Jämförelse av *Reinforcement learning* tekniker vid få observationer och applicerbarhet på GVGAI

Viktor Holmgren
Institutionen för Datavetenskap
Linköping, Sweden
vikho394@student.liu.se

SAMMANFATTNING

I denna rapport så undersöker och jämförs några av de vanligaste självinlärningsalgortimerna. Först så beskrivs algoritmerna övergripande. Varpå deras styrkor, svagheter och nyckelegenskaper jämförs. Vidare så diskuteras hur dessa tekniker och självinlärning i allmänhet presterar vid ytterst få observationer. Avslutningsvis så redogörs hur självinlärning kan tillämpas för GVGAI, både som en direkt del av beslutsfattandet men även indirekt i form av en hyperheuristik eller algortimselektion.

Keywords

Reinforcement learning; Q-learning; Dyna; Prioritized sweeping; GVG-AI; Sample-efficient reinforcement learning

1. INLEDNING

Under de senaste decennierna så har stora framgångar gjorts inom artificiell intelligens (AI). Framgångar som har lett oss mot allt mera intelligenta och generella agenter.

Många av framgångarna har skett som ett resultat av olika typer av tävlingar inom AI. Vi kan till exempel se hur det stora intresset kring självkörande bilar som vi har idag, tog fart som ett resultat av DARPA tävlingarna som hölls i mitten på 2000-talet[11]. Eller hur tävlingar mellan människor och datorer i brädspel såsom schack och go har öppnat upp för helt nya typer av algoritmer. Tävlingar och utmaningar har fortsatt att visa sig vara ett bra sätt att utvärdera och jämföra olika tillvägagångsssätt och metoder inom datavetenskap i helhet.

Vi ser också hur vi rör oss allt mer mot generell AI. Mot agenter som inte bara klarar av en stor mängd olika typer av uppgifter, utan som gör det utan att skaparen har designat dem med specifik domänkunskap för de områden som de agerar i.

Ett av de mest intressanta områdena inom AI och maskininlärning är självinlärning (eng. *reinforcement learning*). Självinlärning möjliggör för en agent att lära sig från erfarenhet utan hjälp utifrån, till skillnad från till exempel handledd inlärning (eng. *supervised learning*). Istället så lär sig agenten baserat på återkopplingen den får från omgivningen som den agerar i.

I denna rapport så börjar jag med att ge en introduktion till några av de mest vanligt förekommande självinlärningsalgoritmerna och hur de står sig mot varandra när de kommer till deras styrkor och svagheter i allmänhet. Sedan så redogör jag för hur självinlärning fungerar under väldigt få observationer. Avslutningsvis så beskriver jag hur dessa algoritmer kan appliceras på *General Video Game AI Competition*.

2. METOD

I denna rapport så börjar jag med att analysera och jämföra ett par olika självinlärningsalgoritmer. Mer bestämt: Q-learning, Dyna och Prioritized Sweeping. I min jämförelse så redogör jag för de styrkor och svagheter som respektive algoritm har och skillnaderna mellan dessa. Mer specifikt så undersöker jag hur dessa algoritmer, och självinlärning i allmänhet, fungerar med få observationer. Det vill säga hur snabbt algoritmerna konvergerar och inte nödvändigvis hur bra den optimala policyn som genereras är. Detta område av självinlärning kallas för effektiv självinlärning eller *Sample-Efficient reinforcement learning*.

Avslutningsvis så beskriver jag på hur självinlärning kan appliceras på GVGAI. Mer specifikt, hur det kan användas som en del av beslutsfattandeprocessen Både direkt och indirekt.

3. BAKGRUND

3.1 GVGAI

General Video Game Artificial Intelligence (GVGAI) är en tävling inom generell AI med flera grenar. Den gren som är av intresse för denna rapport är planeringsgrenen (eng. *planning track*). Grenen går ut på att skapa en intelligent agent som ska klara spela en stor mängd olika typer av två-dimensionella arkadspel såsom *Pacman* eller *Space Invaders*. Agentens uppgift är att i varje givet steg besluta vilken handling (eng. *action*) som ska utföras. Till sin hjälp så har denna tillgång till spelplanen samt en framtidsmodellen (eng. *forward-model*) som den kan använda för att ställa frågor om hur framtiden ser ut om en viss handling utförs. Notera dock att detta är stokastiskt och svaret från framtidsmodell kan därför skilja sig från vad som faktiskt inträffar om handlingen utförs. Beslutsfattandet har även realtidskrav i form av en maxtid av 40ms[2].

3.2 Reinforcement learning

Inom maskininlärning finns det en mängd olika algoritmer och tillvägagångssätt. Dessa algoritmer kan delas in i tre huvudsakliga kategorier baserat på hur återkopplingen, även kallat signalen, ges. Vid handledd inlärning observerar agenten exempel i form av par av indata och utdata. Agenten försöker då lära sig en funktion som avbildar indata till utdata. Medan i ohandledd inlärning (*unsupervised learning*) försöker

agenten upptäcka mönster i data, ofta i form att göra någon form av klustring[7].

