**Hearthstone AI: Prediksjon av beste valg ved bruk av MCTS og Supervised Machine Learning**

TDAT3025 Anvendt maskinlæring

Skrevet av Emir Derouiche

23.November 2020

# Introduksjon: Hvorfor Hearthstone?

## Maskinlæring og brettspill:

Maskinlæring blir er kjent for å bli anvendt for finne gode strategier for brettspill som sjakk og go hvor tradisjonelle framgangsmåter krever en ikke-triviell mengde prosesseringskraft. Slike spill har lenge vært en av de få resterende intelligenstestene hvor mennesker kan beseire datamaskiner. Dette kommer av menneskers evne til å filtrere ut ugunstige trekk selv når antall mulige kombinasjoner av trekk er ufattelig høye. Mennesker kan gjøre dette på grunn av vår evne til å lære fra erfaring og det er nettopp denne evnen vi prøver å gi til datamaskiner med maskinlæring.

## Hearthstone, et moderne brettspill:

Hearthstone: Heroes of Warcraft er et moderne brettspill utviklet av dataspillprodusenten Blizzard Entertaiment. Hearthstone kan beskrives som en kombinasjon av sjakk og poker. Hearthstone ligner på sjakk fordi man har brikker på slagfeltet som kan angripe fiendens brikker, hvor strategisk valg av angrepsmål er relevant. Hearthstone har allikevel mye tilfeldighet og skjult informasjon slik som i poker ettersom spillerne trekker tilfeldige kort og ikke vet hva slags kort motstanderen har på hånden. Evnen til å forutse motstanderens kort og fremtidige trekk er det som skiller nybegynnere fra mestere[1]. Da Hearthstone kom ut i 2014[2] vekket det umiddelbart interesse hos maskinlæringsmiljøet. HearthSim[3] er en gruppe utviklere som slo seg sammen for å skape et eget miljø rundt Hearthstone hvor man kan dele verktøy og erfaringer som kan være nyttige for andre utviklere. For mitt prosjekt valgte jeg å bruke en av simulatorene HearthSim har utviklet: fireplace[4].

## Inspirasjon og begrensninger:

Den opprinnelige tanken for dette prosjektet var å benytte Monte Carlo Search Tree (MCST) i kombinasjon med deep learning i likhet med Deepminds AlphaGo[5]. Jeg endte opp med å benytte supervised learning i stedet for deep learning. Det finnes flere steder hvor deep learning kan anvendes for å forbedre systemet mitt, men for dette prosjektet gikk jeg dessverre tom for tid før jeg kunne begynne å implementere slike optimaliseringer. Min bruk av MCST ble også annerledes fra Deepmind sin framgsmåte ettersom systemet mitt i stor grad ble simplere enn AlphaGo grunnet åpenbare tidsbegrensninger og teambegrensninger. Dette vil bli utdypet i diskusjonsdelen av denne artikkelen.

# Teori

## Begrensnigner for tradisjonell Hearthstone AI:

Hearthstone har en divers mengde problemer hvor maskinlæring kan anvendes, noe som er godt utdypet i artikkelen «The Many AI Challenges of Hearthstone»[6]. For dette prosjektet snevret jeg inn omfanget til å bare se på angrepsmål for brikkene til spilleren. Hearthstones innebygde AI har en tradisjonell framgangsmåte, men denne er i stor grad mangelfull, dette er årsaken til at maskinlæring kan anvendes for å gi bedre resultater. En viktig årsak til at en maskinlæringsalgoritme kan utkonkurrere Hearthstone AIen er at forgreningsfaktoren for Hearthstone er ufattelig høy. Dette kommer av at Hearthstone tillater flere handlinger per tur og mengden handlinger kan gå mot uendelig. Mengde kombinasjoner av handlinger per tur vokser i tillegg eksponentielt i forhold til antall handlinger. Hearhstones AI måtte dermed ha store begrensninger for å oppnå sanntidsevaluering av trekk. Et av de største begrensningene er at AIen kun ser på kombinasjoner av opp til to kort når det kommer til angrepsmål[1].

