Självinlärning av fyra-i-rad

JOHAN DALENIUS
och BJÖRN LÖFROTH

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2011

Självinlärning av fyra-i-rad

J O H A N D A L E N I U S
o c h B J Ö R N L Ö F R O T H

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2011
Handledare på CSC var Johan Boye
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/dalenius_johan_OCH_lofroth_bjorn_K11059.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Referat

Vi har implementerat en självinlärande datorspelare för spelet fyra-i-rad. Utgångspunkten är att spelaren endast får veta spelbrädets utseende och möjliga drag, och därför själv måste lära sig de relevanta spelkoncept som behövs för att spela framgångsrikt. Inlärningen sker med belöningsbaserad inlärning med TD(λ) där värdesfunktionen approximeras med att artificiellt neuralt nät. Vi har använt ett tvålagers nät som tränats med BackProp-algoritmen.

Spelaren har under inlärning utvärderats mot en slumpspelare och fyra spelare som använder minimax-algoritmen med en enkel evalueringsfunktion, som söker på olika djup i spelträdet. Dessutom har spelaren utvärderats mot specifika speltillstånd för att undersöka om den lär sig viktiga spelkoncept så som attack och försvar, men även mer specifika begrepp så som rader, kolumner och diagonaler.

Vi lyckas träna upp en datorspelare som slår en slumpmässig spelare i nästan samtliga matcher. Mot den svåraste förprogrammerade taktiken (den som söker djupast) utvecklas agenten ifrån att förlora nästan alla matcher till att vinna ungefär 60% av matcherna.

Abstract

Self-learning of the Connect4 game

We have implemented a self-learning computer player for the board game Connect 4. The idea has been to only provide the player with information about the board state and allowed moves in order to force it to learn all the relevant game concepts on its own. The learning is done with the reinforcement learning method TD(λ) using an artificial neural net as a function approximator for the value function. We have used a two-layer net that has been trained by the BackProp algorithm.

During training, the computer player has been evaluated against a random player and four players who were using the minimax-algorithm with a simple board evaluation function to search the game tree at different depths. The player has also been evaluated against specific board states to determine if it is learning important game concepts such as attack and defense, but also more specific ones such rows, columns and diagonals.

We have been able to train a computer player that manages to beat a random player in nearly all games. The computer player also shows strong developement against the best minimax-player (the one with depth 4). In the beginning of the training our player is beaten nearly every game, but towards the end it manages to win almost 60% of the games.

Förord

Denna rapport är framställd som en del av kandidatexamensarbetet för programmet Civilingenjör Datateknik på Kungliga Tekniska Högskolan.

Arbetet har bestått i att implementera ett program som kan lära sig spela fyra-i-rad, genomföra tester, analysera resultaten och dokumentera projektet i denna rapport.

Vi delade upp implementationen så att Björn Löfroth fokuserade på spellogiken och TD(λ)-algoritmen medan Johan Dalenius skrev koden för träning av det artificiella nätet och för minimax-spelaren. Av nödvändighet innebar uppdelningen av implementationen även en uppdelning för inläsningen av teori.

Analys av resultat och sammanställningen av rapporten har till stor del skett tillsammans.

Vi tackar vår handledare Johan Boye för tips och kommentarer under projektets gång.

Innehåll

Innehåll

Kapitel 1

Introduktion

Frågan om datorer kan vara intelligenta är kontroversiell. Det kanske är ett tag kvar tills datorer kan förstå och producera språk eller konst på samma nivå som människor, men det finns många sysslor som vi kan programmera datorer att utföra intelligent. I detta projekt tänker vi undersöka om ett datorprogram med maskininlärningsmetoder kan lära sig att spela brädspel på ett framgångsrikt sätt med utgångspunkten att endast spelreglerna är kända.

Ett annat alternativ hade varit att även bistå datorspelaren med detaljerad information om spelbrädets utseende, där en tolkning redan är gjord av vilka koncept som är relevanta för spelet. Detta skulle kunna underlätta inlärningen, men vi tycker att det är mer intressant att se om inlärningsmetoden kan lära sig dessa koncept självständigt.

Vi har valt att fokusera på spelet fyra-i-rad, som vi bedömer inte är för komplext för ett program att lära sig, men inte heller för simpelt för att kunna utforskas på ett intressant sätt. Ämnet är intressant eftersom det utforskar artificiell intelligens på ett mätbart sätt.

1.1 Problemformulering

Målet med projektet är att undersöka hur väl en dator kan lära sig att spela fyra-i-rad då den tränas upp mot sig själv med hjälp av belöningsbaserad inlärning och artificiella neurala nät. Inlärningsmiljön ska endast bidra med information om spelbrädets utseende, tillåtna drag och vilken spelare som vann och förlorade.

Utvärdering av datorspelaren kommer ske genom att dels låta den spela mot två olika typer av förprogrammerade taktiker, men även genom att undersöka vilket drag spelaren tar i specifika spelplanskonfigurationer för att om möjligt detektera om den kan lära sig relevanta koncept.

Den ena förprogrammerade taktiken är en spelare som väljer sina drag slumpmässigt. Den andra taktiken går ut på att betrakta alla möjliga drag och motdrag ett visst antal steg framåt i spelet för att se vilket drag som gör det mest sannolikt att vinna. De spelplanskonfigurationer som kommer att testas ska bestå av

attackmöjligheter och försvarsmöjligheter utspridda i olika delar av spelplanen.

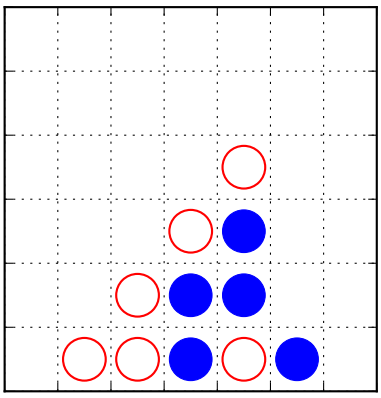
Kapitel 2

Bakgrund

2.1 Fyra-i-rad

Spelet fyra-i-rad är ett spel för två spelare på en spelplan som har 7 kolumner och 6 rader. Varje spelare har varsin färg: röd eller blå. Ett drag är att placera en bricka i någon kolumn. Brickan faller nedåt och hamnar på den första lediga positionen nedifrån. Den spelare som först får fyra brickor i rad antingen horisontellt, vertikalt eller diagonalt vinner spelet. Det finns totalt 69 sådana olika vinnande rader. Om spelplanen fylls utan att någon spelare får fyra-i-rad så blir det oavgjort.

Spelet marknadsfördes 1974 under namnet *Connect Four* av Milton Bradley.



Figur 2.1. Ett vinnande tillstånd i fyra-i-rad

Ett grundläggande koncept är att försöka placera ut brickor som kan vara med i så många vinnande rader som möjligt. Detta innebär att rutorna i mitten av spelplanen är viktiga.

Fyra-i-rad löstes matematiskt 1988 av Victor Allis och James D. Allen oberoende av varandra. Detta innebär att det finns matematiskt bevisade regler för hur en omgång ska spelas optimalt. Om spelaren som börjar lägger sin första bricka i den mittersta kolumnen och sedan följer den optimala strategin så är vinst garanterad. Om brickan istället läggs i någon annan kolumn förutom de yttersta kan spelare två tvinga fram oavgjort. Om brickan läggs i någon av de yttersta kolumnerna så kan spelare två tvinga fram en vinst. [6]

2.2 Minimax sökning

En vanlig algoritm för att spela brädspel är den s.k. *minimax*-algoritmen. Den söker bland framtida möjliga drag och motdrag för att hitta sluttillstånd, dvs tillstånd där någon av spelarna vinner, eller då spelet blir oavgjort. Algoritmen antar att motståndaren alltid gör bästa möjliga motdrag, och returnerar utifrån det ett drag som beräknas vara det bästa, ifrån detta utgångstillstånd.

För många brädspel är tillståndsrymden väldigt stor, vilket gör att det tar för lång tid att söka igenom hela spelträdet till alla sluttillstånd. En möjlig åtgärd är då att använda en teknik som heter *alpha-beta pruning* som undviker att söka igenom vissa onödiga delar av spelträdet, genom att ta hänsyn till tidigare funna värden för genomsökta grenar. För att söka ännu snabbare kan man istället för att söka till sluttillstånd endast söka på ett visst djup framåt, och använda en evalueringsfunktion som bedömer hur bra ett tillstånd är, om det inte är ett sluttillstånd. [9]

2.3 Maskininlärning

Maskininlärningsmetoder brukar vanligtvis delas in i två huvudområden: övervakad och oövervakad inlärning. Vid *övervakad inlärning* tränas ett program utifrån en mängd träningsexempel som består av indata och korrekt utdata. Utifrån denna träningsmängd försöker programmet generalisera för att kunna ge rimlig utdata för all tänkbar indata, även sådan som inte setts under inlärningen. Med *oövervakad inlärning* tränas istället programmet endast på exempel som inte är kopplade till något korrekt svar. Inlärningen består då i att gruppera indata baserat på likheter. För en mer ingående översikt om maskininlärning, se Marsland [1].

För brädspel är övervakad inlärning svår att tillämpa eftersom det är svårt att få tag i en stor mängd träningsexempel med korrekt utdata, dvs en mängd spelplanskonfigurationer och vilka de korrekta dragen därifrån bör vara.

2.3.1 Belöningsbaserad inlärning

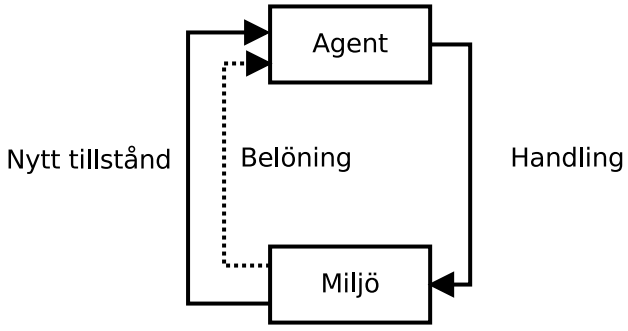
En maskininlärningsmetod som har visat sig passande för att användas för brädspel är s.k. *belöningsbaserad inlärning* [5], framöver förkortat BI. Denna metod sägs vara mitt emellan övervakad och oövervakad inlärning. Å ena sidan använder metoden

inte träningsexempel som exakt säger vilken utdata som förväntas för indata, vilket gör att metoden inte helt kan sägas vara övervakad. Å andra sidan används en s.k. belöningsfunktion, som ger viss återkoppling på hur väl algoritmen presterar men inga förslag på hur den ska förbättra sig, vilket gör att metoden inte kan sägas vara helt oövervakad.

Inlärningsprocessen i BI liknar sättet som vi människor lär oss på. Vi interagerar med miljön omkring oss och tolkar det vi upplever för att dra slutsatser om vilka handlingar vi gör som är bra och vilka som är dåliga, genom att lära från misstag och framgångar.

Grundtanken i BI är att det finns en *agent* som interagerar med *miljön* för att uppnå ett *mål*. När agenten tillåts interagera med miljön får den möjlighet att lära sig hur målet bäst kan uppnås, genom att ta hänsyn till de belöningar och bestraffningar den tar emot. I brädspel ses spelaren som agent, medan miljön består av själva spelet och motståndaren. Målet är att vinna spelet, vilket agenten lär sig genom att spela många omgångar och lära sig av vilka omgångar som ger vinst (belöning) respektive förlust (bestraffning) . [3]

Förloppet i BI sker i en cykel, där agenten först betraktar det tillstånd den befinner sig i för att besluta vilken handling den ska utföra. Miljön bestämmer sedan vilket nytt tillstånd agenten hamnar i och vilken belöning agenten får. Se figur 2.2. För en mer ingående genomgång av belöningsbaserad inlärning än vad som presenteras i denna rapport, se Sutton [3].



Figur 2.2. Förloppet för belöningsbaserad inlärning

Målet är alltid att maximera den totala belöningen som agenten får under sin livstid i miljön. Låt belöningen efter handling nummer t vara r_t, och tillståndet agenten då kommer till vara s_t. Då kan den totala återstående belöningen, R_t ifrån ett tillstånd s_t beskrivas på följande sätt:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... = \sum_{k=0}^{\infty} r_{t+k+1} \tag{2.1}$$

Vi kan då definiera en värdesfunktion, V, för alla tillstånd s som agenten kan befinna sig i, där $V(s)$ är den förväntade totala framtida belöningen då agenten

befinner sig i tillstånd s [3]:

$$V(s_t) \quad = \quad E(R_t) = E(\sum_{k=0}^{\infty} r_{t+k+1}) \tag{2.2}$$

För att justera hur högt agenten ska värdera en belöning som ligger långt fram i tiden kan värdesfunktionen definieras om med en parameter γ ($0 \leq \gamma \leq 1$) som viktar framtida belöningar:

$$R_t \quad = \quad r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.3}$$

$$V(s_t) \quad = \quad E(R_t) = E(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}) \tag{2.4}$$

Då γ sätts till 0 innebär det att agenten enbart betraktar möjliga belöningar för nästa handling. Ju högre γ sätts, desto mer långsiktigt planerar agenten.

