KTH Computer Science
and Communication

Självlärande Dots & Boxes-spelare

ANDREAS PETTERSSON

Kandidatexamensarbete inom datalogi, grundnivå
Kurs 143X
Handledare: Johan Boye
Examinator: Mads Dam

andrepet@kth.se Vintrosagatan 5 124 73 Bandhagen 073-809 83 86

Sammanfattning

Denna rapport handlar om reinforcement learning-algoritmen Q-Learning. Syftet med arbetet är att implementera en självlärande dots & boxes spelare som efter träning får testspela mot två stycken förprogrammerade spelare. Jag har undersökt hur träningsfasen påverkar hur bra den självlärande spelaren blir genom att variera hur länge den ska få utforska alla möjliga tillstånd spelet kan hamna i. Resultaten är framförda i grafer som analyseras i arbetet. Den självlärda spelaren och Q-Learning-algoritmen analyseras för att ta reda på vad det är den har lärt sig och hur den har lärt sig sina strategier under träningsfasen.

Resultatet jag kom fram till var att den självlärande spelaren behöver spela flera hundra tusen matcher mot sig själv innan den slutar att lära sig. Den självlärande spelaren blev i alla testerna bättre än mina förprogrammerade spelare – den blev till och med så bra att den besegrade mig majoriteten av matcherna jag spelade mot den.

Abstract

This report is about the reinforcement learning-algorithm Q-Learning. The purpose of this work is to implement a self-learning dots & boxes-player which after training will be evaluated against two pre-programmed players. I have investigated how the training period affects how good the self-learning player gets by vary how long it will be exploring all the possible states the game can be in. The results are presented in graphs which are analyzed throughout the work. The self-learning player and the Q-Learning-algorithm are analyzed to find out what it has learned and how it has been taught its strategies during the training period.

The result I came to was that the self-learning player needs to play against itself for several hundred thousand of games before it stops to learn. The self-learning player became in all of the tests better than mine pre-programmed players - it even became so good that it beat me the majority of the games I played against it.

Förord

Den här rapporten skrivs som en del av kandidatexamensarbetet vid CSC på Kungliga Tekniska Högskolan. Arbetet har utförts av Andreas Pettersson, med handledare Johan Boye.

Arbetet har bestått av tre delar: rapportskrivning, programmering av algoritmer och testning. En stor del av tiden gick åt till programmerandet vilket har stått för omkring 80 % av arbetstiden.

Ett stort tack till min handledare Johan Boye och Örjan Ekeberg på NADA [7] för all den hjälp jag har fått under arbetets gång.

Innehåll

Del I

Inledning

Kapitel 1

Introduktion

Ett populärt ämne inom datorvetenskapen är artificiell intelligens, AI. De flesta känner igen begreppet från TV-, och datorspel och det läggs ner mycket tid och pengar på att utveckla AI-agenter som är tillräckligt svåra för att spelare ska få en underhållande utmaning. Problemet med många av dagens AI-agenter är att de har en bestämd strategi de följer efter, vilket betyder att spelare inom tid listar ut deras strategier. Eftersom AI-agenten inte kan ändra på sin strategi kommer den nu att förlora jämnt och ständigt.

En lösning på detta problem skulle kunna vara att låta AI-agenterna få *lära sig* av sina egna misstag så att de kan lära sig handskas med strategier man inte tänkt på eller ens visste fanns. Därför kommer jag att undersöka hur man kan lära en dator att spela brädspelet dots & boxes och se om den blir tillräckligt bra för att slå mig. En liknande undersökning gjordes 2010 av Oskar Arvidsson & Linus Wallgren vid Kungliga Tekniska Högskolan [5].

1.1 Problemformulering

Detta arbete kommer att undersöka hur pass bra man kan lära en dator spela brädspelet dots & boxes genom att använda en reinforcement learning-algoritm vid namn Q-Learning. Denna algoritm kommer att implementeras till en spelare som kommer att tränas upp genom att spela en massa partier mot sig själv. För att validera hur bra den självlärande spelaren blir kommer den att testspela under sin träning mot två förprogrammerade spelare med bestämda strategier. Den första testspelaren spelar helt slumpaktigt medan den andra har en förbestämd strategi den spelar efter.

Kapitel 2

Bakgrund

2.1 Dots & Boxes

Dots & boxes (ungefär Sv. ”kvadrater”) är ett brädspel som kan spelas med papper och penna av två eller flera spelare och publicerades först 1889 av Édouard Lucas [4]. Spelplanen är ett begränsat rutnät av kvadrater, vanligtvis ett 3x3-rutors bräde, och spelarna turas om med att markera en kant, vertikalt eller horisontellt, mellan två intilliggande punkter. När en spelare lyckas markera en kant så att det bildas en eller flera kvadrater (vars storlek är 1x1 kanter) är det samma spelares tur att ta en till kant. Spelet är slut när alla kanter är markerade och vinnaren är den som har lyckats omringa flest kvadrater. Om spelarna lyckas omringa lika många kvadrater förlorar den spelare som började.

