Djup förstärkt maskininlärning i en strategisk brädspels-miljö

Jimmy Fagerholm

Kandidatavhandling i datateknik

Handledare: Sepinoud Azimi

Fakulteten för naturvetenskaper och teknik

Åbo Akademi

2019

Abstrakt

I denna avhandling kommer maskininlärning tas upp samt en djupare genomgång av förstärkt maskininlärning med vikten på djup förstärkt maskininlärning. Läsaren kommer bekantas med algoritmer samt teorier som används inom maskininlärning och brädspel. Hur brädspel representerar i ett digitalt format för en agent inom förstärkt maskininlärning i form av spelträd och sökträd, hur agenter byggs upp med hjälp av Monte-Carlos sökträd men också hur policy inlärning används för att ge motspelaren en utmaning samt motstånd som påminner om, och ibland överträffar, professionella spelare inom dagens populära brädspel. Avhandlingen tangerar begränsningar så som standardiseringar och bristfälliga utvecklingsmiljöer vid implementationen av förstärkt maskininlärning i strategiska brädspel.

Innehållsförteckning

Förteckning över förkortningar och beteckningar

ML	Maskinlärning (*eng. Machine Learning*)
Dataset	En samling obearbetat data som innehåller sampel med flera särdrag.
Särdrag	Delar av ett objekt som gör att dom skiljs åt från varandra.
Etikett	Ett namn eller ord som definierar objekt, en rubrik, som används inom maskininlärning.
Själv spel	Då någon spelar mot sig själv.

Inledning

Denna avhandling kommer att introducera maskininlärning till läsaren samt en djupare genomgång av förstärkt maskininlärning med vikten på djup förstärkt maskininlärning. Läsaren kommer bekantas med algoritmer samt teorier som används inom maskininlärning och brädspel. Hur brädspel representerar i ett digitalt format för en agent inom förstärkt maskininlärning, hur agenter byggs upp med hjälp av såväl gamla algoritmer som nya metoder för att ge motspelaren en utmaning samt motstånd som påminner om, och ibland överträffar, professionella spelare inom dagens populära brädspel. Avhandlingen tangerar också begränsningar som förekommer vid implementationen av förstärkt maskininlärning i strategiska brädspel.

Varför implementera maskininlärning i brädspel

- Kan överföras till situationer i verkligheten.
- Kan användas som mått på intelligens. Är maskiner lika intelligenta som människan?
- För ekonomisk vinst, tävlingar?
- Kan användas som plattform för evaluering av effektiviteten hos algoritmer.

Brädspel som utvecklingsmiljö

Brädspel går lätt att simulera i en digital miljö samt definitionen på slutlägen i brädspel är ofta självklara, vilket betyder alltså en vinst eller förlust för en spelare. Det finns klara begränsningar vad en spelar får och inte får göra i ett brädspel i form av spelregler.

Dagens samhälle har också väl definierade sätt att mäta intelligensen hos dessa implementationer tack vare datainsamling från professionella turneringar där handlingar av mänskliga spelare används för att träna dessa agenter i bland annat klassiska brädspel som Schack [1] och Go [2] men också i digitala spel som Dota2 [3] och Starcraft2 [4]. Agenterna används sedan för att mäta systemens och implementationernas styrkor och svagheter.

Maskininlärning — introduktion och bakgrund

Maskinlärning är en underkategori till artificiell intelligens. Maskininlärning använder sig av olika matematiska algoritmer för att samla information från obearbetade data, ett dataset, för att representera den i en typ av modell. Sedan kan modellen används för att dra slutsatser om olärda data eller för användning i andra modeller.

Neurala nätverk är en av många modeller som tränas med hjälp av maskininlärning så att maskiner kan utföra handlingar som påminner om mänskligt beteende. Byggstenarna i ett neuralt nätverk kallas noder, vilket baserar sig vagt på den biologiska neuronen som finns i hjärnan hos ett däggdjur. Likaså byggs kopplingarna upp mellan noderna liksom i den biologiska hjärnan och utvecklas under processen (med hjälp av ”träning”). [5]

Tack vare att beräkningskapaciteten på systemen har ökat markant de senaste 20 åren har maskininlärningen exploderat och industrin har lyckats med beräkningstekniker som inte tidigare varit möjliga. En av teknikerna är djup maskininlärning, vilket kommer tas upp i denna avhandling mera grundligt lite senare. [5]

Maskininlärning kan grovt delas in i tre huvudkategorier, *oövervakad inlärning*, *övervakad inlärning* och förstärkt inlärning beroende på vilka möjligheter den har vid inlärningsprocessen. [6] Till näst kommen en överblick vad dom tre huvudkategorierna innebär.

Oövervakad inlärning

Algoritmer inom oövervakad inlärning *(eng. Unsupervised Learning)* använder sig av dataset som innehåller många särdrag för varje enskilt sampel, ett dataset innehåller mängder av sampel. Algoritmerna lär sig de nödvändiga egenskaperna för datasetet i fråga och kan till exempel skapa kluster av sampel som påminner om varandra. Inom oövervakad inlärning förekommer inga träningsexempel, istället försöker algoritmerna hitta mönster och korrelationer i datasetet. Oövervakad inlärning används inom bland annat filkomprimering samt gruppering av användare för personlig marknadsföring.

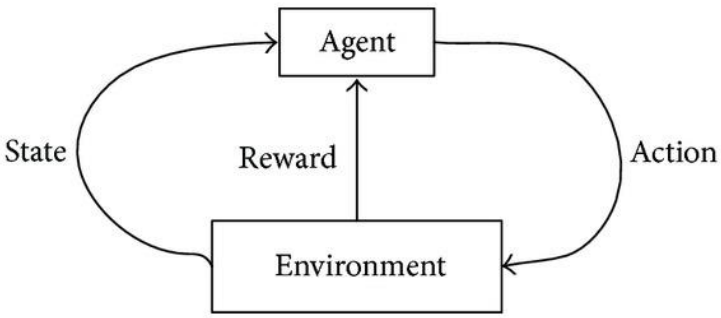
Övervakad inlärning

Inom övervakad inlärning *(eng. Supervised Learning)* används träningsdata i form av sampel som finns i ett dataset där alla sampel har en etikett *(eng. lable)*. Övervakad inlärnings algoritmer är klassificerings algoritmer. I övervakad inlärning försöker algoritmerna istället för att skapa kluster eller hitta samband mellan den data som bearbetas söker algoritmerna komma fram till ett svar, en etikett, från ett givet sampel som mest liknar samplen i datasetet.

Ett bra exempel på var övervakad inlärning används är i spamfilter för e-post. För varje e-post granskas innehållet och modellen gör en bedömning av ifall mejlet är skräp eller inte.