Självinlärning fungerar på lite annorlunda sätt. Här lär sig agenten utifrån en serie av belöningar (eng. *reinforcements*). Belöningar kan även vara i form av negativa belöningar eller straff. Genom att använda sig av observerade belöningar är en självinlärande agents uppgift att lära sig en optimal (eller nära optimal) *policy* för omgivningen. En optimal policy är den policy som maximerar det förväntade totala belöningen över tid. Det finns även delkategorier av självinlärning. En av dessa delkategorier är TD (*Temportal difference*) algoritmer, där nuvarande estimeringar baseras på tidigare inlärda estimeringar. En väldigt viktig aspekt av självinlärning är *aktiv inlärning*. Här behöver agenten inte bara lära sig nyttan av att utföra en handling i ett visst tillstånd utan måste också lära sig vad den ska göra i ett givet tillstånd. Agenten måste alltså även ta hänsyn till utforskning. För att kunna lära sig vad som är en bra och mindre bra handling i ett visst tillstånd måste agenten pröva något nytt (utforska) och inte bara utföra det som den redan har lärt sig (utnyttjning). Ett stort problem inom självinlärning är att balansera denna utforskning mot utnyttjning. Problemet benämns vanligen *exploration vs exploitation*[7].

3.3 Q-learning

Q-learning algoritmen är kanske den mest välkända självinlärningsalgoritmen. Den presenterades av Watkins i slutet på 80-talet och är en relativt enkel algoritm för att lära sig en optimal policy i en Markoviansk miljö [9].

För att kunna lära sig en optimal policy bygger en Q-learning algoritmen upp en tabell över så kallade Q-värden $Q(s, a)$ som representerar värdet av att utföra handlingen a i tillstånd s. Den optimala handlingen i ett visst tillstånd s är den handling a som på sikt leder till den största belöningen.

Q-learning har ett par väldigt viktiga egenskaper, först är den *modellfri*, vilket betyder att den varken behöver en modell för inlärningen eller för beslutsfattandet. Vidare är den Q-learning en så kallad *off-policy* inlärningsalgoritm vilket innebär att den inte själv är medveten om policyn som den utför, utan väljer bara den handling a i tillstånd s som har störst Q-värde.

Den viktigaste delen i Q-learning algoritmen är uppdateringsfunktionen, som uppdaterar $Q(s, a)$ i och med att agenten agerar i värden. Om en agent utför handling a i tillståndet s som leder till tillstånd s' uppdateras Q-värdet enligt

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \quad (1)$$

där α och γ kallas för inlärningstakt- respektive avdragskonstanten (eng. *learning rate, discount factor*). Det α avgör är till vilken grad agenten prioriterar ny information över tidigare erfarenhet[9]. Man kan säga att α är ett mått på hur villig agenten är att ändra sin åsikt. Allstå, en agent med en låg α faktor är skeptisk till ny information medan en agent med hög α som gärna släpper vad den redan har lärt sig när ny information kommer in.

γ bestämmer vikten av framtida belöningar. En låg faktor leder således till en girig agent som tänker väldigt kortsiktigt och bara intresserad av nuvarande belöningar, medan en hög faktor leder till en agent som tänker väldigt långsiktigt. Dessa koncept med inlärnings- och avdragskonstanten är något som kännetecknar TD algoritmer i allmänhet.

3.4 Dyna

Dyna är en annan självinlärningsalgoritm. Den presenterades av Sutton i början på 90-talet, och inkluderar inlärning, planering och reaktivt handlande. Med detta menas att den använder sig av *trial-and-error* som del av inlärningen av en optimal så kallad reaktiv policy. Det vill säga en avbildning av tillstånd till handlingar. Att den under tiden lär sig en handlingsmodell (*action model*) som givet ett tillstånd och handling kan prediktera nästa tillstånd. Utifrån denna handlingmodell kan den sedan skapa en optimal reaktiv policy. Viktigt att notera här är att ingen planering behöver göras mellan det att agenten observerar ett tillstånd och dess respons Den övergripande algoritmen för Dyna ser ut som följande:

1. Observera världen och välj reaktivt en handling
2. Observera den resulterade belöningen samt det nya tillståndet
3. Applicera självinlärning på denna erfarenhet
4. Uppdatera handlingsmodellen baserat på denna erfarenhet
5. Repetera k gånger:
   (a) Välj ett hypotetiskt tillstånd och handling
   (b) Prediktera den resulterade belöningen samt det nya tillståndet genom användning av handlingsmodellen
   (c) Applicera självinlärning på denna hypotetiska erfarenhet

Ett högre värde på k resulterar i bättre utnyttjande av observationen men resulterar i fler beräkningssteg.