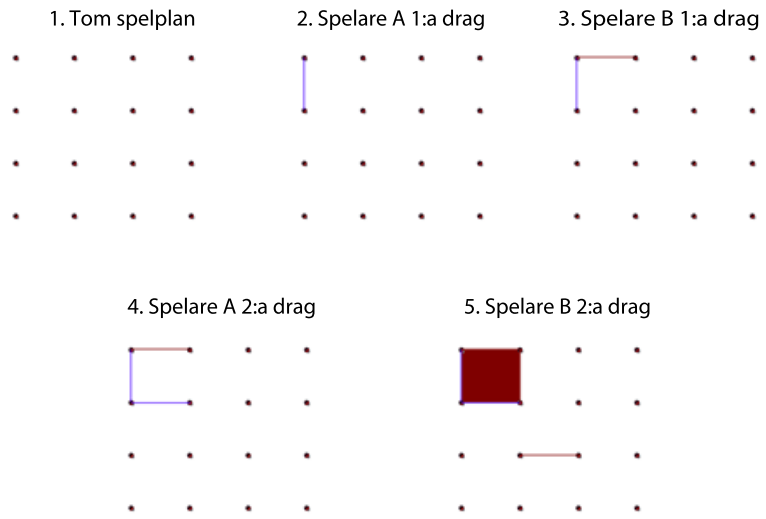
Figur 2.1 illustrerar hur de första dragen i ett parti kan se ut på en 3x3-rutors spelplan.

2.2 Maskininlärning

Maskininlärning är en gren inom artificiell intelligens som bygger på att istället för att man hårdkodar hur ett program ska lösa ett problem så använder man sig av insamlad data för att modifiera eller anpassa sitt programs handlingar för att hantera problemet. Det finns olika typer av inlärningsvarianter, varav detta arbete kommer att beröra en av dessa: belöningsbaserad inlärning, så kallad reinforcement learning.

2.2.1 Reinforcement learning

Reinforcement learning är en belöningsbaserad inlärningsmetod och bygger på att om ens program gör något bra ska den få en belöning och om den gör något dåligt ska den bli bestraffad. Programmet strävar efter att få en så stor sammanlagd belöning som möjligt innan dess uppgift är över, till exempel att den nått slutet på ett spel. För att göra detta måste den utgå från den feedback man ger till den. Fast eftersom



Figur 2.1. Illustration över de första dragen i ett parti mellan spelaren A (som börjar) och spelaren B på en 3x3 kvadraters spelplan. Spelare B:s andra drag (bild 5) leder till att den omringar en kvadrat och måste därför ta en till kant.

man inte talar om hur den ska göra för att få en så hög belöning som möjligt måste den själv lista ut hur den ska göra genom erfarenhet av belöningarna.

Den lärande delen i programmet brukar kallas för agent och för att agenten ska kunna göra någonting måste det finnas en miljö som beskriver alla tillstånd, hur de hänger ihop och dess belöningar man får om man kommer dit. Agenten har som uppgift att, given att den befinner sig i ett tillstånd, välja ett drag som leder till ett giltigt tillstånd att gå till. Den regel för hur agenten väljer sitt val kallas för policy och det finns flera olika sådana. En policy kan till exempel vara att bland de giltiga tillstånden agenten kan gå till alltid välja den med högst belöning. En annan kan vara att med en viss sannolikhet välja ett annat tillstånd än det bästa för att leta efter andra tillstånd som kan leda till en större total belöning. För att en agent ska lära sig att välja de drag som leder till en optimal belöning i framtiden måste den kombinera gammal erfarenhet med utforskning av nya tillstånd. Det man kan göra är att till en början låta agenten, med en ganska hög sannolikhet, välja andra drag än det förväntade bästa för att med tiden minska sannolikheten för slumpvisa drag. När träningen är över låter man agenten välja de drag den anser vara bäst för varje givet tillstånd för att utnyttja all kunskap den har samlat på sig.

För att en agent ska kunna hålla reda på vilken belöning olika tillstånd är värda använder den sig av en värdefunktion. Denna funktion returnerar den förväntade totala belöningen i varje tillstånd. Den kan uttryckas så här:

$$V(s) = E(r_t|s_t = s) \tag{2.1}$$

där högerledet är väntevärdet för den förväntade totala belöningen r_t givet att man befinner sig i tillstånd s. Den metoden jag kommer att använda mig av kallas för Q-Learning och har en liten annorlunda värdefunktion. Mer om den finns att läsa i nästa avsnitt 2.2.2.

2.2.2 Q-Learning

Den reinforcement learning-algoritm som används i detta arbete går under namnet Q-Learning. I denna variant skrivs värdefunktionen som $Q(s, a) = E(r_t|s_t = s, a_t = a)$ där högerledet är väntevärdet för den förväntade totala belöningen r_t givet att man befinner sig i tillstånd s och gör val a. Q-funktionen uppdateras enligt följande formel:

$$Q(s, a) = Q(s, a) + \mu(r + \gamma \max_{a' \in A}(Q(s', a') - Q(s, a))) \tag{2.2}$$

vilken uppdaterar den totala förväntade belöningen i tillstånd s om drag a spelades genom att lägga till belöning från nästa framtida tillstånd s' och dess bästa drag a'. A är alla tänkbara val man kan göra i det framtida tillståndet s' och r är den direkta belöningen man tilldelas för att ha valt a från tillstånd s. r kan ses som en funktion $r(s, a)$ som delar ut olika belöningar beroende på vilket tillstånd s och drag a man gjorde.

μ är inlärningshastigheten som bestämmer hur mycket man ska uppdatera Q-värdet med genom att skala ner värdet som var tänkt att gå till Q-funktionen. Denna parameter kan anta värden mellan 0 till 1 och fungerar så att om $\mu = 1$ är man bara intresserad över framtida värden och bryr sig inte om de gamla värdena från Q-funktionen, och om $\mu = 0$ lär sig agenten ingenting.

γ är en skalningsfaktor som bestämmer hur viktiga de framtida värdena är för Q-funktionen och kan anta värden mellan 0 till 1. Anledningen till denna parameter är för att handskas med osäkerheten kring framtida belöningar som med tiden sprids ut till de andra Q-värdena. En belöning som förväntas komma efter n drag skalas ner med faktorn y^n.