Förstärkt inlärning

I förstärkt inlärning (*eng. Reinforcement Learning)* introduceras en agent till en miljö. Agentens uppgifter är att interagera med miljön och genom feedback från miljön maximera sin belöning. Agenten får en belöning när den utför en handling som leder till önskat resultat och ett straff, ofta definierad som en negativ belöning, när den utför en fruktlös handling. Ramverket för förstärkt inlärning ritas oftast upp enligt figur 1**.**



Figur 1 – Ramverk för förstärkt maskininlärning. [4]

Miljön i förstärkt inlärning är ofta modellerad med hjälp av en Markov-beslutsprocess *(eng. Markov decision process, MDP),* ett matematiskt ramverk för beslutsfattning i en miljö där slutsatsen är delvis slumpmässig och delvis bestämd. En mera fördjupad genomgång av MDP gås genom senare i denna avhandling. Agenten följer en policy där

det finns en mängd handlingar som den kan göra för varje olika läge av MDP. Agentens mål är att förbättra policyn för att maximera sin belöning. [5] [7]

Modellerna i alla tre kategorierna bearbetas oftast i artificiella neuronnät, till följande kommer en överblick om vad artificiella neuronnät är och hur dom byggs upp.

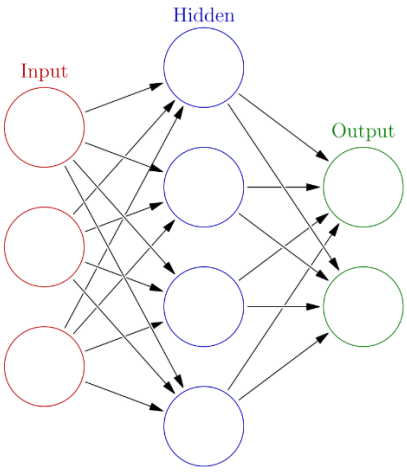
Artificiella neuronnät

Inom maskininlärning används sammankopplade noder, ett så kallat artificiellt neuronnät (enligt svenska datatermgruppen), också känt enbart som neuronnät eller bara ANN (*eng. Artificial nerual network)*. Artificiella neuronnät påminner mycket om det mänskliga neuronnätverket där neuroner är sammankopplade för att behandla och sända vidare information till sammankopplade närliggande neuron. [8]

I maskininlärning byggs ett neuronnät upp i lager där första lagret är ett inputlager, där ett neuron motsvarande en nod skapas för varje indata. Sista lagret är ett outputlager där varje nod motsvarar till exempel en handling eller en klassificering.

> *Neuronnät kan används i maskin läge eller regressions läge där maskin läge producerar en etikett, till exempel ”spam”/”inte spam” eller ”Ja”/”Nej”. Outputlagret kommer således att innehålla noder som motsvarar antalet etiketter.*
>
> *I regressions läge returnerar neurala nätverket ett värde, till exempel ett pris för en aktie eller mängd nederbörd. Outputlagret kommer i detta läge att innehålla en nod.*

Mellan första och sista lagret finns ett antal gömda lager vilka definieras enligt ändamålet. Mycket simpla neuronnät kan visualiseras enligt figur 2.



Figur 2 – Visualisering av ett artificiellt neuronnät. [9]

Ifall de gömda lagren överskrider 2 i neuronnäten man använder kallar man maskininlärningen för djup maskininlärning. Neuronnäten kan skapa sammankopplingar vilket påminner om till exempel olika strategier i ett brädspel.

Det pratas även mycket om sammanlänkade artificiella neuronnät (*eng.* convolutional neural networks, CNN) vilka är djupa neuronnät som i huvudsak används till klassificering av bilder.

Djup förstärkt maskininlärning

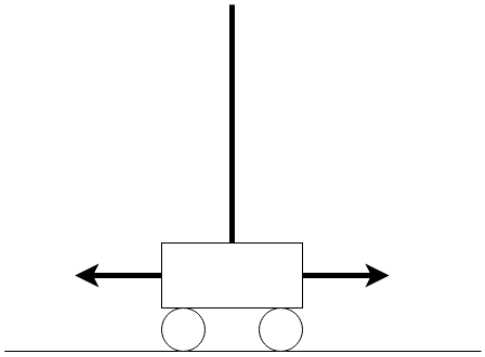
Nu när bakgrunden till maskininlärning och förstärkt maskininlärning är bekanta kommer den huvudsakliga delen av denna avhandling, till näst kommer djup förstärkt maskininlärning, dess användningsområden samt en djupare genomgång på hur den används inom brädspel så som tre i rad, schack, Go och sedan också hur djup förstärkt maskininlärning kunde användas inom en mycket bredare område inom brädspel.

Djup förstärkt maskininlärning är förstärkt maskininlärning där en agent interagerar med en miljö och genom feedback genomföra handlingar som strävar till önskat resultat och där besluten som agenten utför bearbetas i ett neuronnät med flera än 2 gömda lager.

Inlärningsprocessen består av en agent, en miljö och en belönings signal. Agenten väljer en handling att utföra i miljön vilket resulterar i en belöning. Agenten väljer handlingen genom en så kallad policy. Agenten strävar till att maximera sin belöning och måst därför lära sig en optimal policy att interagera med miljön. Vilket kan beskrivas med figur 1 som visades i ett tidigare avsnitt om förstärkt inlärning.

Stolp-balansering (*eng. Pole-balancing)* är ett simpelt exempel som brukar tas upp på ett förstärkt inlärnings och kontrollproblem där målet är att balansera en stolpe som står på en kärra. Miljön belönar agenten som styr kärran till höger eller vänster enligt hur länge agenten kan hålla stolpen i en upprätt position. Agenten bestraffas, med en negativ belöning när stolpen faller över. [10]

Miljön visualiseras enligt figur 3.



Figur 3 - Stolp-balansering miljö (egen bild)

Markovs-beslutsprocess

Exemplet ovan har några viktiga element, som kan formaliseras enligt en Markov-beslutsprocess *(eng. Markov decision process, MDP),* dessa element är följande:

Läge	Kärran kan befinna sig på en rad av positioner längs x-axeln. Likaså kan stolpen befinna sig på en rad av vinklar.
Handling	Agenten kan flytta kärran åt höger eller vänster.
Övergångs läge	När agenten agerar ändras miljön — Kärran flyttar på sig samt stolpens lutning och hastighet påverkas.
Belöning	Ifall agenten balanserar stolpen väl kommer den belönas positivt. Ifall stolpen trillas kommer agenten belönas negativt.

Matematiskt definieras MDP enligt följande:

$S, en\ uppsättning\ möjliga\ lägen.\ (eng.state)$
$A, en\ finit\ uppsättning\ handlingar.$ (eng. action)
$P(r, s'|s, a), överföringsläge.$
$R, en\ belönings\ funktion.\ (eng.reward)$

MDP ger möjlighet till ett matematiskt ramverk för att modulera beslutsfattning i en given miljö.
Då en agent utför en handling i ett MDP ramverk utformas en episod. En episod består av en serie av tupler av *läge*, *handling* och *belöning*. En episod håller på tills miljön kommer till ett slutligt läge. Slutligt läge i brädspel är när man utsätt en vinnare eller ingen spelare kan göra någon laglig handling längre, i vårt exempel ovan är ett slutligt läge när stolpen vidrör marken.

Följande ekvation beskriver variablerna i en episod,

$(s_0, a_0, r_0), (s_1, a_1, r_1), \dots (s_n, a_n, r_n)$.

I stolp-balanserings exemplet kommer läget *S* bestå av en tupel på kärrans position och stolpens vinkel,

$(x_{kärra}, \theta_{stolpe})$.