En *policy* är något som definierar agentens beteende. För ett tillstånd avgör policyn vilken handling som ska utföras. Målet med agentens inlärning är hitta en policy som maximerar den totala belöningen. Om värdesfunktionen är helt korrekt är en optimal policy att alltid utföra den handling som leder till det nästa tillstånd s_{t+1} som har högst $V(s_{t+1})$. Därför kan målet med inlärningen ses som att hitta en bra värdesfunktion.

Ett stort problem är att belöningen ofta är fördröjd, dvs att agenten inte enkelt kan avgöra vilken handling som bidrog mest till en mottagen belöning. Detta brukar benämnas som *Temporal Credit Assignment*-problemet. [1] Detta är ett vanligt scenario för brädspel, eftersom belöningen endast ges i slutet av spelomgångarna.

Temporal difference

En klass av metoder som löser Temporal Credit Assignment-problemet är *Temporal Difference* (TD) inlärningsmetoder. Grundidén för TD-metoder är att inlärningen baseras på skillnaden mellan två successivt uppskattade värden på den totala framtida belöningen. För dessa metoder blir målet med inlärningen att agentens uppskattning för nuvarande tillstånd ska passa väl med uppskattningen för nästa tillstånd i sekvensen av besökta tillstånd. Nedan visas en härledning för hur värdesfunktionen i ett tillstånd förhåller sig till värdesfunktionen i nästa tillstånd [3]:

$$V(s_t) = E(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}) \tag{2.5}$$

$$V(s_t) = r_{t+1} + E(\sum_{k=1}^{\infty} \gamma^k r_{t+k+1}) \tag{2.6}$$

$$V(s_t) = r_{t+1} + E(\sum_{k=0}^{\infty} \gamma^{k+1} r_{(t+1)+k+1}) \tag{2.7}$$

$$V(s_t) = r_{t+1} + \gamma E(\sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k+1}) \tag{2.8}$$

$$V(s_t) = r_{t+1} + \gamma V(s_{t+1}) \tag{2.9}$$

Resonemanget ovan visar att uppskattningen i nuvarande tillstånd ska vara lika med uppskattningen i nästa tillstånd samt belöningen som agenten fick när den utförde handlingen som tog den dit. Om de inte överensstämmer har vi upptäckt en oregelbundenhet i våra uppskattningar (agenten blev "förvånad") och storleken på denna bör avgöra hur vi ska justera uppskattningen för nuvarande tillstånd. Detta är grunden för uppdateringsregeln nedan som betraktar differensen mellan uppskattningarna (därav namnet Temporal Difference) [3]:

$$V(s_t) \leftarrow V(s_t) + \eta[r_t + \gamma V(s_{t+1}) - V(s_t)] \tag{2.10}$$

Termen η justerar hur stora uppdateringar som görs. Uppdateringarna bör inte vara för stora eftersom uppskattningarna kan vara felaktiga, men är η för litet kommer inlärningen ta lång tid.

Tillståndsrummet, dvs mängden av alla möjliga tillstånd, kan utforskas på olika sätt. Vid inlärning vill man åstadkomma en kombination av att utforska stora delar av tillståndsrummet, och att besöka tidigare besökta delar flera gånger, för att på så sätt göra deras uppskattningar mer korrekta. Man kan välja bland flera olika policies att använda under inlärningsfasen. Tre vanliga policies är *girig*, ϵ-*girig* och *Soft-max*. [1]

Girig Väljer alltid den handling som ger högst $V(s_{t+1})$-värde. Detta leder under inlärning till snäv utforskning av tillståndsrummet.

ϵ-girig Väljer oftast den handling som ger högst $V(s_{t+1})$-värde, men med en liten sannolikhet (ϵ) väljs en av de andra handlingarna slumpmässigt. Med denna policy kan parametern ϵ justeras för att få en högre eller mindre grad av utforskning under inlärningen.

Soft-max Handlingarna väljs med sannolikhet proportionerlig till deras respektive $V(s_{t+1})$-värde, enligt Soft-max funktionen, som beräknar en normaliserad sannolikhet baserat på alla möjliga nästa tillstånd, s'_{t+1}:

$$P(s_{t+1}) \quad = \quad \frac{\exp(V(s_{t+1})/\tau)}{\sum_{s'_{t+1}} \exp(V(s'_{t+1})/\tau)} \tag{2.11}$$

När τ är stort får alla handlingar liknande sannolikhet, medan skillnaden mellan den handling med högst $V(s_{t+1})$ och de andra blir större med ett lågt värde på τ. Genom att justera värdet på τ kan graden av utforskning under inlärningen ökas och minskas.

TD(λ)

En metod som baseras på Temporal Difference är TD(λ). Skillnaden är att TD(λ) uppdaterar uppskattningarna för alla tidigare besökta tillstånd när en oregelbundenhet upptäcks. TD(λ) har i vissa fall gett bättre resultat än TD [7], och har fungerat väl för brädspel [2].

Tanken är att uppskattningen för det närmsta tidigare besökta tillståndet ska påverkas mycket av den upptäckta oregelbundenheten, medan uppskattningarna för ännu tidigare tillstånd ska justeras i förhållande till hur längesedan de besökts. Detta görs eftersom det är osäkert hur mycket ett tillstånd som besöktes för lång tid sedan bidrog till att man kom till det nuvarande tillståndet.

För varje tidigare tillstånd $s_{prev} \in \{s_0, s_1, ..., s_t\}$ görs uppdateringen av uppskattningen enligt regeln nedan, där s_t är nuvarande tillstånd och s_{t+1} är nästa tillstånd:

$$V(s_{prev}) \quad \leftarrow \quad V(s_{prev}) + \eta[r + \gamma V(s_{t+1}) - V(s_t)](\gamma\lambda)^{t-prev} \tag{2.12}$$

Uttrycket $(\gamma\lambda)^{t-prev}$ kallas för lämplighetsspår. Parametern λ ($0 \leq \lambda \leq 1$) används i TD(λ) för att justera hur betydelsefullt ett tillstånd är som ligger på ett visst avstånd bakåt från nuvarande tillstånd. Med värdet 0 på λ uppdateras endast det nuvarande tillståndet, vilket då gör metoden likvärdig med TD. Med högre värden på λ antas tidigare tillstånd bidra mer till att vi kom till nuvarande tillstånd.

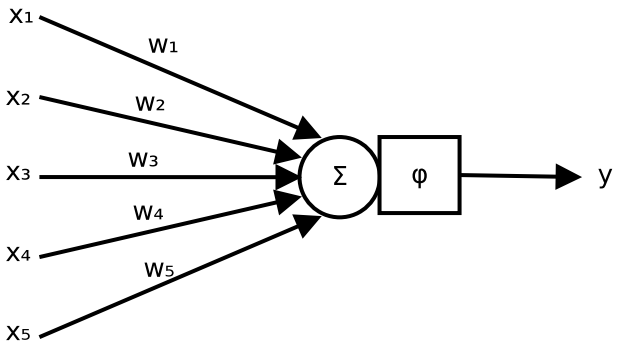
En algoritm för TD(λ) som hämtats från Sutton [3] presenteras i algoritm 1. I koden stegar en agent runt i tillståndsrummet och väljer handlingar enligt sin policy. Varje upptäckt oregelbundenhet blir utgångspunkten för uppdateringar av värdesfunktionen. För varje tidigare tillstånd avgör lämplighetsspåret hur stor uppdatering som ska ske. I pseudokoden är λ, γ och η tidigare diskuterade parametrar. Utöver dessa förekommer π, som är policyn som används och S som är tillståndsrummet. I koden betecknas belöning med r, handling med a, nästa tillstånd med s', nuvarande tillstånd med s, lämplighetsspåret med *etrace* och värdet på en upptäckt oregelbundenhet med δ.

Algorithm 1: TD(λ)
TDLambda(λ, γ, η, S, π)
(1) **foreach** s $\in S$
(2) $V(s) \leftarrow$ slumpTal()
(3) $etrace(s) \leftarrow 0$
(4) **foreach** episod
(5) $s \leftarrow$ startTillstånd()
(6) **while** s inte är sluttillstånd
(7) $a \leftarrow \pi$(s)
(8) $(s', r) \leftarrow$ UtförHandling(s, a)
(9) $\delta \leftarrow r + \gamma V(s') - V(s)$
(10) $etrace(s) \leftarrow etrace(s) + 1$
(11) **foreach** $s \in S$
(12) $V(s) \leftarrow V(s) + \eta\delta \cdot etrace(s)$
(13) $etrace(s) \leftarrow \gamma\lambda \cdot etrace(s)$
(14) $s \leftarrow s'$

2.3.2 Artificiella neurala nät

Den värdesfunktion som används av belöningsbaserade inlärningsmetoder måste vara definierad över hela tillståndstrummet. Detta gör att det i många fall blir orimligt att lagra funktionen som en tabell. För fyra-i-rad finns det över $4.5 \cdot 10^{12}$ giltiga tillstånd [8], vilket gör värdesfunktionen svårlagrad som tabell. *Artificiella neurala nät* (ANN) är en metod inom övervakad maskininlärning som kan användas för att approximera linjära och icke-linjära funktioner. Ett artificiellt nät kräver lite lagringsutrymme i förhållande till hur stor rymd det kan evalueras på. En annan stor fördel med att använda ANN är att ett nät har möjlighet att generalisera för osedd indata. [4] Eftersom tillståndsrymden är så stor för fyra-i-rad kommer en stor del av den inte hinna att utforskas inom rimlig tid, vilket gör generalisering nödvändig för att agenten ska prestera väl.

Idén med ANN är att till viss del simulera hur neuronerna agerar i den mänskliga hjärnan. Grundkomponenten i ANN är den s.k. neuronen, som i abstrakta termer är en enhet som tar andra neuroners utsignaler som indata, och själv skickar vidare signaler till andra neuroner. Det som avgör om en neuron "avfyrar" sin signal är om den sammanlagda mängden insignal överstiger neuronens interna tröskel. Varje koppling till en neuron har en viss styrka, kallad vikt, som avgör hur starkt en insignal ska tolkas. En skiss över en neuron presenteras i figur 2.3. För en mer ingående genomgång av ANN, se Haykin [4].



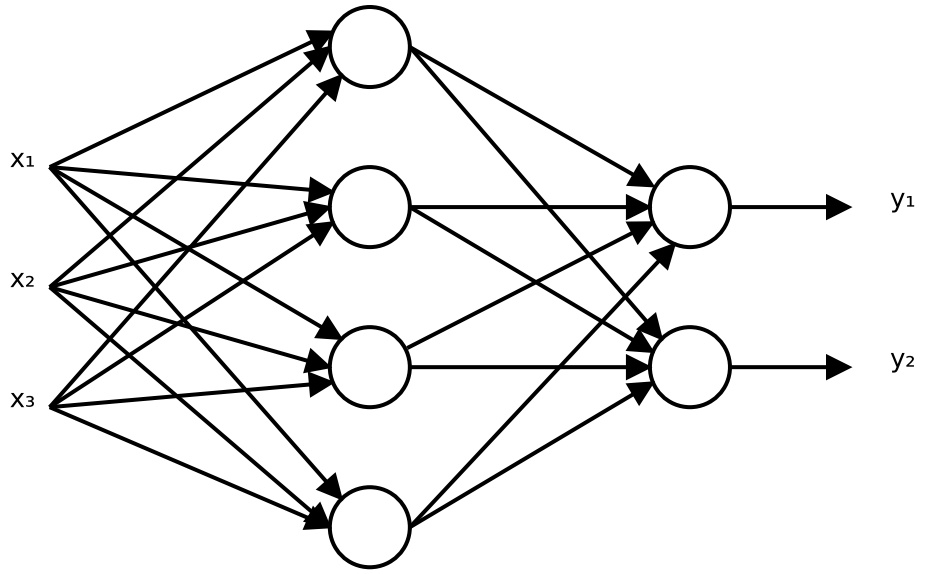
Figur 2.3. ANN neuron

Formellt kan utsignalen ifrån en neuron, y, ses som en funktion av insignalerna, x_i och vikterna, w_i, där φ är en aktiveringsfunktion som bestämmer mängden utsignal utifrån mängden insignal:

$$y = \varphi(\sum_i w_i x_i) \tag{2.13}$$

Det finns många olika sätt att koppla ihop nät av neuroner. Det är dock bevisat att ett nät i två lager, ett som har kopplingar från indata och ett som har kopplingar från det första lagret, kan approximera godtycklig icke-linjär funktion

av indata givet att tillräckligt många neuroner används i det första lagret och att aktiveringsfunktionen är icke-linjär [4]. De lager vars enheter inte direkt genererar utdata brukar kallas gömda lager. Ett exempel på ett tvålagersnät med 3 indata, 4 enheter i det gömda lagret och 2 utdata presenteras i figur 2.4.