2.3 Termer

Dots & Boxes Traditionellt brädspel som brukar spelas med papper och penna. Eftersom det inte finns något officiellt namn för spelet på svenska kommer det engelska namnet att användas i arbetet. Spelet beskrivs i avsnitt 2.1.

Reinforcement learning Sv. ”Belöningsbaserad inlärning”. En metod för att programmera självlärande system. Den engelska termen är betydligt mer använd och välkänd än den svenska benämningen, därför kommer den engelska termen att användas i detta arbete. Metoden beskrivs i avsnitt 2.2.1.

Q-Learning En typ av reinforcement learning-algoritm. Beskrivs i avsnitt 2.2.2.

Värdefunktion En funktion som en reinforcement learning-algoritm använder för att returnerar den förväntade totala belöningen för varje angivet tillstånd.

Q-funktion Värdefunktionen $Q(s, a)$ som Q-Learning-algoritmen använder sig av.

Q-tabell En tabell som håller reda på alla värden för Q-funktionen.

Q-värde Ett värde i Q-tabellen.

Agent En reinforcement learning spelare.

QL-agent En förkortning för Q-Learning-agent.

Policy Regler för hur en agent väljer ett drag.

Del II

Metod

Kapitel 3

Träning

3.1 Q-Learning-agenten

Q-Learning-agenten som jag kommer att undersöka implementeras med en tabell som håller reda på alla Q-värden för Q-funktionen. Agenten kommer i arbetet att kallas för QL-agent eller bara agent och implementeras på följande vis:

Algorithm 1 Q-Learning-agenten

Sätt $Q(s, a)$ till små slumpade värden för alla tillstånd s och för alla möjliga drag a.
for alla träningsomgångar **do**
  Initiera tillståndet s
  repeat
    Observera tillstånd s
    Slå upp värdena i Q-tabellen som motsvarar alla drag a för tillstånd s
    Välj drag a med hjälp av policy π_s (se avsnitt 3.4)
    Utför drag a
    Observera nya tillståndet s' (efter att motspelaren har gjort sitt drag)
    $Q(s, a) \leftarrow Q(s, a) + \mu(\max_{a' \in A}(Q(s', a') - Q(s, a))$
  until träningsomgången är slut
  Mottag en belöning r
  Observera sluttillståndet s_{slut} och det sista draget a_{slut} man utförde
  Mottag en belöning r
  $Q(s_{slut}, a_{slut}) \leftarrow Q(s_{slut}, a_{slut}) + \mu \cdot r$
end for

Algoritmen för uppdateringen av Q-funktionen är lite annorlunda än formel (2.2). Anledningen till detta är för att QL-agenten bara kommer att få en belöning r i slutet av ett parti. Sedan har jag satt ett fixt värde $\gamma = 1$ eftersom agenten bara bryr sig om framtida belöningar som delas ut i slutet. Av denna anledning kan man ta bort parametern γ helt från formel (2.2) vilket gör den mer lättbegriplig.

Spelplan	Antal kanter	Antal tillstånd	Minnesutrymme (8 bytes per värde)
n x n	$2n(n+1)$	$2n(n+1) \cdot 2^{2n(n+1)}$	$2n(n+1) \cdot 2^{2n(n+1)} \cdot 8$ B
2 x 2	12	49 152	393.2 kB
3 x 3	24	402 653 184	3.2 GB
4 x 4	40	$4.4 \cdot 10^{13}$	351.8 TB

Tabell 3.1. Tabell över hur antal tillstånd till Q-tabellen och minnesutrymmet påverkas av storleken på spelplanen. n x n beskriver spelplanens storlek där n är antalet kvadrater det finns i varje rad eller kolumn. Till exempel är 2x2 en spelplan med totalt 4 kvadrater som tillsammans delar på 12 kanter.

Belöningen r har som syfte att informera agenten om den har gjort bra ifrån sig eller inte. Jag har valt att dela ut belöningen $r = 1$ vid vinst och $r = -1$ vid förlust.

Inlärningshastigheten har jag satt till ett fixt värde $\mu = 0.1$ med hänvisning till [5] som kom fram till att det lönade sig att ha ett lågt värde på inlärningshastigheten.

3.2 Storlek av spelplan

Storleken på spelplanen spelar väldigt stor roll på hur många värden Q-tabellen måste hålla reda på. Tabell 3.1 illustrerar hur spelplanens storlek påverkar antal möjliga tillstånd man måste hålla reda på samt hur mycket minnesutrymme de tar upp. Som tabellen visar växer minnesutrymmet exponentiellt med anseende på hur många kvadrater spelplanen innehåller. Även fast man egentligen bara behöver hålla reda på hälften av tillstånden (de Q-värden som representerar en redan tagen kant behöver inget värde eftersom den ändå aldrig kan bli vald) kvarstår faktumet att minnesutrymmet blir väldigt stort redan vid 3x3-rutor. Av denna anledning har jag valt att använda en spelplan på 2x2-rutor för detta arbete. Detta val låter mig kunna spara undan flera Q-tabeller för olika träningsperioder utan att behöva oroa mig om minnesutrymmet de tar upp.

3.3 Antal träningsomgångar

En viktig del i Q-Learning-algoritmen är antalet träningsomgångar. Om man väljer för få träningsomgångar är chansen stor att agenten inte hinner besöka alla möjliga tillstånd tillräckligt många gånger för att finna någon bra strategi. Efter ett visst antal träningsomgångar slutar agenten att lära sig, förutsatt att man inte ändrar på omgivningen agenten befinner sig i till exempel genom att ändra spelreglerna. Jag har bestämt mig för att låta agenten få spela mot sig själv 1 miljon gånger på en 2x2 kvadraters spelplan. Anledningen till detta val har jag hämtat från [5] som kom fram till att 1 miljon spelade partier är tillräckligt innan QL-agenten slutar att lära sig på en 2x2 spelplan. För att kunna validera hur bra träningen var sparas Q-tabellerna efter var 10 000:e parti.