Policy

Med hjälp av MDP söks den optimala policyn för agenten. Policyn avgör hur agenten agerar i ett visst läge. Formellt kan policyn beskrivas som e funktion π vilket väljer handlingen α som agenten gör i läge *s*.

Målet med MDP är att hitta en policy som maximerar det förväntade resultaten i framtiden, flera handlingar framöver. Policyn kan beskrivas med följande ekvation.

$$\max_{\pi} E[R_0 + R_1 + \cdots R_t]\pi]$$

Där *R* representerar förväntat resultat av varje episod. Till näst definieras vad exakt förväntat resultat betyder.

Förväntat resultat

Förväntat resultat är det som förväntas fås i belöning i slutet av en episod. Då handling väljs kan inte bara den ögonblickliga belöningen beaktas utan ett förväntat resultat måste räknas ut. Ibland ger den optimala handlingen en ögonblicklig negativ belöning men den långsiktiga belöningen är den bästa från ett givet läge.

Därför strävar man till att agenten optimerar det förväntade resultatet. Agenten måst överväga sekvensen av handlingarna så att den kan optimera det förväntade värdet. Vidare, med jämlik beaktning vid varje tidsenhet kommer agenten att snart bara beakta en belöning långt i framtiden. Detta leder till en policy där agenten inte känner till nödvändigheterna att fullfölja sin belöning inom relativ tid. Därför implementeras en strategi där förväntade resultatet betyder något mindre till agenten i jämförelse med en ögonblicklig belöning. Denna strategi kallas *rabatterat förväntat resultat*.

Rabatterat förväntat resultat

Genom att skala på belöningen med en faktor γ, upphöjt till den nuvarande tidsenheten kan ett rabatterat förväntat resultat implementeras. Detta betyder att agenten bestraffas ifall många handlingar görs innan den får en positiv belöning.

Rabatterat förväntat resultat påverkar agenten så att den föredrar en omedelbar positiv belöning hellre än en större belöning i framtiden. Vilket är en fördel för att lära agenten en bra policy. Belöningen kan beskrivas med följande ekvation.

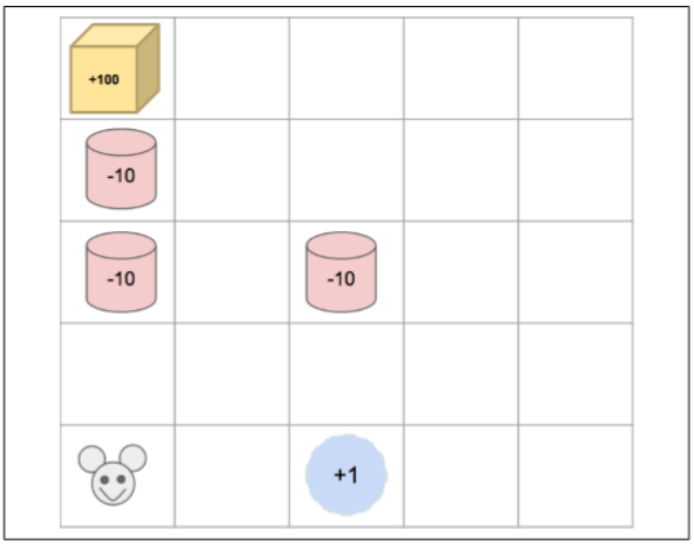
$$R_t = \sum_{k=0}^{T} \gamma^t \, r_{t+k+1}$$

Faktorn γ representerar den nivå på rabatt per tidsenhet som strävas till och kan vara mellan 0 och 1. Ifall γ ligger närmare 1 kommer rabatten att vara liten och ifall den är närmare 0 betyder det att rabatten kommer vara högre. Vanligen kommer γ ligga mellan 0.99 och 0.97. [11]

Utforska eller utnyttja

I grund och botten är förstärkt maskininlärning en form av ”försök och misstag” där agenten lär sig av sina misstag och med tillräckligt många försök lär sig en optimal policy. I en miljö där en agent undviker att göra misstag har detta visats vara mycket problematiskt. Reflektera över följande scenario; En mus placeras i en labyrint enligt figur 4**.** Agenten har som uppgift att kontrollera musen för att maximera musens belöning. Ifall musen når vattnet får den en belöning på +1, då musen når en giftig behållare får den en negativ belöning på -10 och maximala belöningen fås ifall musen når osten där

belöningen är +100. Den optimala policyn gäller då musen hittar en väg till osten. Belöningarna går inte att kombinera, programmet avslutas när musen nått en belöning.



Figur 4 - Labyrint där musen styrs av en agent.

I den första episoden tar musen den vänstra vägen och når gifter och får en negativ belöning på -10. I följande episod undviker musen den vänstra vägen, följer den högra vägen och når vattnet vilket resulterar i en positiv belöning på +1. Efter två episoder ser det ut som att agenten har hittat en bra policy. I efterföljande episoder kommer agenten att välja den policyn den lärt sig för en måttlig positiv belöning. Eftersom agenten utnyttjar en girig strategi — Den väljer alltid modellens bästa handling — kommer den vara fast i en lokal maximum policy.

Genom att avvika från modellens rekommendationer och ta en suboptimal handling för att utforska miljön kan detta undvikas.

Istället för att genast välja den högra vägen för att utnyttja miljön för att nå vatten och den säkra belöningen på +1, kommer agenten välja att låta musen gå mot vänster för att utforska det förrädiska områden i hopp om en mera optimal policy. Med för mycket utforskandet kommer agenten aldrig till en optimal policy och med för lite utforskande kan det resultera i att agenten fastnar i en lokal minimum policy. Balansen mellan utforskning och utnyttjning är avgörande när det gäller att lära agenten en framgångsrik policy.

Exemplet var taget ur boken *Fundamentals of Deep Learning, sidorna 251–252.* [10]

ϵ-Girig

ϵ-Girig *(eng.* ϵ-Greedy) är en strategi för att balansera utforskning—utnyttjning. ϵ-Girig är en simpel strategi som involverar ett val vid varje läge där agent gör en slumpmässig handling istället för den rekommenderade handlingen som ges av policyn. Sannolikheten att agenten väljer en slumpmässig handling är värdet ϵ. [12]

Annealed ϵ-Greedy

Annealed ϵ-Greedy kan översättas som regressiv ϵ-Girig vilket är en metod som utnyttjar ϵ-Girig på ett lite annat sätt. När en modell tränas i förstärkt maskininlärning strävar man till att utforska mera i början eftersom modellen inte känner till miljön. Senare, när modellen skapat en god policy och utforskat största delen av miljön försöker man få agenten att tro på sig själv för att vidare utveckla den optimala policyn. Genom att regressivt utnyttja ϵ värdet efter varje episod kan vi åstadkomma detta. En typisk inställning för regressiv ϵ-Girig är att låta den gå från 0.99 till 0.1 över 10 000 episoder. [12] [10]

Policy inlärning versus Värdes inlärning

Vad betyder inlärning och hur går man till väga för att lära en agent att maximera sin belöning? Inom förstärkt maskininlärning finns det generellt två metoder till detta. Policy inlärning och värdes inlärning. I policy inlärning lär sig agenten direkt policyn för att maximera sin belöning. I värdes inlärning lär agenten sig ett värde för varje handling från varje läge.

