Een lerende strategie gebruikmakend van Markov modellen om te gebruiken in de herhaalde vorm van bimatrix spelen

Ivar de Haan

July 7, 2018

Abstract

Het doel van het onderzoek was het ontwerpen en testen van een nieuwe lerende strategie voor in herhaalde bimatrix spelen. Er zijn al lerende strategieën gemaakt voor in herhaalde bimatrix spelen maar deze behalen niet altijd optimale resultaten. Hierom hebben wij een nieuwe lerende strategie ontworpen genaamd de markov-learner die gebruik maakt van een Markov model.

In dit onderzoeksverslag wordt eerst uitgelegd wat bimatrix spelen zijn, om daarna specifiek te kijken naar het prisoners dilemma. Hierna worden de twee lerende strategieën uitgelegd. Hierna wordt uitgelegd wat een Markov model is om daarna deze kennis te gebruiken om de zelf ontworpen strategie uit te leggen. Hierna volgt een methodesectie en de bespreking van de gevonden resultaten.

De methode bestond uit het testen van twee al eerder omschreven lerende strategieën, namelijk fictitous play en reinforcement learning, samen met de zelf ontworpen markov-learner strategie. Deze strategieën zijn onderling vergeleken door ze paarsgewijs en met zichzelf de herhaalde vorm van het prisoners dilemma te spelen.

Het gevonden resultaat van het experiment was dat de ontworpen markov-learner een constante, relatief hoge opbrengst bereikte wanneer het tegen zichzelf de herhaalde vorm van het prisoners dilemma speelde in vergelijking met andere tegen elkaar spelende strategie paren. Verder onderzoek zal nodig zijn om de mogelijkheden en het gedrag van de markov-learner volledig te doorgronden.

Contents

1 Inleiding

Speltheorie is een groot en belangrijk wetenschappelijk gebied. In een groot aantal verschillende wetenschappelijke disciplines zoals economie, psychologie en computertechnologie, wordt speltheorie gebruikt. Met speltheorie worden samenwerking en competitieve situaties bestudeerd met behulp van wiskunde. Er worden belangrijke keuzeproblemen mee bekeken, zoals wat de beste manier is om iets te veilen, hoe oorlog te voeren, en hoe een winst te verdelen. Er zijn op het moment elf Nobelprijzen in de economie weggegeven voor ontwikkelingen in de speltheorie.

Een belangrijk deelgebied binnen speltheorie zijn twee speler herhaalde spelen. Bij twee speler herhaalde spelen spelen twee spelers meerdere keren hetzelfde spel. Een spel bestaat er hierbij uit dat allebei de spelers een keuze maken uit een verzameling acties die zij kunnen spelen. Iedere actie van een speler heeft in combinatie met een actie van de tegenstander een bepaalde van de spelers afhankelijke opbrengst. Het doel van de spelers is hun opbrengst te maximaliseren.
Het interessante bij herhaalde spelen is dat de spelers hun keuzes kunnen laten afhangen van wat de tegenstanders in hun voorgaande beurten hebben gedaan. Om dit soort herhaalde spelen te spelen zijn er verschillende lerende algoritmes bedacht zoals fictitious play [3] en algoritmes gebaseerd op reinforcement learning [6].

Het probleem echter is dat deze lerende strategieën soms acties spelen die voor beide spelers minder winstgevend zijn dan andere acties waarbij alle spelers er op vooruit zouden gaan. Dit probleem kan ontstaan doordat het bij het spelen van een spel altijd het beste is om een bepaalde actie te doen, maar dit ertoe leidt dat als beide spelers dit doen, dit voor beide spelers tot lagere opbrengsten leidt.

Er zijn wel lerende strategieën bedacht zoals fictitious play [3] en algoritmes gebaseerd op reinforcement learning [6], maar deze lossen het probleem niet op.
Wij hebben een nieuw lerende strategie bedacht genaamd de markov-learner die gebruik maakt van Markovmodellen. Deze lerende strategie is ontworpen om acties te spelen in een twee speler herhaald spel waarbij hogere opbrengsten worden behaald dan door andere lerende strategieën.

De markov-learner hebben we laten spelen tegen twee andere lerende strategieën en vergeleken met deze twee andere lerende strategieen. Deze twee lerende strategieën waren fictitious play en een vorm reinforcement learning. Het gespeelde spel was het twee speler spel het prisoners dilemma. Uit het experiment kwam naar voren dat de markov-learner als die tegen zichzelf speelde steeds hetzelfde scoorde en dat dit beter was dan de beste actie kiezen voor een los spel.