3.4 Policy

För att välja vilket drag agenten ska göra under sin träning kommer den att använda en slumppolicy π_s. Policyn har som uppdrag att låta agenten få utforska flera möjliga partier som den kanske inte hade upptäckt själv om den alltid hade valt draget med högsta Q-värde. Detta hjälper agenten att lättare finna nya lönsamma strategier så att agenten inte fastnar på en strategi som kanske inte är den optimala. π_s kan skrivas på följande sätt:

Algorithm 2 Policy π_s för Q-Learning-agenten

observera tillstånd s
$i \leftarrow$ antal spelade träningsmatcher
$t \leftarrow$ slumptal mellan 0 till C
if $t \geq i$ **then**
  $a_s \leftarrow$ slumpvald $a \in A$
else
  $a_s \leftarrow a \in A$ så att $Q(s,a) \equiv \max_{a' \in A}(Q(s,a'))$
end if
return a_s

där A är alla giltiga kanter man kan välja från tillstånd s och C är en explorationsparameter med ett fixt valt värde under träningen. Detta värde bestämmer hur länge chansen finns att agenten väljer en slumpvald kant. Ju fler träningsmatcher agenten spelar desto mindre blir chansen för att en slumpvald kant väljs. När antalet träningsmatcher är större än C väljer agenten det drag med högst Q-värde. Jag kommer att använda tre explorationsvärden för att undersöka hur mycket exploration som är lönsammast. De värden jag kommer att undersöka är $C = 150\ 000$, $C = 300\ 000$ och $C = 500\ 000$.

När agenten sedan evalueras mot de två andra spelarna kommer agenten att alltid välja de drag a med högst Q-värde för tillstånd s.

Kapitel 4

Testning

4.1 Valideringsspelare

De förprogrammerade spelarna har bestämda strategier och är till för att validera hur bra QL-agenten blir under sin träning. Strategierna som används av spelarna är strategier som vanliga spelare kan tänkas använda sig av när de spelar dots & boxes.

4.1.1 Slumpspelare

Slumpspelaren har egentligen ingen strategi den går efter utan väljer en giltig kant på ren slump. Detta betyder att den inte vet vilka kanter som den bör ta och undvika för att vinna och kommer med tiden varken bli bättre eller sämre.

4.1.2 Smartspelare

Smartspelarens strategi är att alltid ta den kant som leder till att den fångar in minst en kvadrat. Om en sådan kant inte finns försöker spelare ta en kant som leder till att motståndaren inte kan fånga in en kvadrat under sin tur, med andra ord tar den en kant så att det inte finns ett kvadratområde med tre markerade kanter. Om smartspelaren hamnar i en situation där den inte kan ta en kvadrat eller förhindra att motståndaren tar en kvadrat under sin tur så väljer smartspelaren en kant på ren slump.

4.1.3 Valideringsspelare mot otränad Q-Learning-agent

Innan QL-agenten började sin träning fick den spela 1000 partier mot de båda förprogrammerade spelarna. Matcherna spelades på så sätt att man byte ordning på vem som skulle ta den första kanten efter varje parti och QL-agenten valde alltid de drag med högst Q-värde (även om detta inte spelar någon roll då Q-värdena från början var initierade med små slumpade värden). Resultatet blev att QL-

agenten vann 50 % av matcherna mot slumpspelaren och 8 % av matcherna mot smartspelaren vilket tyder på att smartspelaren är svårare att slå än slumpspelaren.

4.2 Testmetod

QL-agenten har testats mot de två ovanstående spelarna för att validera hur bra träningen har gått. Detta gick till på så sätt att agenten laddades med de sparade Q-tabellerna från träningen och använde dessa för att välja drag. Agenten gjorde så att för varje ny inlärd Q-funktion så spelade den 1000 partier mot de två spelarna där man bytte varannan match vem som började. Under dessa partier använde inte agenten policyn π_s för att välja drag, utan valde alltid det drag med högst Q-värde. Den uppdaterade inte heller Q-värdena med formel (2.2) eftersom man vill validera vad agenten har lärt sig under träningen. Detta betyder att agentens strategi alltid var deterministisk, men eftersom slumpspelaren och smartspelaren inte har deterministiska strategier var detta inget problem för valideringen.

Det fanns ett par parametrar som kunde varieras under träningen vilket påverkade hur bra agenten blev:

- Inlärningshastigenten μ för Q-Learning-algoritmen. Jag valde att använda ett fixt värde $\mu = 0.1$.
- Belöningen r under träningen. Belöningen delades bara ut i slutet av varje parti med värdena $r = 1$ vid vinst och $r = -1$ vid förlust.
- Skalningsfaktorn γ för Q-Learning-algoritmen. Eftersom agenten bara brydde sig om framtida belöningar valde jag ett fixt värde $\gamma = 1$ vilket gjorde att den kunde bortses från formeln (2.2).
- Explorationsparametern C för policy π_s. Jag valde att undersöka tre värden för explorationsparametern: $C = 150\ 000$, $C = 300\ 000$ och $C = 500\ 000$. Dessa värden påverkade hur länge agenten valde drag på måfå istället för de dragen med högst Q-värden under träningen.
- Antal träningsomgångar. Jag valde att låta agenten träna 1 miljon tränings-matcher. Jag ansåg att detta var tillräckligt många matcher innan agenten slutade att lära sig.