Policy inlärning

Vanligtvis i övervakad inlärning, där maskininlärning används för att rubricera sampel, används en så kallad stokastiskt fallande lutning *(eng. stochastic gradient descent, SGD)* för att uppdatera parametrarna för att minimera förlusten vid beräkning av resultaten från neurala nätverken för att få den rätta etiketten för ett sampel. Optimeringen ser ut som följande.

$$arg \min_{\theta} \Sigma_i \log p(y_i|x_i;\theta)$$

Inom förstärkt maskininlärning finns inga etiketter utan endast belöningssignaler. Neuronnäten som används befinner sig alltså i regressionslägen. Således går det att används SGD för att optimera belöningsmängden genom att använda algoritmer som hör till kategorin *policy fallande (eng. policy gradients).* [13] Genom att utnyttja handlingarna utförda av agenten samt belöningen som returneras av dessa handlingar kan agenten uppmuntras att välja goda handlingar som leder till höga belöningar och avhålla dåliga handlingar vilka leder till låga belöningar.

Den algoritm vi optimerar för är följande.

$$arg \min_{\theta} - \Sigma_i R_i \log p(y_i|x_i;\theta)$$

Där y_i representerar den utförda handlingen vid tiden *t* och där R_i är det rabatterade förväntade resultatet.

På detta sätt skalas förlusten beroende på den returnerade belöningen så att ifall agenten utför en handling som returnerar en negativ belöning kommer förlusten vara större. Dessutom ifall agenten är säker på att det är en dålig handlingen kommer förlusten vara ännu större tack vare logaritmen för sannolikheten att agenten väljer den handlingen.

Genom att definiera förlusts funktionen kan SGD tillämpas för att skapa en bra policy och minimera förlusterna.

TD inlärning

Temporär differens förstärkt maskininlärning eller bara TD inlärning (*eng. Temporal difference (TD) reinforced learning, TD-learning)* är ett sätt att få kunskap om hur agenten estimerar sin belöning i ett givet läge. TD inlärningen är av kategorin värdes inlärning och dess algoritmer hör till den så kallade modell fria förstärkta maskininlärningen.

Q-Learning

Q-Learning är en metod som hör till temporär differens inlärning och dess huvudsakliga uppgift är att optimera val av handling för varje enskilt läge. Q-Learning algoritmen är anpassad för inlärning av strategisk sekventiell kontroll vilket motsvarar det som används inom brädspel. Q-Learning algoritmen är alltså den som formar agenten inom förstärkt maskininlärning. Namnet Q-Learning kommer från det att Q är namnet på funktionen som returnerar belöningen vid förstärkt maskininlärning och det går att tänka att Q står för kvalitén av handlingen som utförts i ett givet läge.

Hur algoritmen fungerar

Agenten kan uppfatta en uppsättning lägen *S (eng. State)* från miljön och kan utföra en uppsättning handlingar *A (eng. Action).* Vid varje iteration känner agenten till nuvarande läge *s* och väljer därefter en handling a vilket den utför. Miljön svarar med belöningen *r (eng. Reward)* och fortsätter därefter till ett nytt läge $\delta(s, a)$, där δ är överföringsfunktionen för givet läge s och handling a.

Låt $Q(s, a)$ vara evalueringsfunktionen så att den maximerar det rabatterade kumulativa belöningen som kan fås från nuvarande läge s genom att utföra den första handlingen a. Således, när agenten lär sig Q värden korrelerar det med att den lär sig den optimala policyn för en given handling. Q värdet är alltså belöningen $r(s, a)$ som agenten får genom att utföra handlingen a vid läge s, plus värden (rabatterad av $\gamma = (0..1)$) ifall agenten följer den optimala policyn därefter. [14]

Allt detta kan summeras med hjälp av följande ekvation ekvationen,

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

För att förstå ekvationen ovanför bättre skrivs ekvationen om till en uppdateringsekvation i en viss tid *t*.

$$Q(s_t, a_t) \mathrel{+}= \alpha \left[r_{t+1} \gamma \max_{\alpha} Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Där,

$Q(s_t, a_t)$ $= Nuvarande\ värde$
α $= \text{Inlär}ningshastighet(0 < \alpha < 1)$
r_{t+1} $= Dirrekt\ belöning$
γ $= Rabattfaktor,$

$högt\ \gamma\ värde\ betyder\ att\ en\ belöning\ av\ en\ handling\ efter$
$många\ drag\ väger\ lika\ mycker\ för\ agenten\ som\ en\ handling\ efter\ få\ drag.$

Hur belöningen tilldelas

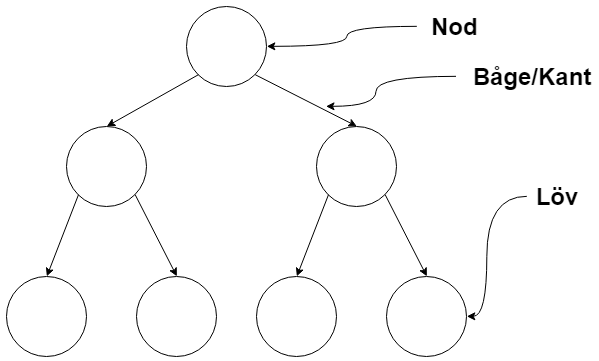
Med belöningen som tilldelas agenten, efter att den har utfört en handling, strävar man till att nå agentens mål. Som tidigare diskuterats tilldelas agenten en positiv belöning ifall handlingen gör att agenten närmar sig sitt mål och en negativ belöning (bestraffning) tilldelas ifall agentens handling tar den längre från sitt mål. Belöningen är ofta noll ifall handlingen inte led någonvart, en så kallad tom handling.

Tyvärr är det i brädspel mycket svårt om inte omöjligt att ge belöning för varje enskild handling innan sista handlingen är gjord och en vinnare är utsedd, för att definiera ifall handlingarna varit goda. Det är först säkert då sista handlingen är gjord så att agenten förlorar eller vinner ifall handlingarna varit godartade och på basen av det ge en positiv respektive negativ belöning. Vid ett oavgjort spel belönas handlingen med noll (ingen belöning) eftersom vi inte kan vara säkra på ifall det var en bra handling eller inte.

Sökträd inom brädspel

Introduktion

Innan en djupare genomgång i hur agenten avgör vilka handlingar som den ska utföra för att sannolikheten för det bästa resultat ska uppnås med hjälp av sökträdsalgoritmer måste det klargöras vad ett sökträd är och dess komponenter. Sökträd är en riktad graf som används inom grafteori för att beskriva samband mellan punkten, så kallade noder. Varje nod sammankopplas genom en båge, också känd som kanter, för att visa samhörighet. Figur 5 beskriver hur sökträd ritas upp samt dess komponenter. Noder utan utgående bågar kallas löv.