Figur 2.4. Flerlagernät

Vid träning av ett nät betraktas par av indata och korrekt utdata för att justera vikterna så att nätets utdata närmar sig den korrekta. En välanvänd metod för att träna flerlagersnät är den s.k. Error Back-propagation metoden, även kallad *BackProp*. En förutsättning för att den metoden ska kunna användas är att aktiveringsfunktionen för alla enheter är deriverbar. BackProp betraktar för varje exempel felet mellan nätets utdata och den korrekta utdatan som en funktion vikterna. Genom att derivera felfunktionen med avseende på varje enskild vikt kan vikterna justeras i den riktning felet minskar mest. Vanligt använda aktiveringsfunktioner med BackProp är sigmoidfunktioner så som $f(x) = \frac{1}{1+e^{-x}}$ och $tanh(x)$. [4]

2.4 Tidigare studier

Kombinationen av belöningsbaserad inlärning och artificiella neurala nät har tidigare framgångsrikt använts för brädspel. Det mest kända exemplet är Gerald Tesauros TD-Gammon från 1992, där ett program lärde sig att spela Backgammon på likvärdig nivå med de bästa mänskliga spelarna. Implementationen använde sig av belöningsbaserad inlärning med TD(λ) och ett tvålagers artificiellt neuralt nät som approximerade värdesfunktionen. Agenten tränades upp genom att spela matcher mot sig själv 1 500 000 gånger. När representationen av spelplanen endast visade

vilka pjäser som fanns på vilken plats nådde programmet en nivå som var likvärdig med en medelmåttig mänsklig spelare. Tesauro utökade senare representationen med särskilt valda egenskaper ur speltillståndet, vilket fick programmet att prestera i klass med de bästa spelarna. En intressant effekt var även att programmet använde öppningsdrag som bland experter tidigare betraktats som riskabla, men efter TD-gammons framgång betraktas de nu som standarddrag. [2]

Samma metod har även applicerats på andra brädspel. För vissa spel har resultaten inte blivit lyckade, t ex schack och Go. Imran Ghory diskuterar i en rapport om belöningsbaserad inlärning för brädspel [5] vilka egenskaper som ett brädspel bör besitta för att det ska gå att lära sig med belöningsbaserad inlärning och ANN. Han menar att en viktig egenskap är att den sanna värdesfunktionen är slät, dvs att två tillstånd vars representation skiljer lite även ska skilja lite i värdet av funktionen. Ghory menar vidare att fyra-i-rad besitter rätt egenskaper för att kunna läras med belöningsbaserad inlärning och ANN. Han implementerar även samma metod som i TD-Gammon, fast för fyra-i-rad, och når som bäst 85% vinster mot en spelare söker med minimaxalgoritmen (se sektion 2.2) på ett litet djup efter sluttillstånd i spelet, men spelar slumpmässigt när inga sådana hittas. Detta resultat nås efter självinlärning med endast 10 000 spelade matcher.

Martin Stenmark har genomfört en annan studie som applicerar $TD(\lambda)$ och ett tvålagers ANN på fyra-i-rad. Agenten tränades här inte genom självinlärning utan istället mot en minimaxspelare som använder en evalueringsfunktion som värderar lagda brickor utifrån hur många vinnande rader de kan vara med i och vilken spelare de tillhör. Agenten tränades under 100 000 matcher. Resultaten mot spelaren som agenten tränar mot är goda, med tanke på att den spelaren är någorlunda avancerad. Mot en slumpmässig spelare vinner agenten i ungefär 80% av matcherna. [9]

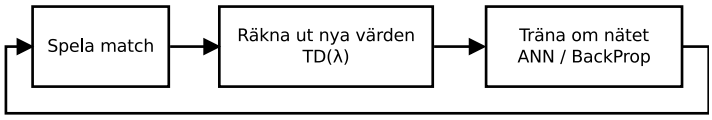
Stenmarks resultat antyder att agenten har lätt att lära sig att slå den spelare den tränar mot, men blir en sämre spelare generellt. Efter 100 000 matcher borde agenten ha lärt sig tillräckligt för att slå en slumpmässig spelare mycket oftare än i 80% av matcherna.

Kapitel 3

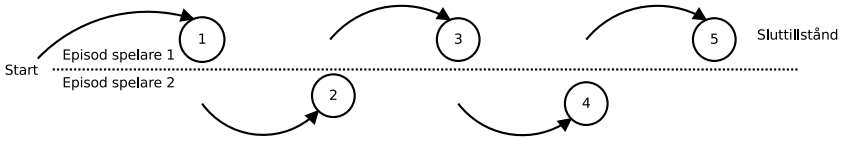
Metod

De goda resultat som uppnåtts i tidigare studier med TD(λ) och ANN för brädspel antyder att denna metod kan vara lämplig även för fyra-i-rad. Som nämnts tidigare är tillståndsrummet väldigt stort, vilket för belöningsbaserad inlärning kräver att värdesfunktionen går att lagra på ett effektivt sätt. ANN uppfyller den egenskapen och har dessutom, som påpekats tidigare, egenskapen att funktionen generaliseras för osedd indata.

Vi har valt att träna agenten med TD(λ) och låta värdesfunktionen approximeras av ett tvålagers neuralt nät som tränas med BackProp-metoden. Två agenter med samma neurala nät tränas mot varandra, och resultatet av varje match används för att träna om nätverket. Figur 3.1 illustrerar träningsförloppet. Ett matchresultat består av två olika episoder, en för vardera spelare. En episod för en spelare innehåller alla speltillstånd som spelaren har valt att gå till under matchen. Tillstånden ifrån en episod kan då användas för att uppdatera värdesfunktionen för de tillstånden, beroende på episodens resultat. Indelningen av matchtillstånd i episoder illustreras i figur 3.2.



Figur 3.1. Inlärningscykel



Figur 3.2. Indelning av tillstånd i episoder för en match med fem tillstånd, ej inräknat starttillståndet

Om en spelare vinner är antagandet att den gjorde bra val under matchen vilket ska öka värdena för episodens tillstånd. Vid förlust ska värdena istället minska. De nya värdena ska beräknas enligt TD(λ) för att sedan användas som målvärden för det neurala nätet när detta tränas efter matchens slut. Belöning ges endast i slutet av en episod, och är 1 för en vinnande episod och -1 (bestraffning) för en förlorande episod. Om en match slutar oavgjort blir belöningen 0.

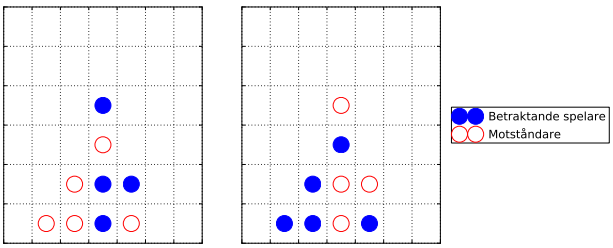
3.1 Representation av tillstånd

Sättet som ett tillstånd representeras på påverkar indatan till det neurala nätet. Olika representationer kan ge olika resultat under inlärningen. Vi har valt att testa två olika representationer.

R1 - enkel Ett tillstånd representeras som en vektor $\mathbf{x} = (x_1, x_2, ..., x_{42})$ av alla olika $6 \cdot 7$ rutor på spelbrädet, där ett element har ett av värdena 0, 1 eller -1. Om ingen bricka ligger på ruta i är $x_i = 0$, om den betraktande spelaren har en bricka på ruta i är $x_i = 1$ och om motståndaren har en bricka på ruta i är $x_i = -1$.

R2 - separerad Ett tillstånd representeras som en vektor $\mathbf{x} = (x_1, x_2, ..., x_{84})$ med två element för varje ruta på spelplanen. Här är x_1 och x_2 markörer för ruta 1. Om den betraktande spelaren har en bricka i ruta 1 är $x_1 = 1$ och $x_2 = 0$. Om motståndaren har en bricka i ruta 1 är istället $x_1 = 0$ och $x_2 = 1$. Ingen bricka ger $x_1 = 0$ och $x_2 = 0$. På motsvarande sätt bestämmer varje ruta två element i vektorn.

Tillstånden i båda episoderna efter en match representeras ur respektive spelares synvinkel, se figur 3.3. På detta sätt lär sig agenten som om den hade spelat båda sidor samtidigt.



Figur 3.3. Samma speltillstånd sett ur de båda spelarnas synvinkel

3.2 Parametrar för TD(λ)

Beräkningen av målvärden med TD(λ) påverkas av flera parametrar, se ekvation 2.12. Parametern η (se TD i sektion 2.3.1) påverkar hur stora steg som tas under inlärningen. Den stegvisa inlärningen kommer i detta system ske i nätet, så här väljs $\eta = 1$. Parametern γ (se ekvation 2.4) påverkar hur agenten värderar belöning som ligger långt fram i tiden. Agenten får i detta problem endast en belöning, vilket sker i slutet. Eftersom agenten till fullo ska ta hänsyn till den belöningen sätts $\gamma = 1$.

Parametern λ (se TD(λ) i sektion 2.3.1) påverkar hur mycket en upptäckt oregelbundenhet i värdesfunktionen ska propagera tillbaka till tidigare tillstånd i en episod. Det är intressant att undersöka olika värden på λ eftersom inga definitiva resultat visats för vilket värde som bör användas. Speciellt intressant är att jämföra olika värden med fallet $\lambda = 0$, då TD(λ) är likvärdig med TD. Vi har valt ut tre testfall:

- **L1**: $\lambda = 0$
- **L2**: $\lambda = 0.3$
- **L3**: $\lambda = 0.6$

Under inlärning används policyn ϵ-girig, med en linjärt avklingande sannolikhet för slumpade drag, från $\epsilon = 0.5$ till $\epsilon = 0$. Detta innebär att agenten fokuserar på att utforska spelträdet i början, medan den mot slutet istället fokuserar på att korrigera värdesfunktionen i de mest intressanta av de hittills utforskade tillstånden.

3.3 Parametrar för ANN

Nätet initialiseras med slumpade vikter i båda lager av neuroner. Det är viktigt att vikterna är anpassade så att de summerade insignalerna till varje neuron i genomsnitt ligger på de icke-linjära delarna av aktiveringsfunktionen, för att kunna approximera icke-linjära funktioner. För ett lager bör varje vikt w_i väljas att vara i intervallet $[-1/n, 1/n]$ där n är antalet insignaler till lagret. [1] Nätet använder aktiveringsfunktionen $tanh(x)$ i både det gömda lagret och utlagret.

Utlagret består endast av en neuron vars utsignal ger värdesfunktionen för indata. Eftersom $tanh(x)$ är aktiveringsfunktion kommer utsignalen från nätet att vara mellan -1 och 1. Antal neuroner i det gömda lagret, n_g, påverkar nätets förmåga att lära sig olika komplexa funktioner. Det är svårt att på förhand uppskatta hur komplex värdesfunktionen behöver vara, vilket gör det intressant att testa uppsättningar med olika antal enheter. Vi har valt att testa fyra olika antal enheter:

- **G1**: $n_g = 20$
- **G2**: $n_g = 40$
- **G3**: $n_g = 80$

- **G3**: $n_g = 120$

För två olika val av representation av indata till nätet ställs nätets inlärningsalgoritm inför skilda problem. Detta innebär att om ett visst antal gömda enheter fungerar bra för en representation behöver det inte nödvändigtvis göra det för en annan. På grund av detta ska de två olika representationerna testas i olika kombinationer med antal gömda enheter.

Vid träning med BackProp av ett tvålagersnät finns två viktiga parmetrar: inlärningshastigheten i det gömda lagret, α, och β i utlagret. Vi har valt $\alpha = 1/n_{in}$, och $\beta = 1/n_g$, där n_{in} är antal insignaler, enligt riktlinjer från Sutton då TD(λ) används i kombination med ett tvålagersnät. [10].