Del III

Resultat

Kapitel 5

Resultat

Som det beskrivits i avsnitt 4.2 har man sparat undan agenternas Q-tabell för Q-funktionen var 10 000:e parti. Alla dessa sparade Q-funktioner testas mot slumpspelaren och smartspelaren 1000 gånger. Resultaten av dessa matcher redovisas i grafer med vinstprocenten för QL-agenten.

Jag har valt att använda medelvärdet av tre värden från testningen till en datapunkt i grafen. Denna metod reducerar de skarpa svängningarna i graferna samtidigt som de bibehåller de generella trenderna som finns vilket gör graferna mer lättlästa.

För att plotta graferna använde jag programmeringsspråket *python* version 2.6 med paketet *matplotlib*. På grund av att paketet inte kunde hantera nordiska bokstäver beslöt jag mig för att beskriva x-, och y-axlarna på engelska.

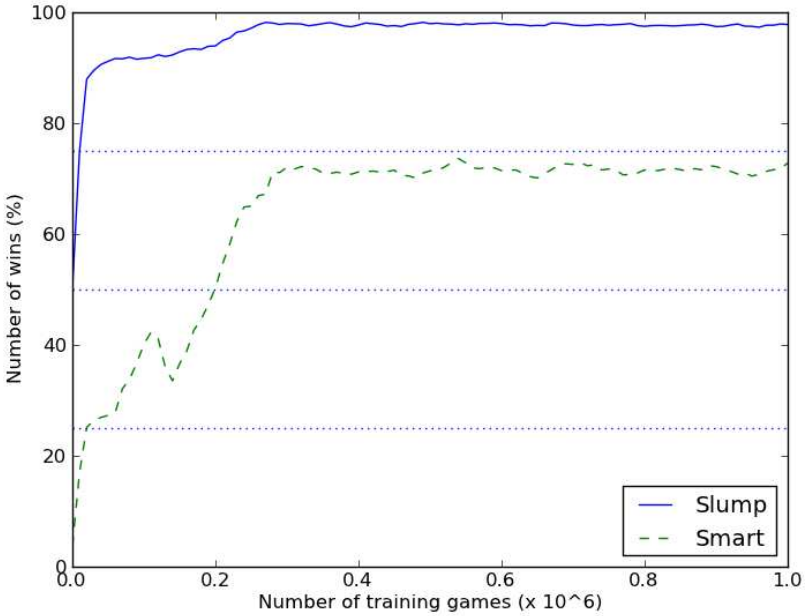
5.1 Validering av Q-Learning-agent med $C = 150\ 000$

Figur 5.1 visar resultatet av valideringen mot slumpspelaren och smartspelaren med $C = 150\ 000$.

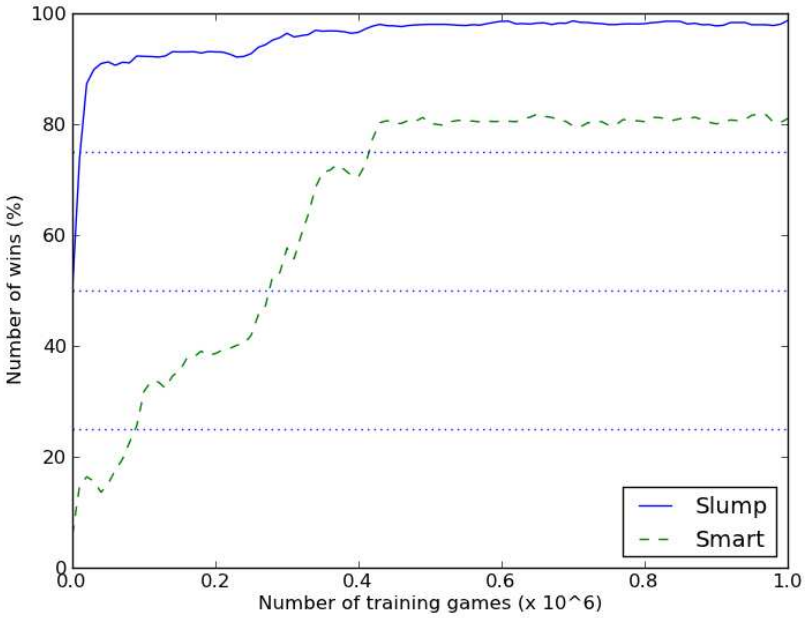
Man kan se att QL-agenten ganska snabbt finner en strategi för att vinna mot slumpspelaren, vilken förbättras fram till omkring 300 000:e träningsmatchen. Efter det vinner QL-agenten i snitt med 98 % av matcherna.

Efter omkring 300 000 träningsmatcher har QL-agenten funnit en strategi som gör så att den vinner omkring 73 % av matcherna mot smartspelaren. Det lilla hoppet bland vinstprocenten som inträffar vid omkring 100 000 träningsmatcher tyder på att QL-agenten fortfarande är i utforskningsstadiet.

Båda spelarna tyder på att QL-agenten har funnit sin optimala strategi efter 300 000 träningsmatcher mot sig själv.



Figur 5.1. Vinstprocent för Q-Learning-agenten med $C = 150\ 000$ efter 1 miljon träningsmatcher.



Figur 5.2. Vinstprocent för Q-Learning-agenten med $C = 300\ 000$ efter 1 miljon träningsmatcher.

5.2 Validering av Q-Learning-agent med $C = 300\ 000$

Figur 5.2 visar resultatet av valideringen mot slumpspelaren och smartspelaren med $C = 300\ 000$.

QL-agenten finner ganska en strategi som slår slumpspelaren. Efter omkring 450 000 träningsmatcher stannar vinstprocenten vid omkring 98 %.