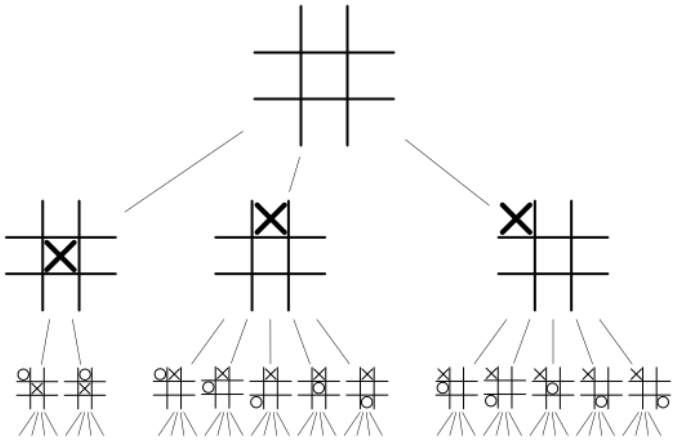
Figur 5 - Sökträds struktur.

Spelträd

Inom maskininlärning används så kallade spelträd. Spelträd är en typ av sökträd, alltså en riktad graf, där noderna representerar spellägen och bågarna representerar handlingar. Varje nod, spelläge, har en eller flera utgående bågar som representerar varje handling som är tillgänglig från det spelläge. Ifall en nod inte har en utgående båge, ett löv, betyder det att spelläget är ett avslutande läge eftersom inga handlingar kan göras från det spelläget och spelomgången kan därför inte fortsättas. I princip kan ett spel avslutas med nästan vilken belöning som helst men för ett standardbrädspel är belöningen begränsad till *vinst* eller *förlust* (ibland också *oavgjort*). Dessa löv värderas vilket möjliggör att agenten kan belönas och lära sig av sina handlingar, vilka handlingar leder till en positiv respektive negativ belöning.

Ett komplett spelträd är ett träd där alla möjliga slutlägen är representerade av löv.

figur 6 beskriver en del av spelträdet för spelet tre-i-rad (*eng. Tic-Tac-Toe)*. Tre-i-rad spelas i tur och ordning av två spelare där målet är att först få tre av markörerna X respektive O i rad inne i en 3x3 rutors spelplan.



Figur 6 - Ofullständig spelträds representation av tre-i-rad (wikipedia)

Ett komplett spelträd av tre-i-rad består av 255 168 ($\approx 2{,}5 * 10^5$) löv, avslutande lägen. I jämförelse med schack som enligt studier gjorda av Claude Shannon redan på 1950-talet har en uppskattad komplexitet på 10^{123}. [15] Genom om att skapa ett komplett träd är det möjligt att beräkna den optimala handlingen från ett givet läge. Men på grund av att komplexiteten för många brädspel överskrider

beräkningskapaciteten och den rimliga tiden det tar för en dator att räkna ut den optimala handlingen kan inte fullständiga spelträd användas utan andra lösningen måste hittas. Inom djup maskininlärning utnyttjar man det att agenten interagerar med miljön och kan själv skapa en uppfattning om en handling leder till ett positivt eller negativt resultat, med lite hjälp förstås. Till näst kommer sökträds algoritmen Monte Carlo sökträdsalgoritmen som används för att hjälpa agenten göra rätt beslut ifall handlingarna inte är tidigare kända för agenten.

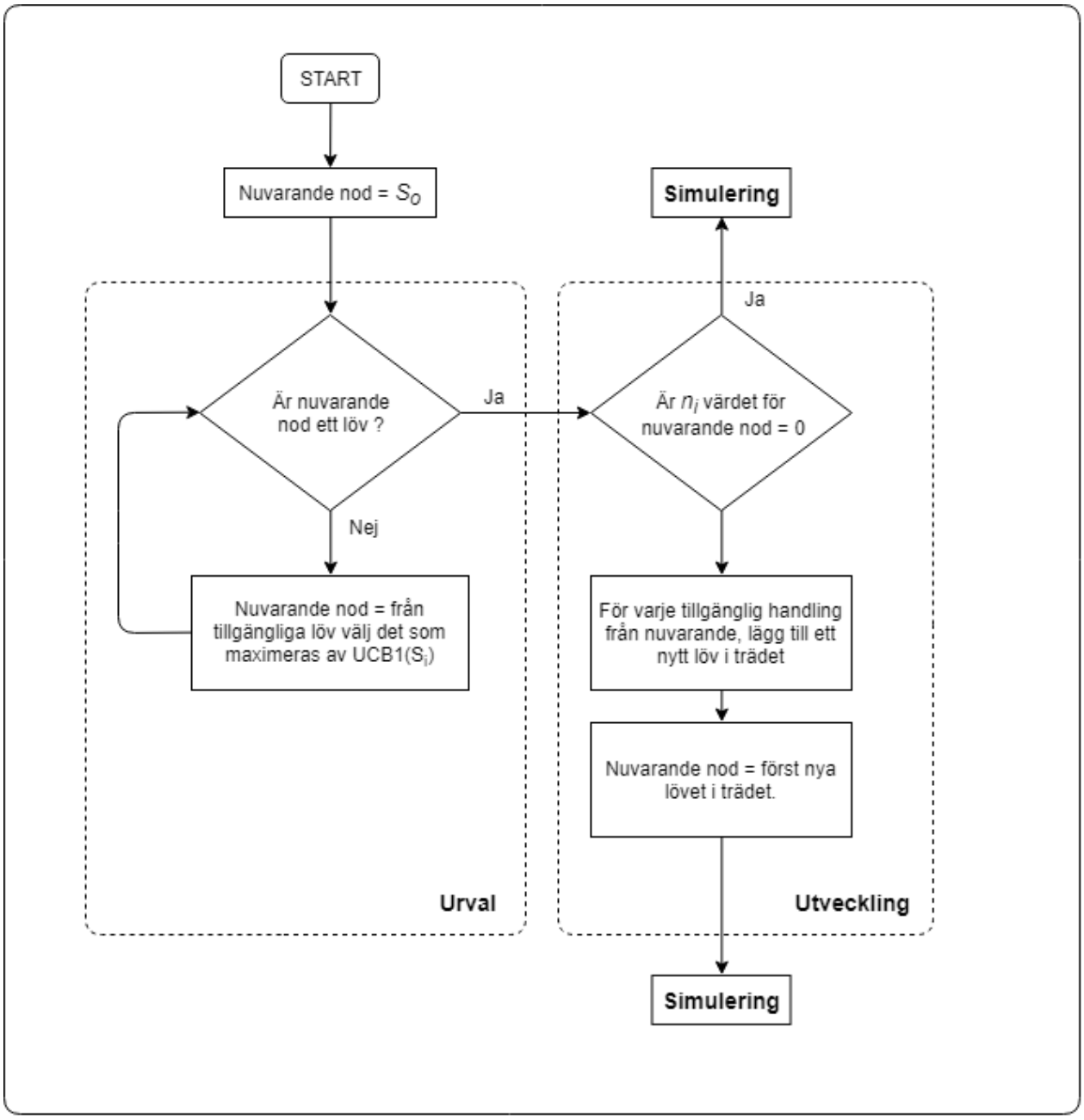
Monte Carlo sökträds algoritmen

En väsentlig del av djup förstärkt maskininlärning inom brädspel är sökträds algoritmer. En av de populärare är Monte Carlos sökträds algoritmen, förkortas MCTS *(eng. Monte Carlo Tree Search)*. MCTS är ett heuristiskt söknings algoritm som används inom besluts bearbetning. [16]

Grund principerna med MCTS simuleringen är att spela en spelomgång med slumpmässigt valda handlingar vilket leder till ett slutläge av ett spel. Från det här läge fås ett slutvärde (belöning eller poäng) som räknas ut och återkopplas till alla tidigare lägen under denna spelomgång. Från ett slumpmässigt spelat spel fås mycket lite information men med en massa spelomgångarna och genom att utföra återkopplingen med given slutpoäng kan en estimering av ett medelvärde för varje läge av varje simulerade spelomgång som besökt ett visst läge vilket gör en stabil strategi för vilka handlingar leder till ett bra spel. Simuleringarna används för att bygga upp ett spelträd där varje nod representerar ett läge och varje kant representerar en handling. [17] [18] [19]

MCTS delas in i 4 steg. *Urval, Utveckling, Simulering* och *Återkoppling* (eng. *Selection, Expansion, Simulation, Backpropagation)*. Varje del av MCTS kan visualiserar med hjälp av ett flödesdiagram enligt figur 7.