3.4 Utvärdering

För de olika representationerna (R1, R2) ska vi testa fram ett lämpligt antal gömda enheter (G1, G2, G3 eller G4) för att sedan avgöra effekten av λ (L1, L2, L3) för den kombinationen. När antal gömda enheter prövas fram används L1. När dessa parametrar testas fram kommer agenten få träna mot sig själv under 1 000 000 matcher. Den bästa parameteruppsättningen kommer sedan användas i en längre träningsomgång där agenten får träna under 5 000 000 matcher.

Med ett ANN som har slumpade initialvikter kan resultatet av inlärningen bli olika beroende på dessa startvikter. BackProp garanterar inte ett globalt minimum nås för felfunktionen, utan riskerar istället att begränsas i ett lokalt minimum. Beroende på vart vikterna börjar kan felfunktionen se annorlunda ut, och därmed ha olika lokala minimum. Det är därför intressant att utvärdera med olika individer av neurala nät. Då testerna körs på datorer med fyrkärninga processorer passar det väl att testa med fyra nät med olika uppsättningar av initialvikter för varje testfall.

Det är intressant att se hur en agent utvecklas under inlärningen. Därför ska agenten med jämna mellanrum utvärderas, så att dessa data kan sammanställas till en graf över utvecklingen.

Vid varje utvärderingstillfälle spelar agenten mot två typer av förprogrammerade taktiker: en slumpmässig spelare och en mer avancerad minimax-söktaktik. Agenten spelar som startspelare och icke-startspelare varannan match. Dessutom ställs agenten inför specifika speltillstånd där det finns ett rätt drag som garanterat leder till vinst eller måste göras för att undvika förlust. Under utvärderingen använder agenten alltid en helt girig policy, så att den på bästa sätt följer sin värdesfunktion.

3.4.1 Slumpmässig spelare

Vid varje utvärderingstillfälle spelar agenten 1000 matcher mot en spelare som slumpmässigt väljer sina drag. Alla tillåtna drag har lika stor sannolikhet att bli valda. I testresultaten benämns denna spelare som *Slump*.

Agenten kommer i början av inlärningen att spela slumpmässigt eftersom dess värdesfunktion använder sig av ett slumpmässigt initialiserat ANN. Det är därför

intressant att låta agenten möta en annan slumpmässig spelare i detta skede eftersom det då blir enkelt att detektera om inlärningen går framåt. Det kan i ett tidigt skede vara svårt att upptäcka inlärning mot en mer avancerad spelare.

3.4.2 Minimaxspelare

För undersöka om agenten når djupare nivåer av spelförståelse behövs en mer avancerad motståndare än en slumpspelare.

Vi har valt att använda en spelare som använder minimax algoritmen med alphabeta pruning (se 2.2). Spelaren söker på ett visst djup framåt och använder sedan en evalueringsfunktion hämtad från Stenmark [9] som utgår ifrån hur många vinnande rader varje spelad bricka kan ingå i. Se tabell 3.1 för dessa värden. Evalueringsfunktionen summerar detta värde för den betraktande spelarens brickor, och subtraherar motsvarande värde för motståndarens brickor.

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

Tabell 3.1: Antal vinnande rader varje ruta på spelplanen är med i

En minimaxspelare med denna evalueringsfunktion blir helt deterministisk, vilket leder till att samma match kommer upprepas om vi låter den möta en agent med en girig policy. Vi har därför låtit minimaxspelaren slumpa 20% sina drag då agenten spelar mot den, vilket gör att flera olika matcher kan spelas. För att uppnå olika grader av svårighet använder vi fyra olika minimax spelare, som tillåts söka 1, 2, 3 respektive 4 drag framåt i spelträdet. Dessa spelare är i testresultaten benämnda *MM1*, *MM2*, *MM3* och *MM4*.

Under varje utvärderingstillfälle spelar agenten 1000 matcher mot varje minimaxspelare.

3.4.3 Test av specifika speltillstånd

Agenten ställs under utvärdering också inför 60 specifika speltillstånd, benämnda *TS1* till *TS60*. Varje speltillstånd har ett givet drag som är rätt, och utvärderingen består i att se om agenten väljer det rätta draget. Tanken med dessa specifika speltillstånd är att kunna få en mer nyanserad bild av agentens spelförståelse, genom att testa olika koncept. Vi har delat in tillstånden i två huvudkategorier: attack och försvar. I varje attacktillstånd har agenten möjlighet att göra ett drag som leder till vinst direkt. Försvarstillstånden är de inverterade attacktillstånden, där det nu finns ett rätt drag som förhindrar att motståndaren vinner efter sitt nästa

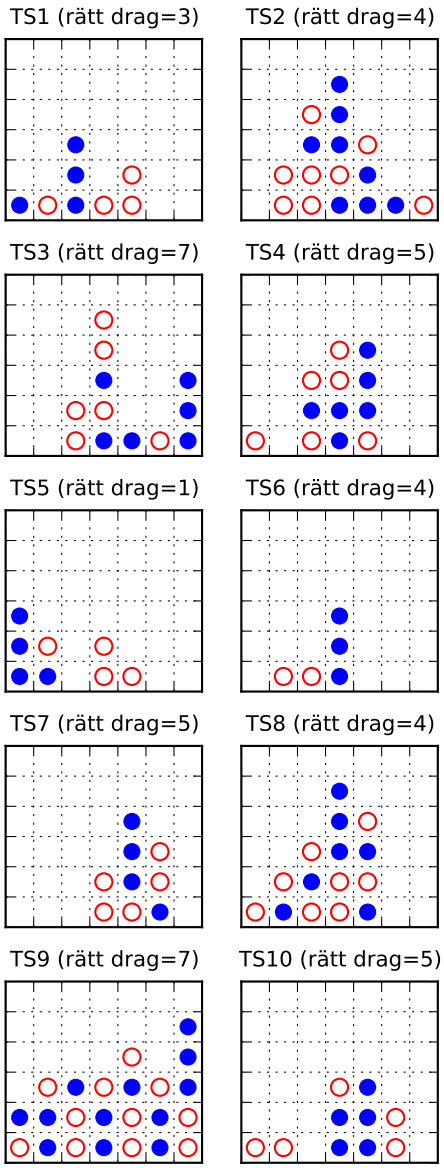
drag. Notera att tillstånden har valts så att det inverterade tillståndet inte ger en möjlighet till egen attack. Inom varje huvudkategori har tillstånden delats in efter vilken typ av vinstmöjlighet som finns: radvinst (horisontellt), kolumnvinst (vertikalt) eller diagonalvinst. Indelningen presenteras i tabell 3.2. Notera att TS31-60 är de inverterade tillstånden för TS1-30. Testfall TS1-30 presenteras i figur 3.4, 3.5 och 3.6, där ifylld ring markerar aktuell spelare.

Benämning	Huvudkategori	Vinsttyp
TS1-10	attack	kolumn
TS11-20	attack	diagonal
TS21-30	attack	rad
TS31-40	försvar	kolumn
TS41-50	försvar	diagonal
TS51-60	försvar	rad

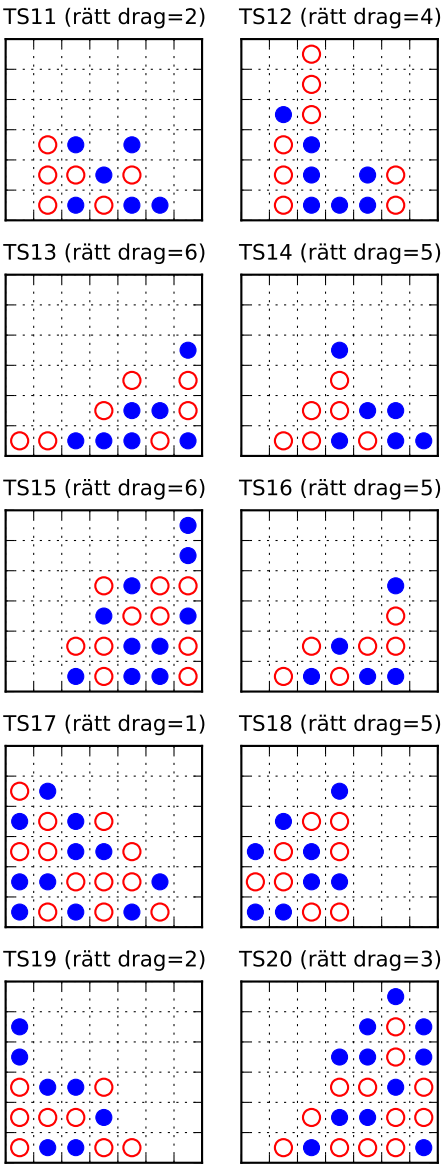
Tabell 3.2: Indelning av testfall för specifika speltillstånd.

Förhoppningen med utvärderingen mot dessa testfall är att se om agenten förbättrar sin förståelse för koncepten attack och försvar, och mer specifikt rader, kolumner och diagonaler.

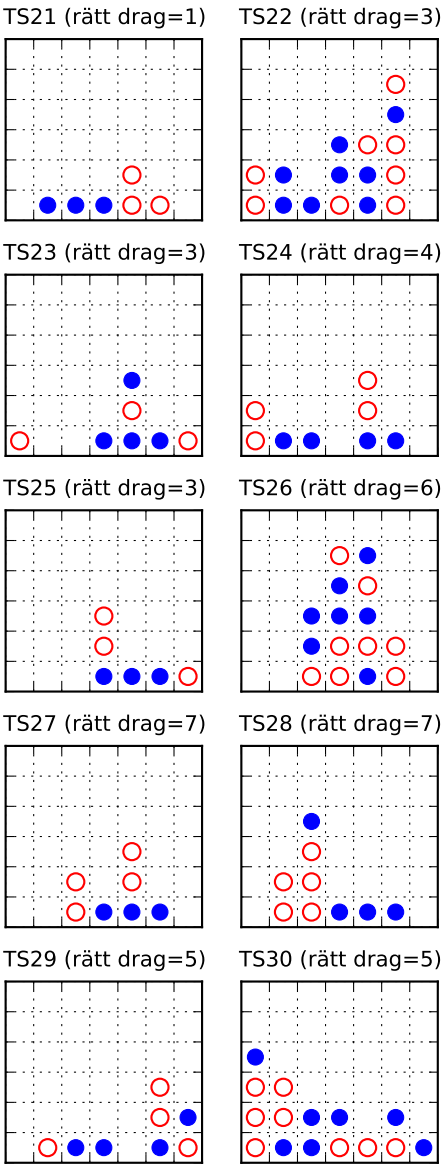
Testfallen har hämtats ifrån matcher där vi har spelat mot en slumpspelare.



Figur 3.4. Testfall för kolumner (ifylld ring = aktuell spelare)



Figur 3.5. Testfall för diagonaler (ifylld ring = aktuell spelare)



Figur 3.6. Testfall för rader (ifylld ring = aktuell spelare)

Kapitel 4

Resultat

Testerna har genomförts i två faser. I den första fasen undersöks parametrars påverkan på agentens prestation under inlärning. Här utvärderas agenten endast med avseende på prestation, och då undersöks endast resultaten vid spel mot de förprogrammerade taktikerna.

I den efterföljande fasen används den bästa funna uppsättningen parameterar för att låta agenten träna under lång tid. Här utvärderas agenten både med avseende på prestation och förståelse för spelkoncept. Agenten undersöks därför både mot de förprogrammerade taktikerna och mot specifika speltillstånd.

I alla grafer som presenteras nedan har kurvorna jämnats ut för att lättare kunna utskilja inlärningstrender. Utjämningen har gjorts genom att varje punkt förutom den första och sista, har fått ett nytt värde som är medelvärdet av punkten, föregående punkt och efterföljande punkt. Första och sista punkten har behållit sina ursprungsvärden för att start- och slutresultatet ska kunna utläsas i graferna.

4.1 Inledande tester

4.1.1 Skillnad mellan individer

För att undersöka skillnaden i prestation mellan olika individer genomfördes ett mindre test med parametervärden R1 (enkel representation), G1 ($n_g = 20$), L1 ($\lambda = 0$). Vinstandelen för varje färdigtränad agentindivid mot de förprogrammerade taktikerna visas i tabell 4.1.

Vi ser att det även efter 300 000 matchers inlärning finns en viss skillnad mellan individer. Individ 4 ser t ex ut att ha nästan dubbelt så hög vinstandel mot minimaxspelarna som individ 3.

I resterande tester har vi alltid valt att visa den bästa individen för varje parameterkombination i graferna, men slutresultaten för samtliga individer presenteras i tabellform i Bilaga A. Den bästa individen valdes med hänsyn till slutresultat och huruvida inlärningstrenden var stabil och uppåtgående.