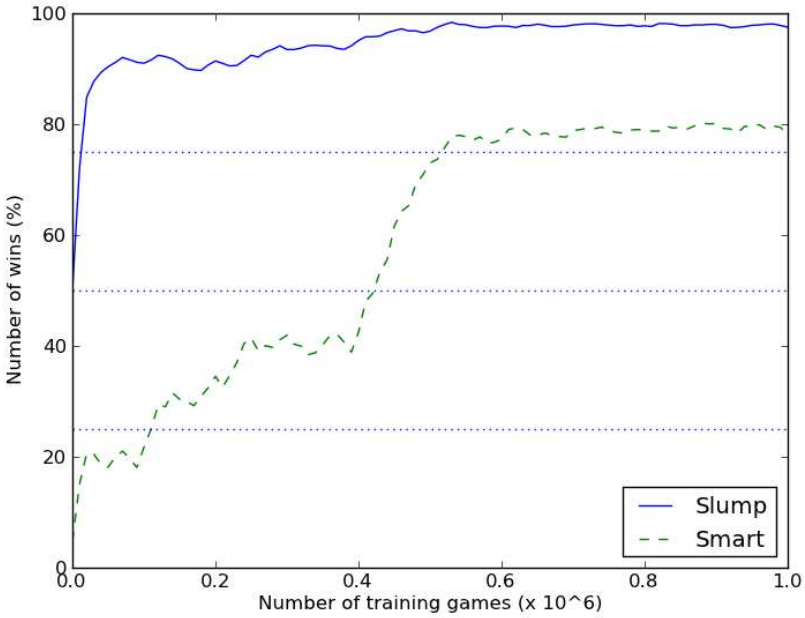
Efter omkring 450 000 träningsmatcher har QL-agenten funnit en strategi som leder till en vinstprocent på omkring 82 % mot smartspelaren. De små dalarna fram till 450 000 träningsmatcher tyder på att agenten fortfarande är i utforskningsstadiet i träningen.

Båda spelarna tyder på att QL-agenten har funnit sin optimala strategi efter 450 000 träningsmatcher mot sig själv.

5.3 Validering av Q-Learning-agent med $C = 500\ 000$

Figur 5.3 visar resultatet av valideringen mot slumpspelaren och smartspelaren med $C = 500\ 000$.

QL-agenten finner ganska en strategi som slår slumpspelaren. Efter omkring 550 000 träningsmatcher stannar vinstprocenten vid omkring 98 %.



Figur 5.3. Vinstprocent för Q-Learning-agenten med $C = 500\ 000$ efter 1 miljon träningsmatcher.

Efter omkring 550 000 träningsmatcher har QL-agenten funnit en strategi som leder till en vinstprocent på omkring 79 % mot smartspelaren. De små svängningarna som inträffar fram till 400 000 träningsmatcher tyder på att agenten är i utforskningsstadiet. Även om den fortsätter att använda policy π_s fram till 500 000 träningsmatcher är det fram till 400 000:e träningen utforskningen är som störst.

Båda spelarna tyder på att QL-agenten har funnit sin optimala strategi efter 550 000 träningsmatcher mot sig själv.

Del IV

Diskussion

Kapitel 6

Diskussion

6.1 Valideringsspelare

6.1.1 Slumpspelare

Min tanke med att ha en slumpspelare var två saker: den första var att visa att en otränad QL-agent är lika bra som en spelare som tar kanter på måfå vilket avsnitt 4.1.3 visar. Den andra tanken var att ha en spelare som snabbt kan indikera att Q-Learning-algoritmen fungerar. Resultaten i kapitel 5 visar att QL-agenten redan efter de första 10 000 träningsmatcherna har blivit tillräckligt bra för att vinna majoriteten av matcherna mot slumpspelaren.

6.1.2 Smartspelare

Min tanke med den här spelaren var att skapa en spelare vars strategi efterliknar den hos en vanlig spelare. Målet med denna spelare var att se om QL-agenten skulle kunna lära sig mer avancerade strategier än de som krävs för att slå slumpspelaren.

Jag måste erkänna att när jag började detta arbete trodde jag att den strategi som jag hade implementerat smartspelaren med var den optimala för en 2x2 kvadraters spelplan. När jag såg resultaten i kapitel 5 för första gången blev jag chockad över att smartspelaren förlorade omkring 80 % av matcherna mot den självlärda spelaren. Hur QL-agenten slog smartspelaren går jag in på i avsnitt 6.3.

6.2 Resultat

Man kan se enligt figur 5.1, 5.2 och 5.3 att QL-agenten har blivit bättre genom att träna mot sig själv. Från att ha haft en vinstprocent på 50 % och 8 % mot slump- och smartspelaren har den efter sin träning en vinstprocent omkring 98 % och 82 % som bäst. QL-agenten verkar vara fullärd efter 300 000 − 550 000 träningsmatcher beroende på hur länge man lät agenten vara i explorationsfasen.

Anledning till varför de olika agenterna slutar att bli bättre efter ett antal träningsmatcher är på grund av att de har funnit en stabil konfiguration av Q-värdena

som leder till att ju mer den tränar mot sig själv desto mer stärks konfigurationen. Detta betyder att agenten kommer alltid att välja samma drag under träningen. Det är bevisat att en QL-agent alltid kommer att konvergera mot en sådan konfiguration [2]. Problemet är den att det kan finnas flera sådana stabila konfigurationer, så kallade lokala maximum. Bara för att den finner ett lokalt maximum betyder det inte att det kan finnas ett annat lokalt maximum som resulterar till en bättre strategi.

