value + c log(total trials)node trials [7]

Ekspansjon gjøres når en node har blitt besøkt før. Det gjøres ved å legge til alle lovelige trekk etter trekket til noden som barn til noden. I det tilfellet dette gjøres velges et av barna for å simulere på. [7]

Under simulering (Rollout) spilles det ut et spill etter trekket i noden. Dette gjøres tilfeldig i vanlig MCTS, men i AlphaGo og i denne koden brukes et nevralt nettverk til å vekte hvilke trekk som velges. Når spillet er ferdig tilbake-propageres result opp gjennom treet til alle foreldrene til noden. resultatet legges til på verdien og antall besøk økes til hver node. [6]

Konvolusjonelle nevrale nettverk

Et konvolusjonelt nevralt nettverk er en type kunstige nevrale nettverk som inneholder et konvolusjonslag, oftest brukt til å analysere bildedata. Denne typen nevralt nettverk er fordelsaktig når man vil kjenne igjen trender i dataen, og gjør for eksempel at plassering av objekter i bilder er mindre viktig. Go-miljøet som brukes i denne oppgaven har egenskaper som ligner digitalt lagrede bilder, som gjør det mulig å bruke slike nettverk i denne sammenhengen. Målet er at modellen skal lære seg hvordan gode trekk ser ut, og deretter finne igjen slike trekk uansett hvor de befinner seg på brettet.

<https://course.elementsofai.com/no/5/3>

Metode

Arbeidet på prosjektet startet med å implementere Monte Carlo Tree Search (MCTS). MCTS treet ble opprinnelig opprettet med totalt antall rollouts for å gi til ucb, men dette kunne enkelt oppnås med å bruke rotnodens antall tester. De nødvendige parameterene for treet var rotnoden og maskinlærings modellen. Backpropagation og selection ble implementert som metoder i treet. Det ble lagt til metoder for å hente vekting av trekk fra modellen lagret i treet under rollouts. Det ble implementert en metode for å kjøre tresøk til et spill var fullført og en metode for å hente treningsdata fra treet. Denne dataen ble brukt for å trene modellen.

Nodene i treet ble implementert med en kopi av spillmiljøet, forventningsverdien og antallet ganger den den har blitt testet. De hadde metode for å regne ut ucb-verdien ved hjelp av ucb1, gjennomføre rollout og ekspansjon.

Vi ser på go brettet som et “bilde”, litt om GymGo hovedsakelig hvordan states er “bilder”

Deretter bruker vi MCTS til å simulere og

For å bygge modellen vår har vi brukt PyTorch, hvor vi enkelt kan sette sammen en modell lagvis. Modellen bruker to konvolusjonslag, etterfulgt av todimensjonale “max pool” lag. Til slutt har vi et fullt sammenkoblet lag som gir oss en sannsynlighetsfordeling av alle mulige trekk, der høyest verdi er modellen sin prediksjon på beste trekk. Siste lag bruker “softmax” for å skalere prediksjonen.

Videre har modellen blitt trent på spill generert av MCTS, der trekk med høyest verdi har vært “korrekt” trekk. Dette har vært mulig å implementere fordi egenskapene til Go-miljøet ligner de som vi finner i digital representasjon av bilder. Vi har et rutenett av data, og flere kanaler. Hver kanal holder her på informasjon om hvor de sorte og hvite brikkene er plassert, hvem sin tur det er, ugyldige trekk, om forrige trekk var pass og om spillet er slutt. På denne måten har vi trent modellen til å predikere hvilken trekk som er best i den forstand at trekket gir størst vinnersannsynlighet framover.

hvordan turnering?

Plukka ut 4 nivåer av trent modell til turnering, 1 random spiller og et tre uten modell

Resultat

MCTS

I testing av MCTS mot en spiller som velger tilfeldige trekk (tilfeldig-spiller) ble det oppdaget en feil i implementasjonen som førte til et bias for hvit spiller for MCTS. Sannsynligheten for MCTS til å vinne mot en tilfeldig-spiller ble beregnet til x% dersom MCTS spilte som svart og x% dersom MCTS spilte som hvit.

Resultat av turnering

Diskusjon

Go

Det ble tidlig i prosjektet klart at manglende dokumentasjon av Go miljøet [1] var en stor hindring. Mye av tid som kunne blitt brukt på andre ting, endte med å bli brukt på å lage metoder som allerede var innebygd i miljøet eller å lese kildekoden til miljøet for å finne disse. Miljøet blir også ikke vedlikeholdt og dermed måtte tid brukes på å finne versjoner av biblioteker som passer med miljøet.

Liten brettstørrelse er dumt, modellen vår er sikkert ikke god i ordentlige spill

Det ble valgt å bruke heuristisk metode for å beregne resultatet av en simulering. Dette gjør at en stor seier er mer verdifullt for MCTS. Dette medfører også at komi eller handicap i Go kommer med. Dette er en verdi som trekkes fra svart spillers score for gjøre opp for fordelen av å være første spiller. For denne oppgaven var ikke komi relevant siden 5x5 brett har en komi på null, men for videre utvikling ville det bli viktig å oppdatere koden for komien relativt til brett størrelse.