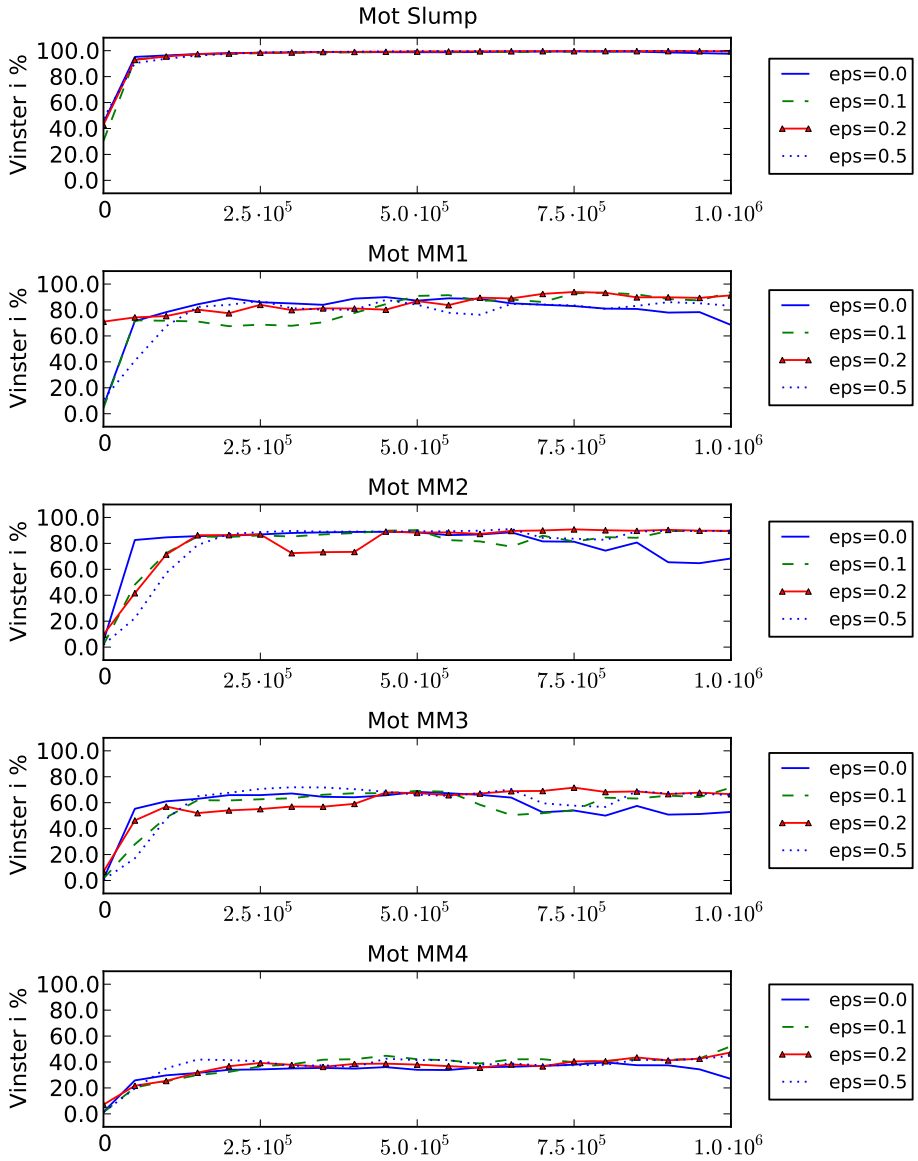
Tabell 4.1. Slutresultat för träningsomgång med enkel representation, $\lambda = 0$ och 20 gömda enheter. Resultatet uppnåddes efter 300 000 träningsmatcher. (R1-G1-L1)

G	Individ	Slump	MM1	MM2	MM3	MM4
G1	1	0.93	0.68	0.77	0.50	0.25
G1	2	0.94	0.72	0.64	0.45	0.26
G1	3	0.90	0.37	0.27	0.23	0.18
G1	4	0.94	0.87	0.49	0.51	0.35

4.1.2 Korrigering av ϵ-avklingande i ϵ-girig policy

Vid ett första genomförande av testerna i fas 1 och 2 noterade vi att resultaten avtog mot slutet av inlärningen, oavsett hur många matcher som spelades eller vilka parametervärden som användes. Ett rimligt resultat hade varit att agenten visar en stadig ökning i prestation under hela inlärningen.

Parametern ϵ i den ϵ-giriga policyn är den enda som förändras under inlärningens gång, eftersom den avklingar linjärt ifrån $\epsilon = 0.5$ till $\epsilon = 0$ (se 3.2). Vi undersökte därför hur resultaten påverkades av att låta ϵ avklinga mot ett högre värde än 0. Vi testade med slutvärdena 0.1, 0.2 och 0.5 (ingen avklingning). Under testet användes den enkla representationen (R1), $\lambda = 0$ (L1) och 80 gömda enheter (G3) och agenten fick spela 1 000 000 matcher. Resultatet presenteras i figur 4.1.



Figur 4.1. Variation av slutvärde på ϵ

Det är tydligt i figuren att de testade slutvärdena (0.1, 0.2, 0.5) inte ger upphov till avtagande prestation mot slutet av inlärningen, vilket särskilt syns mot MM2.

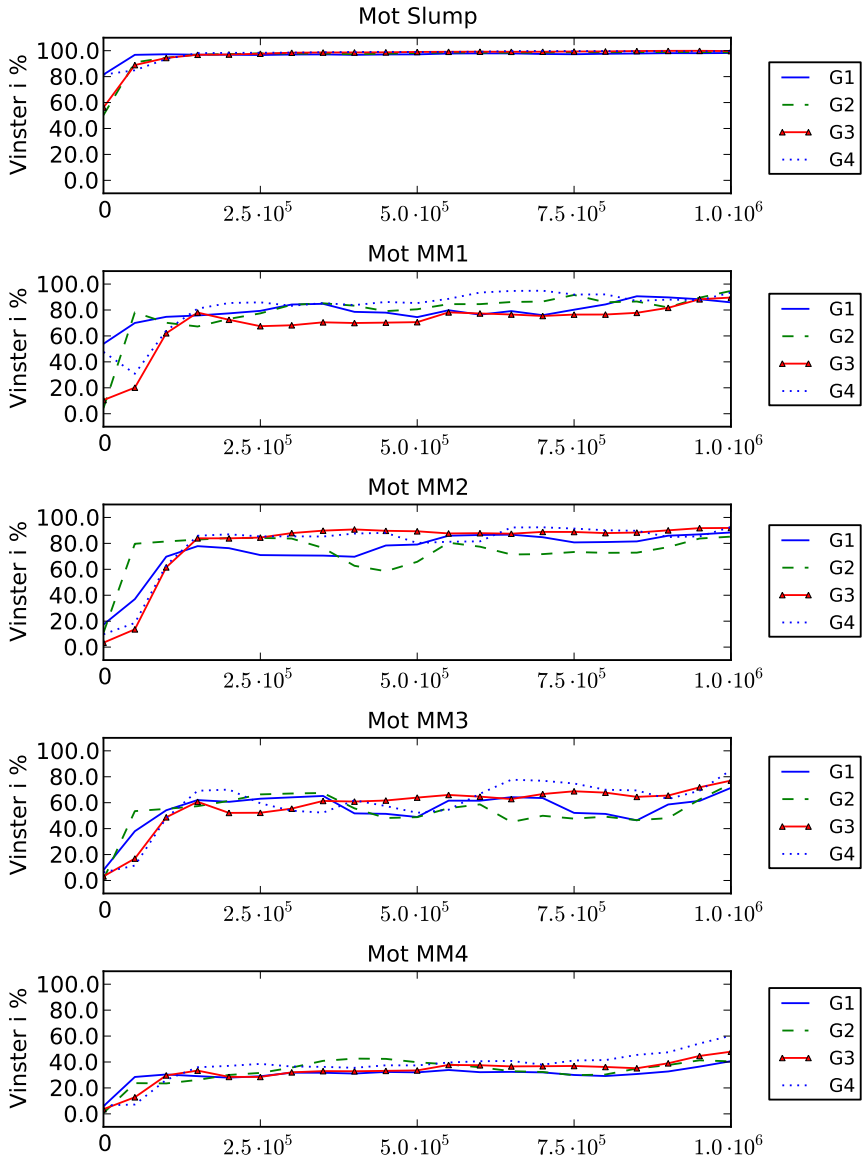
Vi har därför valt genomföra testerna ytterligare en gång, då ϵ avklingar mot värdet 0.1 istället för 0. Det är dessa resultat som presenteras i sektion 4.2 och 4.3.

4.2 Fas 1: Parameterval för god prestation

De parametrar som ingår i testerna är val av representation för speltillstånd (R1, R2), antal gömda enheter i agentens ANN (G1, G2, G3, G4) och värde på λ (L1, L2, L3) för TD(λ). Se sektion 3.1-3.3 för förklaring av parametrar. För varje testad parameterkombination tränades fyra olika agentindivider, med olika initierade ANN (se sektion 3.4), i 1 000 000 matcher mot sig själva. I resultaten nedan visas endast de bästa individerna.

4.2.1 Antal gömda enheter för den enkla representationen

För den enkla representationen (R1) undersöktes vilket antal gömda enheter, n_g, i agentens ANN som gav bäst prestation. De värden som undersöktes var $n_g = 20$ (G1), $n_g = 40$ (G2), $n_g = 80$ (G3) och $n_g = 120$ (G4). Under dessa tester användes $\lambda = 0$ (L1). En graf över prestationsutvecklingen för de bästa individerna för varje val av antal gömda enheter presenteras i figur 4.2. En tabell över slutresultaten för alla individer presenteras i Bilaga A, tabell A.1.



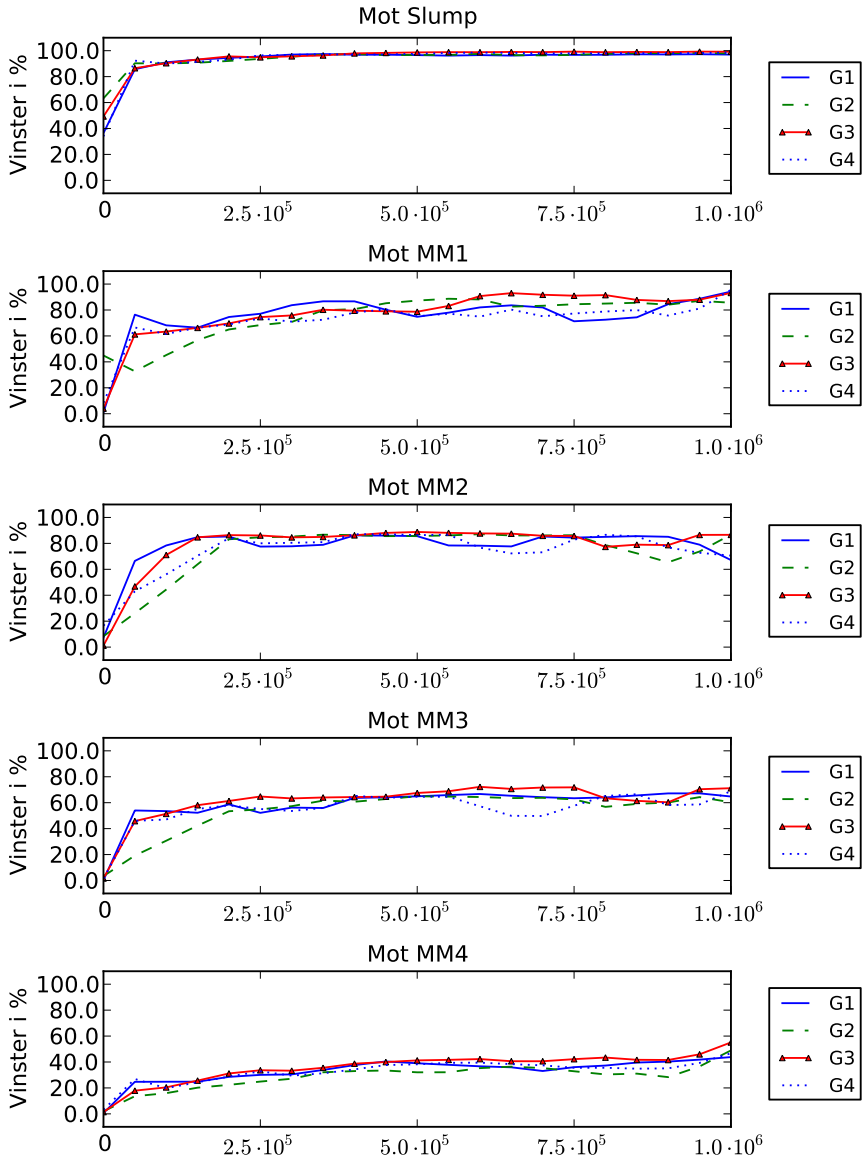
Figur 4.2. Resultat för R1-L1, med olika antal gömda enheter (G1, G2, G3, G4)

Det är tydligt i figur 4.2 att alla agenter snabbt blir bra mot den slumpmässiga spelaren. Det finns också en stark uppåtgående inlärningstrend mot MM1-MM3. Agenterna presterar gradvis sämre ju svårare motståndet är (MM1-MM4). För den svåraste minimaxspelaren (MM4) ser vi att alla agenter förlorar nästan alla matcher i början, men mot slutet lyckas nå en vinstandel runt 40-60%.