Om man jämför graferna från figur 5.1, 5.2 och 5.3 tyder de på att ett värde på $C = 300\ 000$ ger den bästa träningen för QL-agenten. Med $C = 150\ 000$ verkar agenten inte få utforska alla tillstånd tillräckligt många gånger och fastnar i ett lokalt maximum som den tror är den bästa strategin för spelet. Med $C = 500\ 000$ får agenten mer tid att utforska alla tillstånd vilket lätt kan ses i figur 5.3 där matcherna mot smartspelaren ser mera slumpmässiga ut fram till 400 000 träningsmatcher, till skillnad från figur 5.1 och 5.2 som har en jämnare vinstförbättring mot smartspelaren. Att figur 5.3 har en lite lägre topp än figur 5.2 mot smartspelaren tyder på att agenten med $C = 500\ 000$ har fastnat i ett lägre lokalt maximum än agenten med $C = 300\ 000$. Med andra ord har agenten med $C = 300\ 000$ lärt sig den effektivaste strategin jämfört med de andra agenterna.

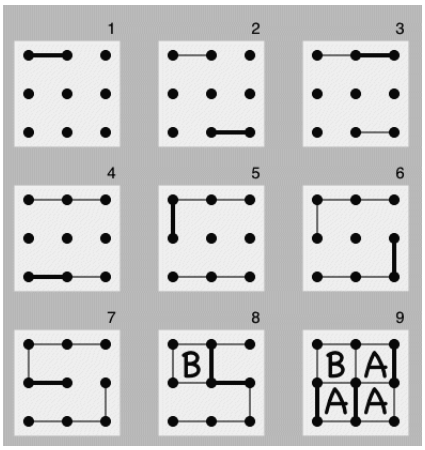
6.3 Analys av Q-Learning-agentens inlärda strategi

Så vad är det egentligen QL-agenten har lärt sig? Hur bär den sig åt för att vinna med mer än 80 % av matcherna mot smartspelaren? Efter att ha analyserat ett antal matcher mellan smartspelaren och QL-agenten med explorationsparametern $C = 300\ 000$ har jag lyckats komma på hur den lyckas med konststycket.

I spelet dots & boxes finns det en speciell strategi som bygger på att man offrar en kvadrat till motståndaren för att sedan tvinga motståndaren att ta en till kant som tillåter en att ta de resterande kvadraterna. Figur 6.1 illustrerar en sådan match mellan spelare A och spelare B där A är den som börjar. B tar de kanter som är spegelvända de kanter A tar i hopp om att de tar två kvadrater var vilket resulterar i vinst till B på grund av lika-regeln. I drag 7 väljer A att offra en kvadrat till B, vilken B accepterar. Men som drag 8 visar måste B nu ta en till kant vilket leder till att A kan ta de resterande 3 kvadraterna i drag 9 och på så sätt vinner matchen.

QL-agenten har genom att tränat mot sig själv lärt sig denna strategi och tvingar smartspelaren att ge de resterande 3 kvadraterna till agenten. På detta sätt vinner QL-agenten varje gång den får börja mot smartspelaren. När smartspelaren är den som börjar spelet försöker QL-agenten offra en kvadrat i hopp om att smartspelaren ska ge bort de resterande 3 kvadraterna. Denna strategi resulterar till vinst omkring 60 % av gångerna, vilket resulterar till en total vinst på omkring 80 %.

Värt att nämna är att man inte förlorar något om motståndaren tackar nej till en kvadrat man offrat. Man kan då ta den offrade kvadraten utan att det påverkar en negativt. Därför vinner QL-agenten nästan alla matcher mot slumpspelaren. De få gånger slumpspelaren vinner är de gånger då den får börja och lyckas slumpa



Figur 6.1. Illustration över en match mellan spelaren A och spelaren B på en 2x2 kvadraters spelplan. Spelare A börjar och tvingar spelare B till att ge bort tre kvadrater genom att offra en kvadrat. Bilden är hämtad från [4]

sina drag så att den vinner på samma sätt som smartspelaren.

6.4 För,- och nackdelar med Q-Learning-algoritmen

Reinforcement learning-algoritmen Q-Learning som användes i detta arbete har både sina fördelar och nackdelar.

Fördelarna med algoritmen är bland annat att den kan anpassa sig efter sin omgivning. Föreställ dig att vi skulle ändra på spelreglerna så att den som tar *minst antal kvadrater* är den som vinner. Då skulle vår tränade agent inte vara något vidare bra, men om den fick träna igen med dessa nya regler skulle den träna bort den gamla strategin och försöka hitta en ny vinnande strategi.

Några av nackdelarna med algoritmen är att den kräver att det är ett ändligt antal tillstånd och drag man kan utföra eftersom QL-agenten måste kunna besöka alla tillstånd för att kunna förbättra sin värdefunktion $Q(s, a)$. Ett annat problem med algoritmen har redan nämnts i avsnitt 3.3 där jag visade hur stort minnesutrymme algoritmen kräver för sin värdefunktion. Detta betyder att om man har ett tillståndsrum med väldigt många tillstånd måste Q-Learning-algoritmen spara undan Q-värdet för *varje* tillstånd. Enligt tabell 3.1 skulle en spelplan på 4x4 kvadrater vara lika med en $4.4 \cdot 10^{13}$ olika tillstånd att hålla reda på. Detta, och att agenten måste besöka varje tillstånd flera gånger, gör att algoritmen skulle bli opraktisk på en sådan stor spelplan.

För att komma ifrån problemet med för stort tillståndsrum kan man försöka att approximera värdefunktionen med ett artificiellt neuralt nätverk. Artificiella neurala nätverk är en slags beräkningsmaskin vars inspiration kommer från hur neuroner fungerar i en biologisk hjärna. Om hur ett artificiellt neuralt nätverk fungerar finns

att läsa i [1]. Fördelarna med att implementera värdefunktionen som ett artificiellt neuralt nätverk är att man bara behöver spara vikterna till nätverket, samt att nätverket har egenskapen att generalisera över data. Detta betyder att den kan approximera hela värdefunktionen även fast den bara har tränats på en delmängd av alla värden. Gerald Tesauro visade att det är möjligt att lära en dator spela backgammon på proffsnivå med hjälp av en reinforcement learning-algoritm vars värdefunktion implementerades som ett sådant nätverk [3].