Det kanske mest intressanta aspekten av Dyna är idéen att planering är att genomföra tankeexperiment utifrån det som är känt angående hur världen fungerar. Det vill säga att den försöker använda sin interna modell för att simulera resultat av handlingar i olika tillstånd och på så sätt försöka få fram vad som är bra och mindre bra[8].

3.5 Prioritized Sweeping

Prioritized Sweeping är ytterligare en algoritm inom TD familjen av självinlärningsalgoritmer. Algoritmen skiljer sig i det att den använder sig av tidigare erfarenhet både för att prioritera vad som troligtvis behöver uppdateras mest och för att styra utforskningen. Det innebär alltså att Prioritized Sweeping kombinerar den typiska TD-tekniken med inkrementell inlärning med mer klassiska metoder som utnyttjar sina observationer till en mycket större grad[6].

Algoritmen är på så sätt väldigt lik Dyna, bortsett från att de värden som lagras är förknippade med ett tillstånd och inte ett tillstånd-handlingspar men ännu viktigare är att uppdateringar inte längre väljs slumpmässigt. Istället för att uppdatera k slumpmässiga tillstånd-handlingspar väljer Prioritized Sweeping de k tillstånd med högst prioritet. Uppdateringen för ett givet tillstånd s, sker enligt:

1. Lagra det gamla värdet: $V_{old} \leftarrow V(s)$
2. Uppdatera värdet:
   $V(s) \leftarrow \max_a(R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s'))$

3. Nollställ prioritet för tillståndet
4. Beräkna värdesdifferensen: $\Delta \leftarrow |V_{old} - V(s)|$
5. Använd Δ för att uppdatera prioriteringarna av föregångna tillstånd

Vi behöver alltså utöka vår modell dels för att hålla reda på alla föregångare för varje tillstånd och dels för att hålla reda på prioriteten för varje tillstånd, där begynnelsevärdet är 0 [4].

4. RESULTAT

Algoritmerna som finns beskrivna ovan skiljer sig åt i flera aspekter, många av vilka redan har tagits upp. Det finns dock vissa skillnader och egenskaper som behöver diskuteras mer utförligt. En sådan egenskap är att Q-learning är modellfri. Det innebär att man inte behöver en fördefinierad modell för att genomföra inlärning. Samtidigt så finns det även en fördel med att använda en algoritm som använder sig av en modell då det ofta resulterar i att inlärningen konvergerar snabbare. Det krävs alltså färre observationer för att nå en optimal policy[4]. Intressant nog så är ingen av algoritmerna som undersöks här rent modellbaserade. Men både Dyna och Prioritized Sweeping skiljer sig från Q-learning i det avseendet att de försöker kombinera fördelarna av både modellfria och modellbaserade tillvägagångsätt genom att lära sig en övergångsmodell (*transition model*) och belöningsmodell parallellt med modellfri självinlärning. Denna modell kan sedan användas för att nå mer effektiv inlärning. [10]. Vi kan se detta i ett experiment av Kaebling et al.[4] där de jämför hur många observationer som krävs för att nå en optimal policy:

	Steg för konvergens	Steg för konvergens
Q-learning	531,000	531,000
Dyna	62,000	3,055,000
Prioritized sweeping	28,000	1,010,000

I denna tabell så ser vi att Dyna kräver en storleksordning färre observationer än Q-learning för att nå en optimal policy. Detta kommer dock till en kostnad av att Dyna kräver ungefär sex gånger så många beräkningssteg. Vidare så ser vi att Prioritized Sweeping inte bara kräver hälften så många observationer som Dyna utan använder bara en tredjedel så många beräkningssteg.

Betyder detta att självinlärning är effektivt för väldigt få observationer? Enligt ovan så ser vi att till och med Prioritized Sweeping kräver 28,000 observationer för ett relativt enkelt problem. Trots att det är betydligt bättre än de andra två algoritmerna så är det fortfarande långt ifrån att vara praktiskt för många områden, såsom robotik. Även för GVGAI är det hopplöst opraktiskt då agenten i praktiken måste hinna lära sig att spela ett nytt spel under en och samma spelomgång då evalueringsspelen är nya mot de som finns att träna på.

Självinlärning med väldigt få observationer, eller *Sample efficient reinforcement learning*, är ett forskningsområde av stort intresse inom flera områden, inte minst robotik. Omständigheterna inom mycket av robotiken liknar det som gäller för GVGAI. Vi behöver tekniker som möjliggör att inte bara lära sig från få observationer utan att även göra det i realtid medan agenten kontinuerligt fattar beslut och agerar i sin omgivning. TEXPLORE är en relativt ny algoritm med avsikt att försöka lösa detta problem. TEXPLORE är en modellbaserad algoritm bygger på att lära sig en *random forrest* modell för domänen. En modell som sedan kan användas för att prediktera i tillstånd som den inte redan har besökt. Utforskningen försöker göras på ett sådant sätt så att osäkra, i den mening att lite är känt, och lovande tillstånd prioriteras först[3].