Varje nod håller reda på antal gånger den blivit besökt samt total potentiell belöning.



Figur 7 - Flödesdiagram av MCTS

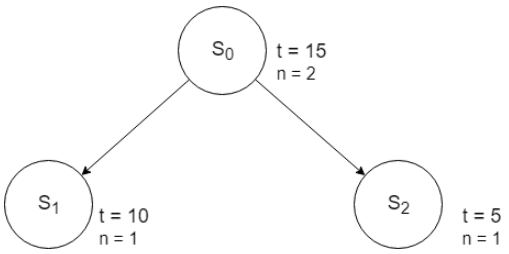
1. Urval

Urval, då agenten gör valet av nod, använder sig av UCB1 algoritmen. UCB står för *Upper Confidence Bound* vilket på svenska betyder övre konfidens gränsen.

$$UCB1(S_i) = \overline{v_i} + C\sqrt{\frac{ln(N)}{n_i}}$$

C är en konstant för att balansera ut utnyttjande av tidigare iterationer, medelvärdet för nodens totala poäng ($\overline{v_i}$), och utforsknings värdet, $\sqrt{\frac{ln(N)}{n_i}}$.

UCB1 algoritmen används på varje tillgänglig nod från den nuvarande noden. Den nod som returnerar det högsta värdet väljs till nuvarande nod.



Figur 8 - exempel på sökträd

Figur 8 visar hur ett mycket simpelt träd kan se ut där det finns två handlingar att utföra. Genom att utföra UCB1 på respektive noder i figur 8 fås följande.

$$UCB1(S_1) = 10 + 2\sqrt{\frac{ln(2)}{1}} \approx 11.67$$

$$UCB1(S_2) = 5 + 2\sqrt{\frac{ln(2)}{1}} \approx 6.67$$

Genom att räkna ut värden för S_1 och S_2 kommer nu agenten välja S_1eftersom UCB1 returnerar ett högre värde för den noden.

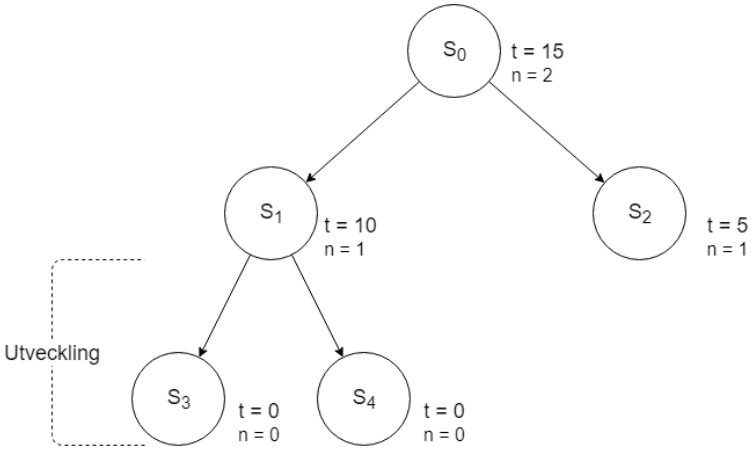
Den nya nuvarande noden är alltså S_1 och MCTS fortsätter till nästa steg, utveckling (eftersom noden tidigare blivit besökt).

Generellt väljs noder med högsta värdet men noder som inte blivit besökta lika ofta väger mera. [18] [20]

2. Utveckling

Ifall UCB1 algoritmen gör så att agenten väljer *(se Urval)* en nod som tidigare har besökts kommer sökträdet att utvecklas, annars utförs simulering. Utvecklingen kan visualiseras med figur 9, på avseende av förändring från figur 8, på följande sätt,

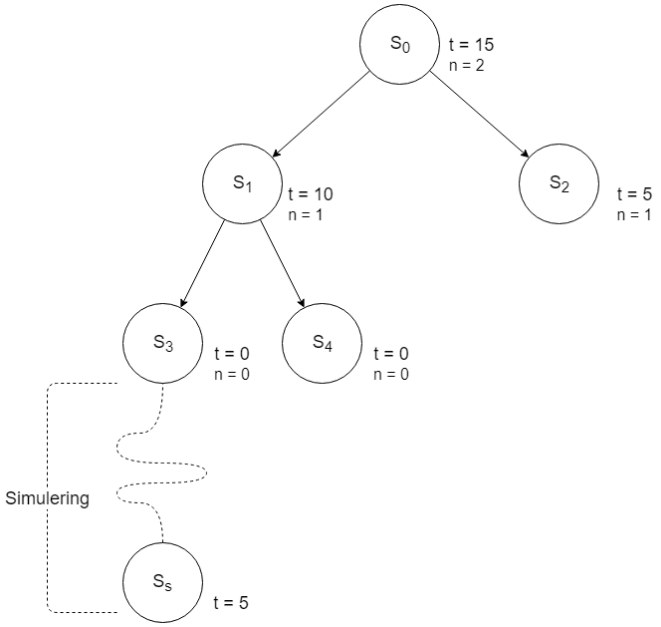
- För varje tillgänglig handling från nuvarande nod (läge), lägg till ett nytt löv (läge) i trädet.
- Välj nuvarande nod till det först i ordningen av dom nya löven.
- Utför simulering



Figur 9 - MCTS utvecklingssteg

3. Simulering

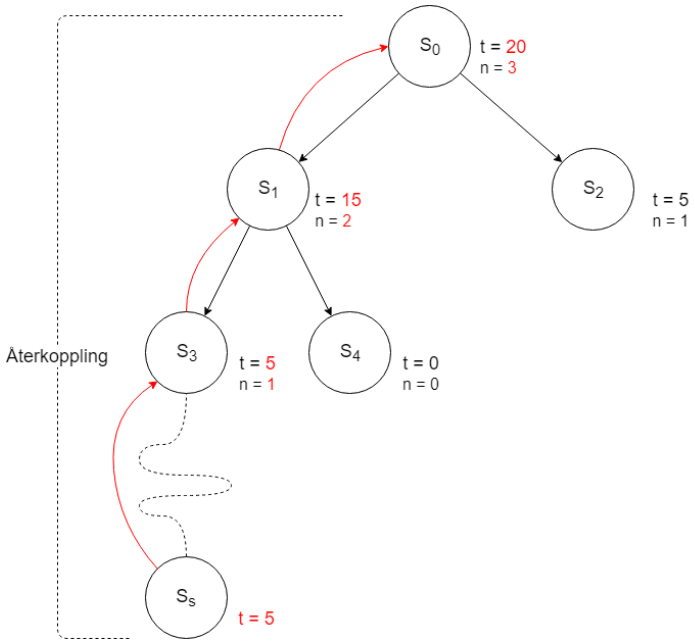
Simulering är en enkel process; agenten väljer en mer eller mindre slumpmässig handling från alla tillgängliga handlingar (kanter) från nuvarande läge (nod). Simuleringen utförs tills nuvarande nod (läge) är ett löv (slutläge). Då simuleringen når ett slutläge noteras nodens värde (resultat eller poäng) efterföljt av återkoppling. Simuleringen visualiseras i figur 10.



