Effektiva självlärande algoritmer i reglertekniken

Martin Christiansson

Intensiv forskning inom området artificiell intelligens har på senare tid resulterat i genombrott som möjliggör att datorer och robotar kan utföra uppgifter effektivare än de tidigare kunnat göra. Ett exempel på detta är utvecklingen av neurala nätverk och olika tekniker för att träna dessa. Neurala nätverk är funktionsapproximatorer och kan därmed användas bland annat för att modulera komplexa flervariabla olinjära funktioner som är svåra att beskriva analytiskt. Genom att träna neurala nätverk i en teknik som kallas Reinforcement-learning kan dessa algoritmer lära sig att till exempel spela olika typer av spel, och de kan uppnå övermänsklig skicklighet inom bara några timmars träning, vilket jämfört med en människa som kanske övat hela sitt liv och som ändå ej kan uppnå samma skicklighet. Dessa algoritmer lär sig helt utan någon förkunskap eller förprogrammering av uppgiften. Oändliga möjligheter ligger nu runt hörnet då dessa självlärande algoritmer kan göra allt fler saker, och även uppgifter i verkliga livet som till exempel köra bilar, flyga flygplan mm. Det finns en del viktiga faktorer som påverkar inlärningen av dessa system och algoritmer, och dessa är därför viktiga att förstå för att kunna effektivisera inlärningen. Inlärningseffektiviteten beror till stor del på hur systemen uppdateras med ny information. I denna artikel beskrivs de olika metoder som algoritmerna använder sig av för en snabb och robust inlärning. Artikeln är en sammanfattning av examenarbetet: Reinforcement Learning-Intelligent Weighting of Monte-Carlo and Temporal-Differences[1]. Metoden går ut på att estimera osäkerheter inom systemet som sedan används till att viktia mellan två olika uppdateringsmetoder.

Självlärande regleralgoritmer idag

Företaget DeepMind lyckades för några år sedan skapa generell artificiell intelligens, då de applicerade neurala nätverk på en teknik som kallas Reinforcement-Learning. De tränade neurala nätverk att spela gamla Atari spel genom att mata in de fyra senaste bilderna av spelet i varje tidssteg samtidigt som algoritmen räknade poängen den erhållit. Algoritmen lärde sig på egen hand vilka drag som genererade mest poäng under spelets gång. Algoritmen räknade hur mycket poäng den erhöll för varje agerande i respektive tillstånd i spelet. Efter det gjorde den en estimering av hur mycket mer poäng den skulle få från resten av

[1]TFRT-6072. Department of Automatic Control. http://www.control.lth.se/

spelrundan med utgångsläget av detta tillståndet. Denna information användes till att uppdatera en så kallad värdefunktion, ofta benämnd Q funktionen. Denna värdefunktion är en funktion som ger ett skalärt värde för varje tillstånd i spelet och är ett mått på hur bra det är i att vara i detta tillstånd. Problemet med denna algoritm är att den introducerar störningar i systemet på grund av att det görs estimeringar under inlärningsprocessen. Ett sätt att undvika detta är genom att låta algoritmen spela klart spelet innan värdefunktionen uppdateras, så kallad Monte-Carlo learning. På detta sätt introduceras inga störningar, men dock så blir algoritmen ineffektiv då det blir stora variationer i poängen den lyckas samla på sig för varje spelrunda. Detta beror på att möjligheterna snabbt förgrenar under spelets gång, och att spelet inte alltid slutar med samma poäng-utgång. Båda dessa metoder, Q-learning samt Monte-Carlo learning, har alltså både fördelar och nackdelar, och beroende på situationen ifråga och hur mycket algoritmen vet om spelet kan de vara bättre att använda den ena framför den andra. $TD(\lambda)$ är ett tredje sätt att uppdatera värdefunktionen på och är en metod som blandar de andra två beskrivna algoritmerna. Genom en parameter λ i intervallet $[0, 1]$ kan de båda blandas i olika grad. Med $\lambda = 0$ används enbart $Q-$learning algoritmen, och med $\lambda = 1$ enbart Monte-Carlo learning algoritmen. λ-värde där emellan använder en mix av de båda metoderna. I examensarbetet utvecklas ytterligare en uppdateringsmetod. Denna metod kan variera värdet på λ när algoritmen kör. Värdet på λ bestäms utifrån systemets varians som är ett approximativt mått på hur väl systemet är känt. Metoden är robustare än de metoder som använder sig av fixa λ-värden, samt att den uppnår samma effektivitet som metoden med ett priori känt optimalt λ. Metoden är testad och utvärderad med hjälp av en simulering av en inverterad pendel. Uppgiften för regleralgoritmerna är att balansera pendeln i upprätt läge, se figur 1. Inlärningseffektiviteten för de olika algoritmerna är presenterade i Figur 2.