Vi ser att resultaten är snarlika för agenter med olika antal gömda enheter, men att agenten med 120 enheter (G4) prestar något bättre mot det svåraste motståndet (MM4). För vidare tester med R1 har G4 använts.

4.2.2 Antal gömda enheter för den separerade representationen

När den separerade representationen (R2) testades användes $\lambda = 0$ (L1). En graf över prestationsutvecklingen för de bästa individerna för varje val av antal gömda enheter (G1, G2, G3 eller G4) presenteras i figur 4.3. En tabell över slutresultaten för alla individer presenteras i Bilaga A, tabell A.2.



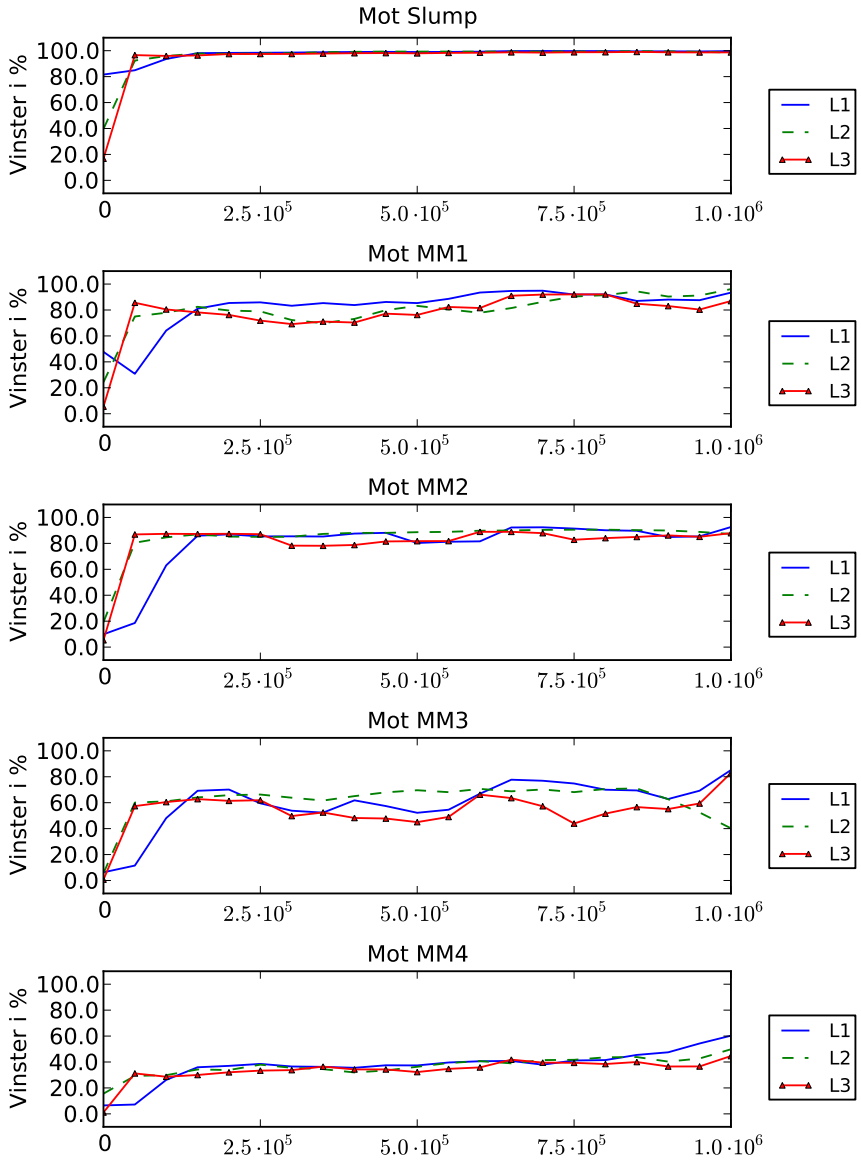
Figur 4.3. Resultat för R2-L1, med olika antal gömda enheter (G1, G2, G3, G4)

I figur 4.3 ser vi återigen att agenterna presterar snarlikt. Det är svårt att välja ut det bästa antalet gömda enheter, men agenten med 80 enheter (G3) uppnår de

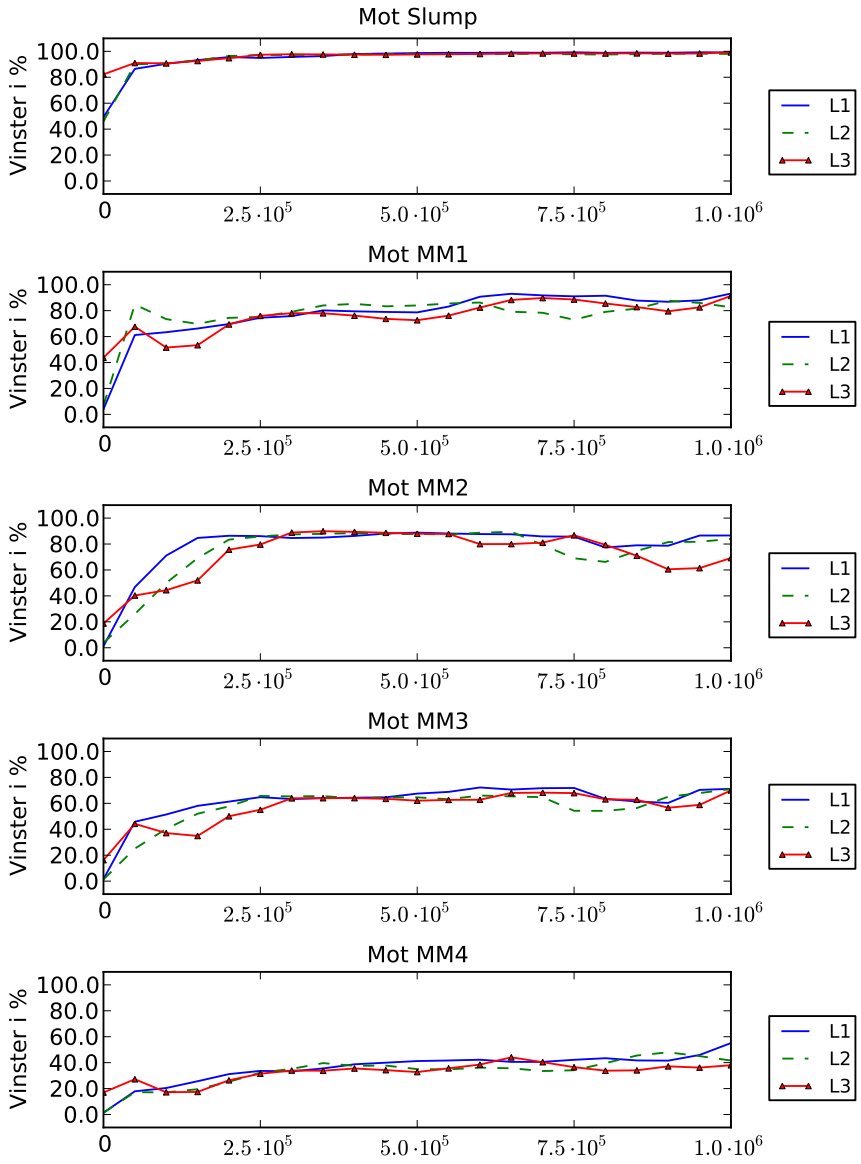
bästa slutresultaten. För vidare tester med R2 har G3 använts.

4.2.3 Parametern λ

För att grundligt undersöka effekten av olika värden på λ testades båda representionerna R1 och R2 med sitt respektive bästa antal gömda enheter (R1-G4, R2-G3). De värden som undersöktes var $\lambda = 0$ (L1), $\lambda = 0.3$ (L2), $\lambda = 0.6$ (L3). Grafer över prestationsutvecklingen för de bästa individerna för varje val av λ presenteras i figur 4.4 och 4.5. En tabell över slutresultaten för alla individer presenteras i Bilaga A, tabell A.3 och A.4.



Figur 4.4. Resultat för R1-G4, med olika värden på λ (L1, L2, L3)

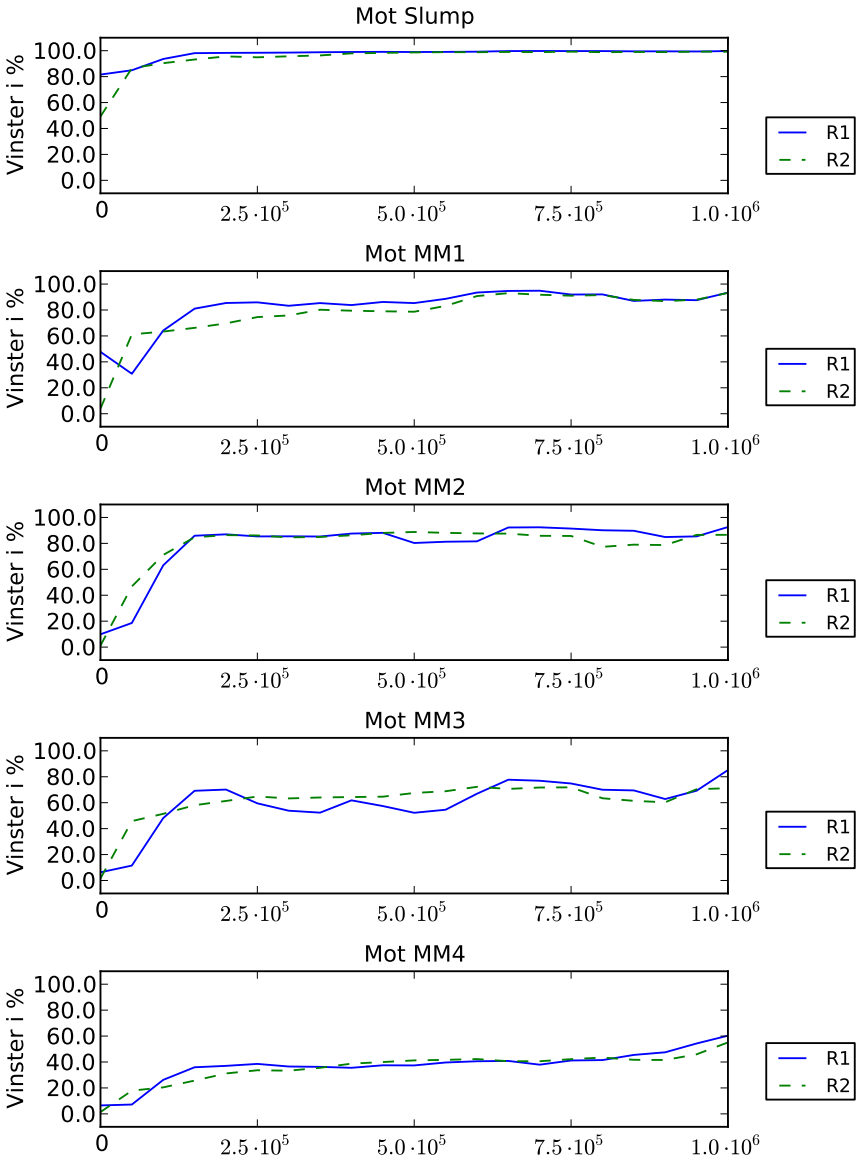


Figur 4.5. Resultat för R2-G3, med olika värden på λ (L1, L2, L3)

I figurerna 4.4 och 4.5 ser vi liknande trender som i förra testet. Agenterna har svårast mot de minimax spelare som söker djupast. Resultaten för olika värden på λ är snarlika. Mot slutet presterar agenten med $\lambda = 0$ (L1) något bättre, med båda representationerna. För vidare tester har L1 använts.

4.2.4 Val av representation

För att jämföra de olika representationerna har vi ställt de bästa parameterkombinationerna för vardera representation mot varandra i figur 4.6. De valda kombinationerna är R1-G4-L1 och R2-G3-L1.