6.5 Tidigare arbeten

Som nämnt i inledningen har det gjorts en liknande undersökning av Oskar Arvidsson & Linus Wallgren [5]. De undersökte hur inlärningsfaktorn μ och skalningsfaktorn γ påverkar inlärningen hos en självlärande dots & boxes-spelare. De kom fram till att inlärningen blev som effektivast om inlärningsfaktorn μ var satt till ett lågt värde och om skalningsfaktorn γ var satt till ett högt värde. Det finns flera skillnader på hur de utförde sina tester i jämförelse med detta arbete.

Först och främst valde de att utföra träning och validering samtidigt. De hade ingen valideringsfas av QL-agenten utan lät agenten spela en massa matcher mot en annan förprogrammerad spelare samtidigt som den uppdaterade sina Q-värden. De undersökte hur bra agenten lärde sig att spela mot spelaren genom att spara undan antalet vinster per 1000:e match som användes för att bygga upp en graf över antalet vinster. Sedan lät de liknande tränade agenter få spela mot varandra för att se vem som hade lärt sig den bästa strategin och för att se hur lång tid det tog för dem att lära sig varandras strategier. Denna metod är jag kritisk mot eftersom man vill skilja på träningsfas och valideringsfas för att se hur effektiv träningen verkligen var om den sätts på prov mot andra spelare.

I deras arbete hade man även valt andra spelare för att spela mot deras QL-agent. De hade en slumpspelare och en simpelspelare som alltid tog kanter efter ett förbestämt mönster.

Utöver att låta QL-agenten få träna mot sig undersökte de hur bra QL-agenten blev om den fick träna mot en annan spelare. Jag valde att inte utföra sådana tester av den anledningen att även om QL-agenten skulle lära sig en bra strategi mot spelaren skulle den strategin kunna vara hur värdelös som helst mot en annan spelare. Att låta QL-agenten få träna mot sig själv leder till att den finner en mer generell strategi för spelet.

En annan undersökning som inspirerade mig gjordes 2010 av Klas Björkqvist & Johan Wester på Kungliga Tekniska Högskolan [6] som lärde en QL-agent att spela spelet Othello med hjälp av ett artificiellt neuralt nätverk. Det var tänkt att jag till det här arbetet även skulle ha undersöka en QL-agent implementerad med ett artificiellt neuralt nätverk, men på grund av en bugg i koden lyckades jag inte få den att lära sig något.

Kapitel 7

Avslutningsvis

7.1 Felkällor

Det finns ett antal möjliga felkällor med denna undersökning:

- Genom att bara träna och validera varje agent en gång med olika värden på explorationsparametern C finns risken att varje agent har fastnat i ett alltför lågt lokalt maximum. Det kan vara möjligt att under en ny träningsomgång med samma parametervärden få bättre vinstprocent än de jag fick. Att Q-tabellen initialiserades från början med små slumpade värden påverkar också resultatet. Anledningen till varför bara en undersökning utfördes var för att allt för mycket tid las ner på att försöka implementera en QL-agent med ett artificiellt neuralt nätverk.
- Det kan finnas andra värden på explorationsparametern C som skulle kunna resultera i en bättre tränad QL-agent än de jag fick. Eftersom policyn π_s är av probabilistisk natur kan man inte vara helt säker på att samma värde på C ger samma resultat för en annan agent.

7.2 Slutsater

I inledningen var jag nyfiken på hur bra datorn kunde bli på dots & boxes genom att bara skaffa lära sig av erfarenhet. Efter testerna jag har utfört kan jag konstatera att jag är förbryllad över hur pass bra datorn har blivit. Datorn lyckades inte bara med att bli bättre, utan den slog alla mina förprogrammerade spelare med en klar marginal – inklusive min smartspelare som jag först trodde hade den optimala strategin för en liten spelplan.

Nu återstår bara en fråga kvar: har datorn blivit tillräckligt bra för att slå mig i dots & boxes? Efter att ha spelat ett par gånger mot den kan jag konstatera att majoriteten av vinsterna går till datorn.

Litteraturförteckning

[1] Stephen Marsland
Machine Learning, An algorithmic perspective
Chapman & Hall/CRC, 2009.

[2] Sutton, Richard S., & Barto, Andrew G.
Reinforcement Learning: An introduction
MIT PRess, Cambridge, MA.
`http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node65.html`, 1998.

[3] Gerald Tesauro.
Temporal Difference Learning and TD-Gammon
Communications of the ACM March 1995 / Vol. 38, No. 3
`http://www.research.ibm.com/massive/tdl.html`, 1995.

[4] Wikipedia artikel.
Dots and Boxes
`http://en.wikipedia.org/wiki/Dots_and_Boxes`, 2010

[5] Oskar Arvidsson & Linus Wallgren.
Q-Learning for a Simple Board Game
Kandidatexamensarbete, Kungliga Tekniska Högskolan
`www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/arvidsson_oskar_OCH_wallgren_linus_K10047.pdf`, 2010.

[6] Klas Björkqvist & Johan Wester.
Självlärande Othello-spelare. Kan en dator lära sig att spela Othello?
Kandidatexamensarbete, Kungliga Tekniska Högskolan
`www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/bjorkqvist_klas_OCH_wester_johan_K10026.pdf`, 2010.

[7] Diskussion med Örjan Ekeberg
`http://www.nada.kth.se/~orjan/`
senast: 2011-02-04