Figur 10 - MCTS simulerings steg

4. *Återkoppling*
Uppdatera alla noders totala potentiella belöning som ligger på den stig som simuleringen utfört.

Figur 11 beskriver hur återkopplingen fungerar.



Figur 11 - MCTS återkopplings steg

Efter återkoppling faller simuleringen bort och MCTS börjar om fråga steg 1, med dom uppdaterade värden för besökt nod i denna episod.

Maskininlärning i Brädspel

Ett mål för många utvecklare inom artificiell intelligens är att utveckla en algoritm som lär sig övermänskliga färdigheter i krävande miljöer. Redan under 1980-talet utvecklade Kai-Fu Lee och Sanjoy Mahajan ett program för att spela Othello, BILL, som var överlägsen på sin tid. Den intresserade läsaren hänvisas till [21] [22]. En mångfald av söknings- och samordningstekniker används för att öka söknings bredd. Effektiviteten fås då BILL används sig av förhandsevaluerade tabeller som känner ingen hundra tusentals mönster av spellägen i en konstant tid. Med hjälp av *zero-window alpha-beta sökning*.

Flera år senare, 2017, blev AlphaGo det första programmet som segrade en världsmästare i det kinesiska brädspelet Go. Genom att träna djupa neuronnät, som används för att evaluera positioner och välja handlingar, på handlingar gjorda av experter och med hjälp av förstärkt maskininlärning där agenten själv spelade miljontals spel mot sig själv, så

kallat självspel (*eng. self-play).* Allt detta genom att introducera en förstärkt maskininlärnings algoritmen till endast spelets regler, utan hjälp från mänskliga data eller någon sorts vägledning.

AlphaGo blir sin egen lärare; ett neuronnät tränas att förutsäga egna handlingar av AlphaGo samt vinnaren av spelen som spelas. Neuronnätet förstärker sökningen i sökträdet, vilket resulterar i en bättre kvalité på val av handlingar och ett starkare självspel. [23]

Till näst följer några aspekter om brädspel som måste kännas till innan en implementation av förstärkt maskininlärning till brädspel går att utföra.

Nollsummespel versus icke nollsummespel

Spel där den enes vinst är den andres förlust kallas för nollsummespel (*eng. zero-sum game).* Poker är bra exempel på nollsummespel där en spelares vinst är lika mycket som motspelarnas sammanlagda förlust, därför blir slutresultatet noll. Vilket är ett nollsummespel.

Ett annat exempel är schack, i schack värderas alla pjäser samt brädpositioner. Ifall den ena spelaren vinner en bonde kommer spelarens momentära poäng att ändra med +5 och motståndarens momentära poäng med respektive -5. Alla handlingar i schack leder till nollsummespel.

Motsvarande; När en spelares vinst inte nödvändigtvis motsvarar den andra spelarens förlust kallas spelet då ett icke nollsummespel (*eng. non-zero-sum-game).* Dessa spel är mycket mera komplexa och visar stora utmaningar inom maskininlärning.

Dessa två kategorier är viktiga att känna till och veta skillnaden på.

fullständig versus ofullständig informations spel

fullständig informations spel (*eng. perfect information game*) är det då all information är känd till båda spelarna, ingen gömd information eller då inga slumpmässiga händelser kan inträffa under spelets gång. Till exempel; schack, där all information är tillgänglig,

alla pjäser är utplacerade och spelplanen är synlig till båda spelarna. Tre-i-rad och Go är andra spel som hör till kategorin fullständig informations spel.

Motsvarande, ofullständig informations spel (*eng. incomplete information game*) är då vissa element i spelet inte är kända till spelarna vid alla tidpunkter. Detta är spel där slumpmässiga handlingar kan inträffa. Majoriteten av alla kortspel faller i denna kategori, men också många moderna brädspel har någon art av ofullständig information i sig.

Ofullständig informations spel är inte att blanda samman med engelskans *imperfect information games*, vilket skulle översättas till ofullkomlig informations spel, spel där spelarna inte känner till all information om andra spelare i spelet.

Ett bra exempel på ofullkomlig informations spel är kortspelet poker. Där spelarna känner till sina egna kort samt värden för sina kort men inte känner till motspelarnas kort innan spelomgången är över. [24]

Hur går man till väga för att utveckla maskininlärning i brädspel

För att utveckla djup förstärkt maskininlärning i brädspel måste tidigare nämnda aspekter av brädspel vara bekanta eftersom de påverkar val av algoritmer och tillvägagång.

Ifall det finns en miljö som är väl utvecklad går det att implementera dom algoritmer samt metoder som diskuterats i denna avhandling för att få en agent som i bästa fall påminner om mänskliga motspelare i ett brädspel. Tack vare att beräkningskapasiten vuxit har forskare lyckats implementera förstärkt maskininlärning i brädspel som inte tidigare varit möjliga. I till exempel det kinesiska brädspelet Go har forskare implementerat MCTS för att slå den då varande världsmästaren. [25] Versioner av MCTS och min-max algoritmer har också använts för att skapa schack maskiner som spelare på elitnivå. [16]

Diskussion

För att vidare utveckla denna avhandling borde flera algoritmer inom både värdes inlärning, så som *TD-Löv* och *TD-Riktad* algoritmer, och policy inlärnings algoritmer gås genom. En noggrann genomgång av flera metoder och deras styrkor och svagheter skulle möjliggöra utvecklingen av djup förstärkt maskininlärning i nya moderna brädspel med hjälp av denna avhandling. Eftersom populära algoritmer som använts tidigare för att spela brädspel som till exempel min-max och *min-max med alpha-beta beskärnings* algoritmerna inte hör till kategorin djup förstärkt maskininlärning har dom inte tagits upp i denna avhandling, trots att dom går att använda vid utveckling av AI i brädspel.

Tack vare OpenAI, en ideell förening grundat av Elon Musk och Sam Altman, och deras *OpenAI Gym* som är ett digitalt hjälpmedel vilket möjliggör och effektiverar testningen av olika algoritmer inom förstärk maskininlärning går det snabbt att utveckla samt testa agenternas styrkor och begränsningar. Det är många brädspel och digitala spel som blivit utvecklade i denna miljö för en implementation av en agent som motspelare.

På grund av bristfälliga standardiseringar inom maskininlärning begränsas utvecklingen i dagens läge och under dom närmaste åren förväntas det komma mera bestämmelser och standardiseringarna vilket för utvecklingen fram också inom djup förstärkt maskininlärning. Dessa brister är bland annat en standard som säger hur mycket belöning en agent får för ett givet läge i en given miljö.