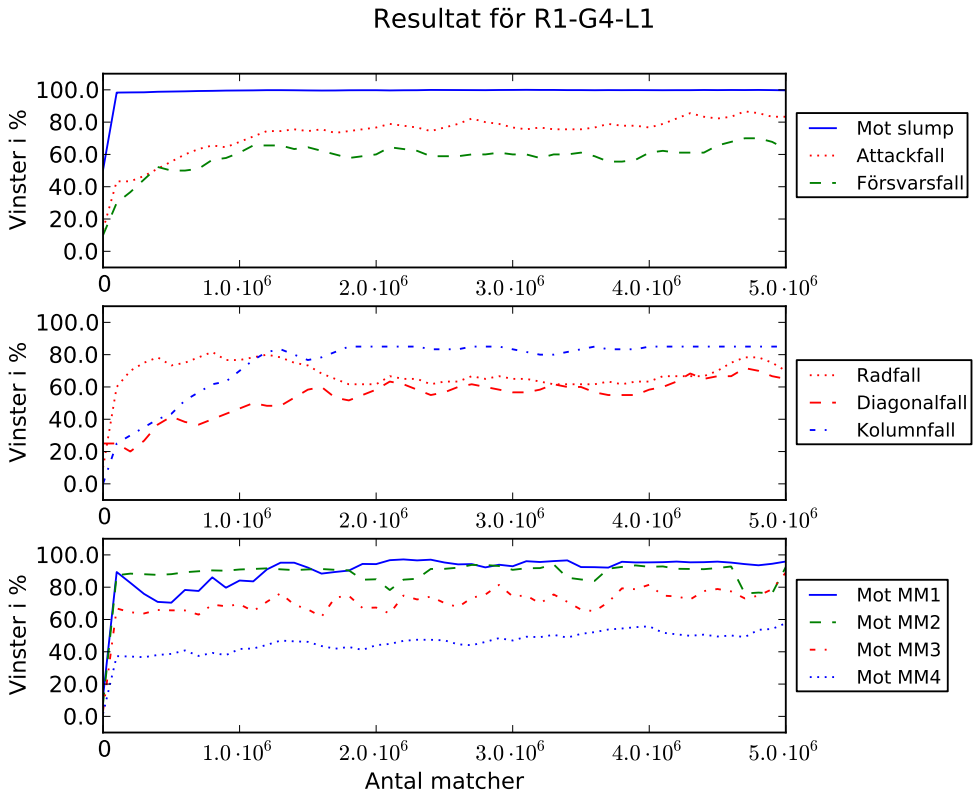


Figur 4.6. Resultat för R1-G4-L1 och R2-G3-L1, för att jämföra de olika representationerna

Vi ser i figur 4.6 att agenterna som använder de olika representationerna presterar väldigt lika. Av praktiska skäl väljer vi att använda R1 i fas 2. Detta eftersom nätet i R2 har fler vikter, vilket gör att det går långsammare att träna. Eftersom representationerna presterar likvärdiga resultat använder vi den som går snabbast att träna för längre tester.

4.3 Fas 2: Prestanda och koncept vid inlärning under lång tid

I denna fas användes den enkla representationen (R1) med 120 gömda enheter (G4) och $\lambda = 0$ (L1). Dessa parametrar valdes utifrån resultaten i tester från fas 1. Vi tränade fyra individer under 5 000 000 träningsmatcher. Resultatet för den bästa individen presenteras i figur 4.7. Den första grafen visar andel vinster mot slumpspelaren, samt andel avklarade testfall i grupperna attack och försvar. I den andra grafen visas andel avklarade testfall för grupperna rader, diagonaler och kolumner. I den sista grafen visas andel vinster mot minimaxspelarna. En tabell över slutresultaten för alla individer presenteras i Bilaga A, tabell A.5.



Figur 4.7. Resultat för enkel representation (R1), 120 gömda enheter (G4) och $\lambda = 0$ efter träning under 5 000 000 matcher

I figur 4.7 ser vi att agenten, som även noterats tidigare, snabbt blir riktigt bra mot slumpspelaren. I alla utvärderingskategorier syns en svag men stabil ökning i resultaten under hela inlärningen. Liksom i tidigare tester syns att agenten förlorar oftare mot de minimaxspelare som söker djupast.

I början ser vi en kraftig utveckling av antalet klarade testfall. Efter 1 000 000

spelade matcher är utvecklingen inte lika stark, men vi ser ändå en viss ökning under den resterande inlärningen. Agenten klarar mot slutet 85% av kolumnfallen, medan den har svårare för diagonal och kolumnfall, där agenten endast klarar runt 70% av fallen.

Vi ser att agenten presterar något bättre på attackfallen än i försvarsfallen. Genomgående under hela inlärningen klarar agenten 10-20 fler procentenheter fler attackfall än försvarsfall. Mot slutet klarar agenten runt 80% av attackfallen och 60% av försvarsfallen.

Kapitel 5

Diskussion

5.1 Utvärderingsmetoder

5.1.1 Förprogrammerade taktiker

För att få en förståelse för hur bra de förprogrammerade taktikerna är har vi ställt dem mot varandra och mot testfallen med speltillstånd. Vinstandelen för respektive taktik presenteras i tabell 5.1 och 5.2. I den första tabellen har Minimaxtaktikerna samma sannolikhet att välja slumpdrag som när de används för att utvärdera den tränade agenten, dvs 20%. I den andra tabellen visas deras fulla kapacitet, då de spelar utan något slumpmässigt inslag. Nedan betecknar *A, F , R, D* och *K* konceptgrupper för speltillstånd. A och F står för koncepten attack och försvar (TS1-30 respektive TS31-60). R, D och K står för koncepten för rader (TS21-30 och TS51-60), diagonaler (TS11-20 och TS41-50) och kolumner (TS1-10 och TS31-40).

Tabell 5.1. Vinstandel för förprogrammerade taktiker, P(slumpdrag) = 0.2

Taktik	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
MM1	0.89	0.52	0.42	0.21	0.16	1.00	0.17	0.50	0.65	0.60
MM2	0.93	0.56	0.48	0.37	0.24	0.80	0.77	0.70	0.95	0.70
MM3	0.97	0.77	0.62	0.49	0.37	0.77	0.83	0.85	0.70	0.85
MM4	0.98	0.86	0.77	0.66	0.51	0.83	0.70	0.90	0.70	0.70
Slump	0.49	0.10	0.08	0.03	0.01	0.07	0.13	0.10	0.00	0.20

Tabell 5.2. Vinstandel för förprogrammerade taktiker, P(slumpdrag) = 0.0

Taktik	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
MM1	0.94	0.50	1.00	0.00	0.00	1.00	0.13	0.50	0.65	0.55
MM2	0.95	0.00	0.50	0.50	0.00	1.00	1.00	1.00	1.00	1.00
MM3	0.99	1.00	0.50	0.50	0.00	1.00	1.00	1.00	1.00	1.00
MM4	1.00	1.00	1.00	1.00	0.50	1.00	1.00	1.00	1.00	1.00
Slump	0.49	0.10	0.08	0.03	0.01	0.07	0.13	0.10	0.00	0.20

Det är tydligt att minimaxspelarna som söker på ett längre djup är bättre, både mot slumpspelaren och mot de som söker på ett kortare djup. Just i fallet då minimaxspelarna inte spelar med slump säger den inbördes statistiken inte så mycket, eftersom de då endast spelar två olika matcher (en med respektive spelare som startspelare). Utan slump klarar MM2-MM4 alla testfall, vilket är naturligt eftersom de söker efter faktiska sluttillstånd för både sig själva och motspelaren. MM1 söker endast på djup 1, och kan då endast hitta sluttillstånd för sig själv, och misslyckas därför på försvarstillstånden. I fallet med 20% slumpdrag är resultaten liknande, men minimaxspelarna misslyckas i ungefär 20% av testfallen.

Att döma av tabellerna ovan tycks de fem förprogrammerade taktikerna erbjuda gradvis svårare motstånd, ifrån den naiva slumpspelaren till den djupsökande MM4-spelaren.

När vi själva spelade mot minimaxspelarna märkte vi att de var svåra att slå när de fick söka på djup 3 och 4. De hittar då dels vinsttillstånd som ligger nära i tiden, och kan använda evalueringsfunktionen för övriga tillstånd. Vi förlorade majoriteten av matcherna mot MM4. Svårigheten tyder på att evalueringsfunktionen som används av minimaxspelarna är en rimlig approximation av vad som är viktigt i spelet.

5.1.2 Specifika testade speltillstånd

De speltillstånd som använts har skapats när vi spelat mot en slumpad spelare. Vi försökte aktivt få till testfall i de kategorier som valts. Detta kan ha resulterat i att speltillstånden kanske inte är särskilt troliga att dyka upp i vanliga matcher. Matcher mellan två bra spelare kommer endast besöka speltillstånd i en viss del av speltillståndsrummet, eftersom spelarna försöker undvika att göra dåliga drag. En mänsklig spelare kan förmodligen ganska lätt klara av våra testfall även om de är osannolika i en vanlig match.

Den datorspelare som vi har låtit träna baserar sin kunskap starkt på de speltillstånd den har sett under inlärningen. ANN har en möjlighet att generalisera värdesfunktionen för tillstånd som liknar de sedda, men kan förmodligen inte göra generaliseringar på samma nivå som en människa. När datorspelaren ställs inför ett tillstånd som är väldigt olikt allt den sett tidigare är approximationen av värdesfunktionen i det tillståndet sannolikt dålig.

Detta innebär att vår datorspelare kan prestera dåligt på testfallen som potentiellt är ovanliga i matcher med åtminstone medelbra spelare. Det är dock inte uteslutet att en spelare som är dålig i ovanliga tillstånd kan spela framgångsrikt. Om spelaren lyckas behålla spelet i den bekanta delen av tillståndsrymden genom att göra bra egna drag minskar risken att behöva använda värdesfunktionen där den inte är bra approximerad.

5.2 Resultat

5.2.1 Inledande tester

Skillnad mellan individer

I tabell 4.1 ser vi att det kan finnas betydande skillnader i prestation mellan olika individer. Även i tabellerna för senare tester (se Bilaga A) syns märkbara skillnader mellan individer. Eftersom den valda metoden påverkas av slump, både i initieringen av ANN och i hur agenten väljer drag, är det viktigt att använda olika individer. Vi använde fyra olika individer för varje test, men det är möjligt att flera individer ger rättvisare resultat.

Avtagande prestation i slutet av inlärningen

Under den första omgången av tester var en övergripande observation i resultaten att oberoende av hur många matcher som tränats och vilka parametervärden som användes så avtog resultaten mot slutet av inlärningen. Detta är märkligt eftersom agenten borde få mer erfarenhet desto fler matcher den spelar.

Eftersom andelen slumpade drag, ϵ, var den enda parametern som förändrades under inlärningens gång så antog vi att den kunde bidra till de avtagande resultaten. Vi valde därför att undersöka hur ϵ påverkade inlärningen. I den första omgången tester avklingade ϵ ifrån 0.5 till 0.

I figur 4.1 ser vi att agentens prestation inte sjunker på slutet då ϵ klingar av mot de andra testade värdena (0.1, 0.2, 0.5), medan det syns en tydlig försämring då ϵ klingar av mot 0. Detta stärker antagandet att ϵ var källan till problemet.

Det är svårt att spekulera i varför ϵ inte borde avklinga till 0. En teori är att en låg andel slumpade drag mot slutet av inlärningen gör att agenten endast spelar matcher i en väldigt begränsad del av tillståndsrymden, och därför fokuserar på att uppdatera approximationen av värdesfunktionen för mycket bara i dessa tillstånd. En effekt av detta kan bli att approximationen försämras i andra tillstånd. När agenten sedan utvärderas mot de förprogrammerade taktikerna är risken stor att matcherna inte rör sig i det begränsade utrymmet i tillståndsrymden som agenten fokuserat inlärningen på mot slutet.

5.2.2 Fas 1

Antal gömda enheter och representation

För den enkla representationen ser vi i figur 4.2 att resultaten är snarlika för agenter med olika antal gömda enheter i nätet. Agenterna med 20 och 40 enheter (G1 och G2) har något lägre vinstandel, framförallt mot de svårare motståndarna (MM3 och MM4). De andra agenterna (G3 och G4) ligger väldigt nära varandra, förutom mot den svåraste motståndaren (MM4), då G4 vinner fler matcher.