MCTS

Med mer tid til å feilsøke hadde det vært mulig å finne årsaken til og korrigere feilen som gjør MCTS til å spille bedre som den ene fargen enn den andre. Det er enkelt å endre biaset fra svart til hvit ved å fjerne koden i tilbake-propageringen som gjør at resultatet blir trukket fra node verdien i stedet for lagt til dersom det er hvit sin tur. Dette gjør den hvite spilleren til å spille så dårlig som mulig. I implementasjonen som er lagt ved er det gjort et forsøk på å gjøre begge spillere bra. Dette har gjort hvit spiller bra, svart spiller dårligere enn tilfeldig, men ikke så dårlig som mulig.

Siden arbeidet startet med å utvikle MCTS ble denne implementasjonen gradvis utvidet etterhvert som nye behov ble oppdaget. Dette førte til at koden til MCTS ble mer uorganisert enn nødvendig. I videre utvikling kunne det vært ønskelig å lage en ny implementasjon eller refaktorere den eksiterende med den kunnskapen vi nå har.

Siden det ble laget en klasse for treet og en for nodene ble det gjort valg om hvilke metoder som skulle plasseres hvor. Dette var ikke absolutt nødvendig siden det er mulig å lage en implementasjon som bare har en nodeklasse, men det var ønskelig å ha en treklasse for to hovedgrunner: tilgang til CNN-modellen og enkel tilgang til totalt antall simuleringer. Siden treet skulle ha en CNN-modell som den kunne hente trekkvekter fra sparte det lagringsplass å ikke ha denne lagret i hver node. I forsøk på implementasjon av MCTS med bare nodeklasse ble det nødvendig å søke opp i treet etter totalt antall simuleringer. Selv om å traversere oppover i treet ikke er veldig tidkrevende, brukes ekstra tid hver seleksjon.

Det var usikkerhet rundt hvordan MCTS skulle generere treningsdata til CNN. Dette gikk hovedsakelig på hvor lenge tresøket skulle kjøre. Det vanlige for MCTS i henhold til litteraturen [6][7] er å kjøre søket i en gitt mengde iterasjoner, men det ble oppdaget at etter en stor mengde iterasjoner kjørte søket bare gjennom den samme grenen gjentatte ganger. Det ble også indikert fra veileder at MCTS skulle kjøre til et spill var fullt utforsket så det var det som ble gjort.

En av grunnene til at det er ønskelig å kombinere MCTS med et nevralt nettverk er at MCTS er veldig offensiv. Siden den bare fokuserer på de mest lovende trekkene er det mulig at det er et veldig bra spill den kan gjøre etter et trekk, men også et veldig bra trekk motstanderen kan gjøre. Dermed kan MCTS gi motstanderen en åpning. Det er derfor ønskelig å hjelpe dette med å kombinere MCTS med et konvolusjonelt nevralt nettverk (CNN) for å prøve å se trender på hva som er gode trekk.

CNN

PyTorch ble valgt over de andre kjente maskinlæringsrammeverkene da dette er det gruppen har hatt mest erfaring med, og dets støtte for å kjøre operasjoner på GPU.

Modellene virker som de blir for opptatt av den første treningen og videre trening ikke hjelper som forventet

[hvorfor turnering?]

Dette prosjektet ble sterkt påvirket av sykdom, begge gruppemedlemmene var i lengre perioder for syke til å gjøre arbeid på prosjektet.

Konklusjon

På tross av utfordringer med sykdom og spillmiljøet ble det utviklet en implementasjon av MCTS som utforsket Go spill og trente et konvolusjonelt nevralt nettverk (CNN) på disse dataene. Det ble gjort mulig for CNN å bidra til bedre simuleringer som gir MCTS mer pålitelige simuleringer. Etter treningen av CNN ble nøyaktigheten til modellen økt. Dette betyr også at en MCTS-simulering fra et tidlig trekk har mindre sannsynlighet for å være et avvik, som for eksempel at begge spillerne trekker seg med en gang.

[Kommentar på turneringsresultater]

Referanser

1. <https://github.com/aigagror/GymGo>
2. <https://www.deepmind.com/research/highlighted-research/alphago>. Hentet 1.12.2022
3. <https://www.usgo.org/brief-history-go>. Hentet 1.12.2022
4. Iwamoto, Kaoru (1977). Go for Beginners. New York: Pantheon. ISBN 978-0-394-73331-9. Hentet 1.12.2022
5. Tromp, John; Farnebäck, Gunnar (31.1.2016). ["Combinatorics of Go"](https://tromp.github.io/go/gostate.pdf) (PDF). tromp.github.io. Hentet 1.12.2022
6. <https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168>
7. [http://www2.agroparistech.fr/ufr-info/membres/cornuejols/Teaching/AGRO/Cours-IA/Tr-intro-MCTS-2020.pdf](http://www2.agroparistech.fr/ufr-info/membres/cornuejols/Teaching/AGRO/Cours-IA/Tr-intro-MCTS-2020.pdf?fbclid=IwAR37VsuwS-67ZLHG3myqcIflck6Csb-jqwDnw-toH06wg_G-kf31vvhrtYc). Hentet 26.10.2022