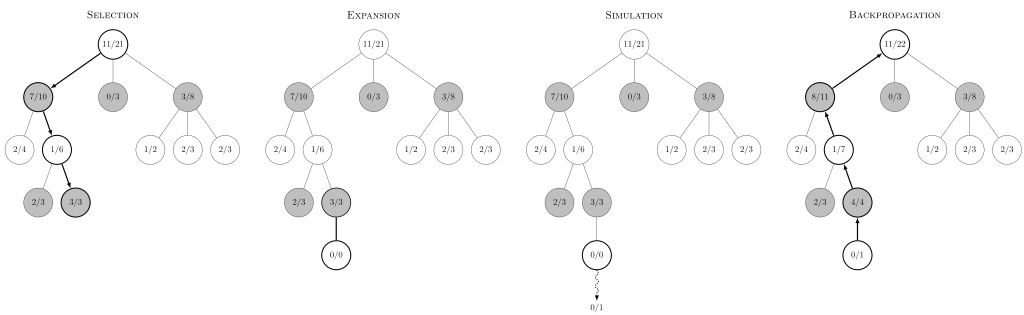


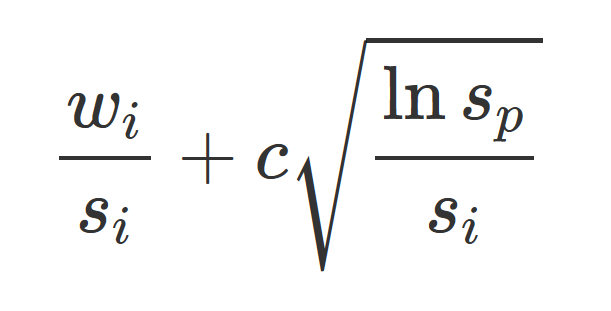
Figur 1: Et hearthstone spill scenario hvor spilleren bruker kortet "Doomsayer" som vil fjerne alle brikker fra brettet om den ikke blir fjernet innen spilleren sin tur

Her ser du et eksempel hvor en spiller har spilt kortet «doomsayer», her må motstanderen angripe den for å unngå å miste alle sine brikker. Denne begrensingen har blitt fikset i nyere versjoner av spillet ved å hard-kode unntak, men vi unngår slike begrensinger helt ved å bruke maskinlæring.

Monte Carlo Search Tree (MCST):

MCST er en AI algoritme som ikke bruker maskinlæring, men som allikevel kan finne seg fram til optimale trekk i ukjente sitasjoner ved hjelp av simulering[7][8]. MCST er oppkalt etter Monte Carlo, fordi i likhet med gamling baserer den seg på tilfeldige utfall. MCST bruker også et tre som forgrener seg ut av initielle staten med trekk som den skal evaluere. MCST forgrener seg ved å gå gjennom fire distikte steg: selection, expansion, simualtion og backpropagation.



Figur 3: Et eksempel på et Monte Carlo Search Tree. Nodene representerer stater og verdiene deres er antall vinn delt på antall simuleringer. Foreldrenoder arver verdiene til barna deres.

Selection er går ut på å velge hvilken løvnode vi skal utvide. I tradisjonell MCST er Upper Confidence bounds (UCB) formellen brukt for å balansere utforskning og utnytting. UCB tar node «verdien» som er antall vinn delt på antall simuleringer og plusser på en konstant C ganger roten til den naturlige logaritmen av totalt antall simuleringer delt for foreldrenoden på antall simuleringer for noden vi undersøker. Størrelsen på C avgjør altså hvor mye vekt vi legger på utforskning. Vi kjører selection algoritmen til vi kommer til en løvnode. I dette steget kan vi benytte maskinlæring for å gi algoritmen et ekstra hint om hva slags trekk som er gode og lede den mot de for å øke effektiviteten på søkene og dog simuleringene.