Eftersom skillnaden är liten mellan olika agenter är det svårt att dra stora slutsatser. Det verkar dock finnas en tendens till att det är bättre med fler gömda enheter. Detta kan tyda på att värdesfunktionen inte är trivial, utan kräver ett större antal gömda enheter för att approximera på ett bra sätt.

För den separerade representationen är skillnaden mellan agenter med olika antal gömda enheter ännu svårare att uttyda. Vi ser i figur 4.3 att G3 når något högre slutresultat medan de andra agenterna ligger samlat. Likheten i resultaten mellan agenterna kan tyda på att alla har ett antal enheter som passar för att approximera värdesfunktionen med den här representationen. Det är möjligt att skillnaderna blir större om agenterna tränas under fler matcher.

Vid jämförelse av de två olika representationerna (se figur 4.6) är resultaten snarlika. Även här kan vi inte utesluta att resultaten skiljer sig efter fler tränade matcher.

Värdet på λ

När värdet på λ varierades (se figur 4.4 och figur 4.5) presterade de olika agenterna likvärdigt under större delen av inlärningen, men agenten med $\lambda = 0$ visar ett något bättre slutresultat.

Det är svårt att dra någon definitiv slutsats för testet av värdet på λ. Med $\lambda > 0$ borde approximationen för värdesfunktionen uppdateras på ett mer effektivt sätt, se TD(λ) i sektion 2.3.1. För de värden som testats syntes ingen betydande skillnad i resultatet. Det antyder att TD(λ) inte presterar bättre än den något simplare Temporal Difference-metoden.

5.2.3 Fas 2

När agenten tränades upp under lång tid uppnåddes något bättre resultat än i fas 1. I figur 4.7 ser vi att resultaten i alla evalueringsfall stadigt ökar under hela inlärningen.

För testfallen för specifika speltillstånd ser vi att agenten lyckas uppnå goda resultat i kolumnfallen. Dock verkar agenten ha svårare att klara rad- och diagonalfallen. Svårigheten att klara testfallen kan bero på deras utformning, se sektion 5.1.2.

Agenten presterar något bättre i attackfall än i försvarsfall. Det är svårt att förklara varför. Det kan betyda att agenten funnit att det är bättre att attackera

än försvara.

När vi själva spelade mot olika motståndare märkte vi att det lättast att missa en möjligt vinstmöjlighet för motståndaren i diagonalfall, medan det är lätt att upptäcka dem för kolumnfall. Det är intressant att se att den upptränade agenten har liknande svårigheter.

5.3 Sammanfattning

Det är tydligt att belöningsbaserad inlärning fungerar för spelet fyra-i-rad. Vi lyckas träna en agent som från början spelar slumpmässigt till att slå en slumpspelare i nästan samtliga matcher. Mot den bästa förprogrammerade taktiken (MM4) så utvecklas agenten ifrån att att förlora nästan alla matcher till att vinna närmare 60%. Detta är ett bra resultat eftersom vi funnit att vi själva har svårt att vinna mot MM4.

Det var svårt att upptäcka några tydliga skillnader när de testade parametrarna ändrades. Störst effekt på prestationen åstadkoms genom att träna fler matcher.

En intressant upptäckt var att andelen slumpade drag under inlärningen inte borde klinga av mot 0, eftersom det verkar ge upphov till försämrad prestation mot slutet av inlärningen.

5.4 Vidare studier för fyra-i-rad

En idé är att hämta testfall för specifika speltillstånd ifrån riktiga spelade matcher mellan bra spelare. Resultatet av ett klarat testfall säger då mer om spelarens kompetens än de testfall som vi har skapat ifrån matcher mot en slumpspelare.

Eftersom vi märkte att policyn och dess parameterar spelar roll för inlärningen, kan det vara intressant att undersöka olika policies och dess parametrar närmare.

Litteraturförteckning

[1] Marsland, S. (2009). *Machine Learning: an Algorithmic Perspective*. CSC-Press.

[2] Tesauro, G. J. (1994). TD–gammon: a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.

[3] Sutton R. S. and Barto R. (1998). *Reinforcement learning*. The MIT Press.

[4] Haykin, S. (1999). *Neural Networks: a Comprehensive Foundation*. Prentice Hall.

[5] Ghory, I. *Reinforcement learning in board games* ,`http://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf`, 2011-03-04

[6] Allen, J. D. (2011). *The Complete Book of Connect 4*, Sterling.

[7] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

[8] Tromp J., *John's Connect Four Playground*, `http://homepages.cwi.nl/~tromp/c4/c4.html`, 2011-04-08

[9] Stenmark M. (2005). *Synthesizing board evaluation functions for Connect4 using Machine Learning techniques*. Masteruppsats, Østfold University College.

[10] Sutton, R. S. (1992). *Nonlinear TD/Backprop pseudo C-code*, `http://webdocs.cs.ualberta.ca/~sutton/td-backprop-pseudo-code.text`, 2011-04-08

Bilaga A

Testresultat

Tabell A.1. Slutresultat för träningsomgång med enkel representation, $\lambda = 0$ och varierat antal gömda enheter. Resultatet uppnåddes efter 1 000 000 träningsmatcher. (R1-G*-L1)

G	Individ	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
G1	1	0.98	0.86	0.88	0.71	0.41	0.43	0.40	0.55	0.35	0.35
G1	2	0.97	0.77	0.85	0.63	0.27	0.47	0.30	0.50	0.30	0.35
G1	3	0.94	0.69	0.84	0.53	0.24	0.40	0.27	0.45	0.50	0.05
G1	4	0.99	0.89	0.86	0.74	0.40	0.50	0.33	0.55	0.40	0.30
G2	1	0.99	0.66	0.68	0.58	0.21	0.60	0.40	0.65	0.35	0.50
G2	2	0.99	0.95	0.85	0.75	0.41	0.57	0.40	0.50	0.35	0.60
G2	3	0.98	0.77	0.66	0.42	0.43	0.40	0.43	0.40	0.40	0.45
G2	4	0.99	0.90	0.64	0.74	0.40	0.50	0.40	0.45	0.40	0.50
G3	1	0.99	0.95	0.53	0.59	0.41	0.63	0.53	0.70	0.40	0.65
G3	2	0.99	0.91	0.88	0.66	0.46	0.60	0.53	0.75	0.40	0.55
G3	3	1.00	0.90	0.92	0.77	0.48	0.70	0.63	0.95	0.60	0.45
G3	4	0.99	0.95	0.49	0.56	0.40	0.73	0.67	0.75	0.60	0.75
G4	1	1.00	0.94	0.90	0.68	0.39	0.70	0.50	0.60	0.55	0.65
G4	2	1.00	0.94	0.89	0.48	0.43	0.63	0.67	0.75	0.45	0.75
G4	3	1.00	0.95	0.90	0.79	0.54	0.77	0.73	0.95	0.40	0.90
G4	4	1.00	0.93	0.93	0.85	0.60	0.73	0.47	0.70	0.40	0.70

Tabell A.2. Slutresultat för träningsomgång med separerad representation, $\lambda = 0$ och varierat antal gömda enheter. Resultatet uppnåddes efter 1 000 000 träningsmatcher. (R2-G*-L1)

G	Individ	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
G1	1	0.95	0.87	0.53	0.54	0.40	0.50	0.33	0.55	0.30	0.40
G1	2	0.97	0.94	0.67	0.65	0.44	0.57	0.33	0.55	0.40	0.40
G1	3	0.97	0.92	0.83	0.39	0.31	0.50	0.33	0.65	0.35	0.25
G1	4	0.97	0.78	0.64	0.60	0.34	0.47	0.37	0.70	0.25	0.30
G2	1	1.00	0.92	0.71	0.75	0.43	0.50	0.37	0.55	0.35	0.40
G2	2	0.98	0.76	0.84	0.62	0.39	0.57	0.50	0.65	0.40	0.55
G2	3	0.98	0.86	0.87	0.60	0.49	0.47	0.20	0.45	0.30	0.25
G2	4	0.97	0.94	0.85	0.59	0.35	0.63	0.30	0.55	0.30	0.55
G3	1	0.99	0.93	0.87	0.71	0.55	0.57	0.30	0.60	0.15	0.55
G3	2	0.99	0.96	0.63	0.51	0.54	0.53	0.37	0.55	0.25	0.55
G3	3	0.99	0.90	0.88	0.68	0.49	0.57	0.30	0.60	0.30	0.40
G3	4	0.99	0.94	0.68	0.79	0.44	0.50	0.30	0.40	0.30	0.50
G4	1	0.99	0.81	0.54	0.39	0.35	0.53	0.23	0.50	0.25	0.40
G4	2	0.99	0.65	0.87	0.57	0.35	0.57	0.33	0.70	0.25	0.40
G4	3	0.99	0.96	0.71	0.69	0.47	0.70	0.33	0.45	0.40	0.70
G4	4	0.98	0.93	0.44	0.43	0.35	0.53	0.53	0.65	0.40	0.55

Tabell A.3. Slutresultat för träningsomgång med enkel representation, 120 gömda enheter och varierat värde på λ. Resultatet uppnåddes efter 1 000 000 träningsmatcher. (R1-G4-L*)

L	Individ	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
L1	1	1.00	0.94	0.90	0.68	0.39	0.70	0.50	0.60	0.55	0.65
L1	2	1.00	0.94	0.89	0.48	0.43	0.63	0.67	0.75	0.45	0.75
L1	3	1.00	0.95	0.90	0.79	0.54	0.77	0.73	0.95	0.40	0.90
L1	4	1.00	0.93	0.93	0.85	0.60	0.73	0.47	0.70	0.40	0.70
L2	1	0.98	0.82	0.72	0.43	0.33	0.63	0.47	0.80	0.25	0.60
L2	2	0.99	0.83	0.45	0.34	0.36	0.77	0.60	0.85	0.65	0.55
L2	3	0.99	0.96	0.87	0.40	0.50	0.63	0.43	0.85	0.40	0.35
L2	4	0.99	0.92	0.56	0.56	0.46	0.67	0.57	0.75	0.50	0.60
L3	1	0.99	0.90	0.67	0.81	0.48	0.53	0.47	0.55	0.45	0.50
L3	2	0.99	0.87	0.88	0.83	0.45	0.53	0.40	0.65	0.35	0.40
L3	3	0.99	0.92	0.75	0.48	0.43	0.47	0.37	0.50	0.30	0.45
L3	4	0.98	0.91	0.87	0.70	0.42	0.53	0.53	0.70	0.50	0.40

Tabell A.4. Slutresultat för träningsomgång med separerad representation, 80 gömda enheter och varierat värde på λ. Resultatet uppnåddes efter 1 000 000 träningsmatcher. (R2-G3-L*)

L	Individ	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
L1	1	0.99	0.93	0.87	0.71	0.55	0.57	0.30	0.60	0.15	0.55
L1	2	0.99	0.96	0.63	0.51	0.54	0.53	0.37	0.55	0.25	0.55
L1	3	0.99	0.90	0.88	0.68	0.49	0.57	0.30	0.60	0.30	0.40
L1	4	0.99	0.94	0.68	0.79	0.44	0.50	0.30	0.40	0.30	0.50
L2	1	0.98	0.83	0.84	0.71	0.42	0.63	0.50	0.70	0.55	0.45
L2	2	0.99	0.82	0.84	0.70	0.43	0.63	0.47	0.55	0.60	0.50
L2	3	0.99	0.88	0.44	0.41	0.43	0.43	0.27	0.50	0.15	0.40
L2	4	0.98	0.88	0.87	0.67	0.43	0.50	0.37	0.60	0.35	0.35
L3	1	0.99	0.91	0.69	0.70	0.38	0.47	0.40	0.55	0.35	0.40
L3	2	0.98	0.78	0.66	0.62	0.36	0.60	0.37	0.65	0.30	0.50
L3	3	0.96	0.91	0.55	0.52	0.34	0.47	0.33	0.45	0.45	0.30
L3	4	0.97	0.84	0.46	0.47	0.42	0.47	0.23	0.60	0.20	0.25

Tabell A.5. Slutresultat för träningsomgång med enkel representation, $\lambda = 0$ och 120 antal gömda enheter. Resultatet uppnåddes efter 5 000 000 träningsmatcher. (R1-G4-L1)

L	Individ	Slump	MM1	MM2	MM3	MM4	A	F	R	D	K
L1	1	1.00	0.96	0.92	0.89	0.58	0.83	0.63	0.70	0.65	0.85
L1	2	1.00	0.96	0.88	0.78	0.61	0.70	0.63	0.70	0.55	0.75
L1	3	1.00	0.95	0.89	0.67	0.48	0.73	0.57	0.70	0.55	0.70
L1	4	0.99	0.87	0.67	0.69	0.49	0.70	0.67	0.70	0.60	0.75