Ifall belöningsmängden vore standardiserad kunde studierna om dom olika implementationerna jämföras och vi kunde fokusera på utvecklingen inom dom områden och algoritmer som är bättre lämpade inom området.

Förstärkt maskininlärning saknar också dataset från brädspel som skulle försnabba processen att utveckla agenten för olika brädspel. De dataset som finns är ofta också inte tillräckligt mångsidiga och svåra att använda.

Jämfört med övervakad inlärning, som används för kategorisering, där det finns otaliga dataset med bilder, ljudklipp och finansiella data. [26]

Slutsats

Maskinlärning är en underkategori till artificiell intelligens. Maskininlärning använder sig av olika matematiska algoritmer för att samla information från dataset, informationen används sedan för att dra slutsatser om olärda data eller för användning i andra modeller.

En underkategori av maskininlärning är förstärkt maskininlärning där en agent, en typ av modell, introduceras till en miljö. Agenten belönas med hjälp av definierade värden positivt ifall den utför handlingar som utvecklaren strävar till, och bestraffas ifall den utför handlingar som leder till oönskat resultat.

För optimal inlärning av agenten krävs noggrant utvalda värden för inlärningshastigheten. Dessutom definieras ifall agenten så fort som möjligt ska maximera sin belöning eller hålla tillbaka för att eventuellt hitta en mer optimal lösning. Oftast ges agenten inte all makt utan en slumpmässig handling introduceras för att tvinga agenten utforska nya möjligheter.

Tack vare den ökade beräkningskapaciteten har Monte-Carlo sökträds algoritmen blivit populärare och slagit genom stort tack vare AlphaGo, en agent inom djup förstärkt maskininlärning som vunnit över världsmästaren i Go 2016. Genom att utnyttja MCTS och en väldefinierad miljö var agenten kunde lära sig är det möjligt att skapa en agent som teoretiskt kunde spela vilket spel som helst.

På grund av bristfälliga standardiseringar är det svårt att jämföra lösningar i dagens läge. Förhoppningsvis kommer standardiseringar förekomma inom några år som bland annat definierar hur mycket belöning en viss handling borde ge. Standarder kommer leda till bättre koordinationen mellan forskningsarbeten och detta leda till ännu bättre framgång inom djup förstärkt maskininlärning.

Referenser

[1] S. David, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis och L. Sifre, ”A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” American Association for the Advancement of Science, London, UK, 2018.

[2] J. X. Chen, ”The Evolution of Computing: AlphaGo,” IEEE CS and the AIP, George Mason University, 2016.

[3] J. M. F. Fernandez och T. Mahlmann, ”The Dota 2 Bot Competition,” IEEE, Skåne, Sweden, 2018.

[4] K. Arulkumaran, A. Cully och J. Togelius, ”AlphaStar: An Evolutionary Computation Perspective,” Cornell University, London, UK, 2019.

[5] J. Patterson och A. Gibson, Deep Learning: A Practitioner's Approach (First edition), 1005 Gravenstein Highway North, Sebastopol, CA 9547: O'Reilly, 2017.

[6] I. Goodfellow, Y. Bengio och A. Courville, ”Part I chapter 5: Machine Learning Basics,” i *Deep Learning*, MIT Press, 2016.

[7] P. L. Bartlett och J. Baxer, ”Stochastic Optimization of Controlled Partially Observable,” IEEE, Sydney, Australia, 2000.

[8] M. Mishra och M. Srivastava, ”A View of Artificial Neural Network,” IEEE, Unnao, India, 2014.

[9] Glosser.ca, ”Wikipedia,” 28 2 2013. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/4/46/Colored_neural_network.svg. [Använd 3 4 2019].

[10] N. Buduma och N. Lacascio, Fundamentals of Deep Learning, Sebastopol, CA : O´Reilly, 2017.

[11] J. Foerster, R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel och I. Mordatch, ”Learning with Opponent-Learning Awareness,” i *AAMAS — International Conference on Autonomous Agents and Multiagent Systems*, Stockholm, Sweden, 2018.

[12] X. Lu och B. Van Roy, ”Ensamble Sampling,” i *Neural Information Processing Systems 2017*, California, United States, 2017.

[13] R. S. Sutton, D. McAllester, S. Singh och y. Mansour, ”Policy Gradient Methods for Reinforcement Learning with Function Approximation,” NIPS, 1999.

[14] D. H. Widyantoro och Y. G. Vembrina, ”Learning to Play Tic-Tac-Toe,” International Conference on Electrical Engineering and Informatics, Selangor, Malaysia, 2009.

[15] C. E. Shannon, ”XXII. Programming a Computer for Playing Chess,” *Philosophical Magazine,* vol. Vol. 41, nr Ser.7, No. 314, 1950.

[16] S. B. S. a. P. S. Guillaume Chaslot, ”Monte-Carlo Tree Search: A New Framework for Game AI,” Association for the Advancement of Artificial Intelligence (aaai), The Netherlands, 2008.

[17] M. Świechowski och J. Mańdziuk, ”Self-Adaptation of Playing Strategies,” IEEE, DECEMBER 2014.

[18] P. AUER, N. CESA-BIANCHI och P. FISCHER, ”Finite-time Analysis of the Multiarmed Bandit*,” Kluwer Academic Publishers, The Netherlands, 2002.

[19] T. Anthony, . Z. Tian och . D. Barber, ”Thinking Fast and Slow,” arXiv:1705.08439v4 [cs.AI] , London, 2017.

[20] G. Sylvian och W. Yizao, ”Exploration exploitation in Go: UCT for Monte-Carlo Go,” i *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop,* Canada, 2006.

[21] K.-F. Lee och S. Mahajan, ”BILL: a table-based, knowledge-intensive othello program,” Carnegie Mellon University, Pittsburgh, Pennsylvania, 1986.

[22] K.-F. Lee och S. Mahajan, ”The Development of a World Class Othello Program,” *Science Direct ,* vol. 43, nr 1, pp. 21-36, 1990.

[23] J. S. K. S. I. A. A. H. A. David Silver*, ”Mastering the Game of Go without Human Knowledge,” DeepMind, London, 2017.

[24] J. Levin, ”Games of Incomplete Information,” Stanford University, Stanford, California, 2002.

[25] M. C. Fu, ”AlphaGo and Monte Carlo tree search: The simulation optimization perspective,” i *2016 Winter Simulation Conference (WSC)*, Washington, DC, USA, 2016.

[26] S. Stanford, ”https://towardsdatascience.com/,” https://Medium.com/, 2 10 2018. [Online]. Available: https://towardsdatascience.com/the-50-best-public-datasets-for-machine-learning-d80e9f030279. [Använd 02 04 2019].

[27] X. Zhou, Z. Fei, L. Quan, F. Yuchen och Wei Huang, ”Scientific Figure on ResearchGate,” 3 Fabruary 2019. [Online]. Available: https://www.researchgate.net/figure/Framework-of-reinforcement-learning-Agent-selects-an-action-the-environment-responds-to_fig2_260487058. [Använd 3 Februari 2019].