Figur 3: UCB formelen

Expansion er å gi løvnoden vi har funnet ved hjelp av selection nye barn. Expansion gjøres som oftest ved å kun legge til et barn, men man kan også legge til flere. For dette prosjektet legger vi til alle mulige barn for å simplifisere selection og øke effektiviteten. Ettersom en node av mine noder kun kan ha maks åtte barn er det ikke farlig å utvide slik. Et barn representerer en ny stat hvor ett trekk har blitt spilt ut ifra staten til foreldrenoden.

Simulation er at man simulerer resultatet av staten i noden. Dette kan gjøres med forskjellige framgangsmåter, men som oftest spilles spillet ferdig med utgangspunkt i nodes stat ved å spille tilfeldige trekk fram til en av spillerne vinner. Dette er selvsagt ikke en nøyaktig indikasjon på om trekket som ble gjort i expansion var gunstig eller ugunstig, men ved å kjøre mange slike simuleringer vil man ved hjelp av store tall få et godt estimat om trekket var smart. Simulation gir noden en verdi på 0/1 om den taper eller 1/1 om den vinner. Her kan man også bruke maskinlæring for å gi bedre simuleringer enn rent tilfeldige trekk.

Backpropagation er at man vandrer opp treet fra noden man simulerte og oppdatere alle foreldre med verdien fra den nye simuleringen.

Supervised Learning:

Supervised Learning er en populær metode for maskinlæring. Algoritmen får en input som er tall den skal bruke for å komme fram til en output ved hjelp av tensor transformasjon. Deretter sammenligner den resultatet med en fasit og optimaliserer seg selv slik at neste gang en lik input kommer vil outputen være nærmere fasiten[9]. Denne optimaliseringen skjer vanligvis ved hjelp av en gradient decent optimaliseringsalgoritme. Ulempen med Supervised Learning er at den krever store mengder data hvor fasiten er gitt. Dette er essensielt for algoritmens evne til å forbedre seg, men innsamling er ressurserintensivt. For dette prosjektet har vi dermed brukt MCST for å genere testdataen vår. Ulempen med dette er at fasiten ikke nødvendigvis er korrekt og nøyaktigheten til modellen vår er dermed sterkt korrelert til MCST sin ytelse. Motparten til Supervised Learning er Unsupervised Learning hvor det ikke finnes noen fasit og algoritmen kun grupperer data som ligner.

Forbedring av MCST ved hjelp maskinlæring:

Prediktor Upper Confidence Bounds (PUCB) er en utvidelse av UCT som bruker en prediktor for å gi UCT et ekstra hint om hvilke noder som bør velges under Selection fasen

# Tidligere relevant arbeid

# Framgangsmåte

Yrdy

# Resultat

Yrdy

# Diskusjon og videre arbeid

Played cards could be more relevant than attack targets

# Konklusjon

Yrdy

# Annerkjennelser

Jeg vil takke min veileder William Chakroun Jacobsen for hans veiledning.

Dette prosjektet ville ikke vært mulig uten Hearhtstone simulatoren Fireplace[4] og dens utviklere.

Til slutt vil jeg takke Maciej Swiechowski, Tomasz Tajmajer og Andrzej Janusz for deres artikkel[6] som var til stor hjelp for prosjektet mitt.

# Referanser

1. <https://www.gdcvault.com/play/1020592/AI-Postmortem>
2. <https://en.wikipedia.org/wiki/Hearthstone>
3. <https://hearthsim.info/>
4. <https://github.com/jleclanche/fireplace>
5. <https://deepmind.com/research/case-studies/alphago-the-story-so-far>
6. <https://link.springer.com/article/10.1007/s13218-019-00615-z>
7. <https://en.wikipedia.org/wiki/Monte_Carlo_tree_search>
8. <https://www.youtube.com/watch?v=Fbs4lnGLS8M&ab_channel=FullstackAcademy>
9. <https://en.wikipedia.org/wiki/Supervised_learning>