I experimentet av T.Hester och P.Stone så lyckades de att lära en simulerad självkörande bil att kontrollera sin hastighet med relativt få observationer [3]. Resultaten är avsevärt mycket bättre än vad som fås med Dyna, Q-learning och andra algoritmer. De visade att agenten lärde sig uppgiften inom 11 ”episoder”, där 10Hz över 10 sekunder var en episod. Det vill såga 1,100 observationer (och handlingar).

5. SLUTSATSER

Självinlärning är kanske en av de mest intressanta och användbara områdena inom maskininlärning. Att kunna lära sig utifrån erfarenhet utan en tredjepart som styr inlärningen är ett extremt kraftfullt koncept. Men det är inte oproblematiskt. Inte minst så behövs det ofta betydligt större mängder träningsdata för att få inlärningen att konvergera jämfört med andra inlärningsmetoder. Vidare så finner vi problemen med avvägning mellan utforskning och utnyttjande men också hur man effektivt hanterar stora tillståndsrymder, ofta benämnt *Curse of dimensionality*, något som vi inte hinner beröras i denna rapport.

Som vi har sett så finns det ett antal olika algoritmer för självinlärning med olika egenskaper. Q-learning står ut som en enkel, då den är modellfri, men ofta ineffektiv metod när det kommer till inlärningshastigheten. Prioritized Sweeping och Dyna å andra sidan är betydligt mer effektiva när det kommer till antalet observationer men kräver ofta flera storleksordningar fler beräkningssteg, något som kan vara problematiskt i tillämpningar med realtidskrav såsom robotik men även inom GVGAI. Här finns det dock hopp att nya algoritmer såsom TEXPLORE kan förhoppningsvis hantera bättre. Det dock fortfarande oklart hur väl dessa tekniker presterar i mer realistiska problem (inte bara hastighet som faktor) med större tillståndsrymder.

Vi kan sammanfatta detta som att självinlärning för direkt beslutsfattande i dagsläget är väldigt svårt. Inte bara på grund av att inlärningen måste göras under extremt få observationer, utan även på grund av att tillståndsrymden är väldigt stor och att det finns realtidskrav. Däremot så skulle man kunna se att självinlärnings skulle kunna användas för att indirekt styra beslutsfattandet. Ett av de stora problemen för planeringsgrenen i GVGAI är att befintliga tekniker är olika bra för olika typer av spel. Så istället för att styra vilken handling som ska göras i en specifik spelomgång så kanske man kan lära sig vilken planeringsalgoritm som ska användas för vilket typ av spel. Detta innefattar så klart även en klassificieringsdel då ingen information om speletsregler är tillgängliga för agenten. Men antag att vi har lyckats gruppera våra spel, då skulle vi kunna tillämpa självinlärning för att avbilda speltyp mot planeringsalgoritm, så när vi ser ett nytt spel så kan vi välja den bästa planeringsalgoritmen och därmed kraftigt öka prestandan på vår agent. Detta område med välja algoritm kallas för *algoritm selection* eller *hyper heuristic* och är något som främst forskas kring inom andra områden av datavetenskap. Det finns dock

resultat från Mendes et al. som har lyckas att tillämpa denna typ av heuristik för GVGAI [5]. I deras artikel visar de på hur de kan klassificera spelen genom att försöka upptäcka vissa ”features” eller egenskaper. Utifrån detta predikterar de sedan vilken algoritm i deras ”portfolio” som bör prestera bäst och använder denna för att spela[5]. Det finns även en artikel av Bontrager et al[1]. där de försöker att välja rätt planeringsalgoritm. I båda fallen använde det dock inlärning av beslutsträd istället för någon typ av självinlärningsalgoritm och inlärningen skedde *offline*, det vill säga att träning skedde i ett separat steg mot själva användandet.

6. REFERENSER

[1] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius. Matching games and algorithms for general video game playing. 2016.

[2] GVGAI. The general video game ai competition, 2016. [Online; Tillgänglig 2016/10/5].

[3] T. Hester and P. Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[5] A. Mendes, A. Nealen, and J. Togelius. Hyperheuristic general video game playing. *Proceedings of Computational Intelligence and Games (CIG). IEEE*, 2016.

[6] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[7] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[8] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[9] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[10] M. Wiering and M. Van Otterlo. Reinforcement learning. *Adaptation, Learning, and Optimization*, 12, 2012.

[11] Wikipedia. Darpa grand challenge — Wikipedia, the free encyclopedia, 2016. [Online; hämtad 10-October-2016].