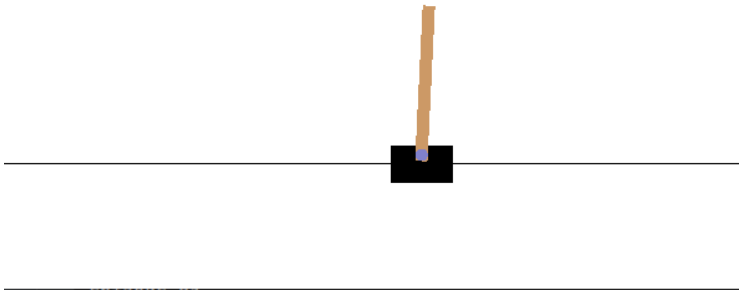
Neurala nätverk

Neurala nätverk används ofta för att imitera hjärnans funktion och inlärningsprocess. Imitationen får dock tas som en förenklad modell av hjärnan, samt med en nypa salt då vetenskapen har en begränsad kundskap och förståelse av hur den mänskliga hjärnan fungerar. Strukturen består av olika lager av beräkningsnoder (neuroner), analogt med hjärnans nervceller, dessa beräkningsnoder är sammankopplade med varandra i ett stort nätverk av så kallade vikter, analogt med nervcellernas dendriter. Antal neuroner varierar och kan röra sig från några få, till flera miljoner beroende på situation och behov. Genom att manipulera förbindelserna (vikterna) mellan neuronerna kan denna lagerstruktur modelera i stort sett vilken matematisk funktion som helst. Vikterna justeras i en uppdateringsprocess och är ett matematiskt optimeringsproblem. Neurala nätverk används här som en funktionapproximator till värdefunktionen.

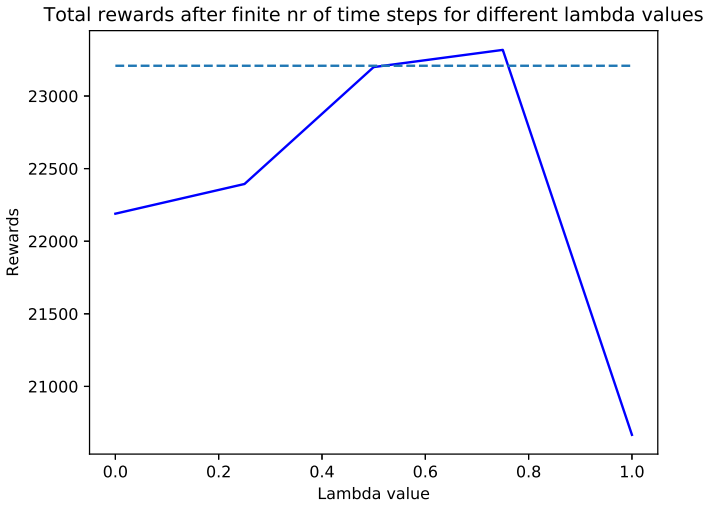
Estimera osäkerheter i systemet med dropout-teknik

Q-funktionen modelleras med ett neuralt nätverk bestående av olika lager av noder. Genom att stokastiskt sätta en viss procent av dessa noder till noll under estimeringen av Q-värdet får man en distribution av olika mindre nätverk som alla estimerar olika värden för Q-funktionen. Genom variansen av denna distribution så estimeras osäkerheten i systemet. Variansen ger en ledtråd

hur väl nätverken stämmer överens med varandra. Faktorer som brus, antal träningsitertioner, konsekvent data och kvalitativ data påverkar variansen och ses därför som ett mått på mycket vi vet om systemet. Med hjälp av denna information kan λ väljas. Om systemets varians är hög så föredras ett högre λ-värde vilket resulterar i uppdateringar av värdefunktionen med mindre vikt på att estimera Q-värden. Om systemets varians är låg så föredras ett lägre λ-värde för att minska påverkan av systemens förgrenade natur.



Figur 1: Inverterad pendel. Ett exempel på spel eller uppgift som används för att utvärdera den framtagna metoden i examensarbetet. Med hjälp av att styra foten (den svarta delen) kan pendeln balanseras i upprätt läge.



Figur 2: Antal poäng algoritmen lyckas få beror på hur väl pendeln är balanserad i upprätt läge. Grafen visar hur väl algoritmen fungerar med fixa värden på λ, och den streckade linjen visar hur väl metoden som väljer λ utefter systemets varians fungerar. Grafen är baserad på medelvärdet av 10 utvärderingar för varje metod.