De tekst begint met een inleiding in bimatrix spelen. De met dit hoofd-

stuk opgedane kennis wordt daarna gebruikt om in het volgende hoofdstuk het spel uit het experiment, het prisoners dillemma, te bestuderen. Na dit hoofdstuk worden de twee lerende algoritmes uitgelegd die werden bekeken met het experiment. Hierna volgt een korte uitleg van Markovmodellen om daarna de bedachte strategie, de markov-learner, te beschrijven. Met alle nodige kennis uitgelegd benoemen we hierna de onderzoeksvraag. Hierop volgt een beschrijving van het experiment in de methodesectie, waarna de resultaten uit het experiment zullen worden besproken. De tekst wordt afgesloten met een korte discussie en conclusie sectie.

2 Bimatrix spelen

In deze tekst bekijken we een competitief soort spel uit de speltheorie: het tweepersoons spel met volledige informatie. Deze spelen worden ook wel bimatrix spelen genoemd. Bij zulke spelen zijn er twee spelers, een rijspeler A en een kolomspeler B. Deze twee spelers kiezen allebei voordat het spel gespeeld wordt een bepaalde actie die zij willen spelen. De actieruimte van speler A geven we aan met Δ^A en de actieruimte van speler B geven we aan met Δ^B. Ook hebben allebei de spelers een payoff functie. Deze functie geeft een opbrengst terug die de speler van de payoff functie krijgt voor ieder paar van een $a \in \Delta^A$ en een $b \in \Delta^B$. Deze payoff functies geven we weer als $Pay_A(x, y)$ en $Pay_B(x, y)$ waarbij $x \in \Delta^A$ en $y \in \Delta^B$. Deze spellen zijn competitief omdat iedere speler als doel heeft zijn eigen winst te maximaliseren. Een oplossing van dit soort spel bestaat uit een paar van acties (a,b) waarbij $a \in \Delta^A$ en $b \in \Delta^B$. De spellen zijn met volledige informatie omdat beide spelers alle mogelijke informatie van het spel hebben. Zij kennen dus de keuzeruimtes van iedere speler en de bijbehorende payoff functies.
Zoals eerder gezegd worden dit soort spellen ook wel bimatrix spellen genoemd. Dit komt omdat het handig is om de payoffs in een twee dimensionale matrix weer te geven. Een voorbeeld van zo een matrix is:

		Kolomspeelster B		
		$b1$	$b2$	$b3$
Rijspeler A	$a1$	$(5, 2)$	$(1, 3)$	$(0, 5)$
	$a2$	$(0, 0)$	$(0, 3)$	$(9, 4)$
	$a3$	$(1, 2)$	$(0, 0)$	$(2, 3)$

Hierbij is de linker waarde in een koppeltje steeds de payoff die de rijspeler krijgt en de rechter waarde is de payoff die de kolomspeler krijgt als de acties bij de desbetreffende rij en kolom worden gespeeld. Ter illustratie: Bij dit

voorbeeld geldt dat $\Delta^A = \{a1, a2, a3\}$ en $\Delta^B = \{b1, b2, b3\}$. Ook geldt bij het voorbeeld $Pay_A(a2, b3) = 9$ en $Pay_B(a2, b3) = 4$.

We bespreken nu enkele belangrijke begrippen binnen de speltheorie.
Een belangrijk begrip binnen de speltheorie is het Nash-equilibrium. In een Nash-equilibrium is bij een oplossing iedere speler zijn strategie de beste strategie gegeven de strategieën van de andere spelers. In andere woorden wil dit zeggen dat bij een Nash-equilibrium geen enkele speler zijn strategie kan veranderen zodanig dat de verwachte payoff voor die speler hoger wordt.
In ons voorbeeld betekend dit dat een Nash-equilibria (a2,b3) is. Want gegeven dat speler B b3 speelt kan de rijspeler niets beters spelen dan a2. Ook is het zo dat gegeven dat de rijspeler a2 speelt, de kolomspelers niets beters kan spelen dan b3. Een voorbeeld dat geen Nash-equilibrium is, is (a2,b2). Gegeven dat de kolomspeler b2 speelt is het immers beter voor de rijspeler om a1 te spelen. Een bimatrix spel kan vele Nash-equilibria bezitten.

Een ander belangrijk begrip zijn pareto-optimale oplossingen. Een oplossing is pareto-optimaal als geldt dat er geen verzameling strategieën is zodanig dat tenminste 1 speler er op vooruitgaat en de rest van de spelers er niet op achteruit gaat.
In het voorbeeld is (a2,b3) een pareto-optimale oplossing. Tegelijkertijd is (a3,b1) geen pareto-optimale oplossing omdat de payoff voor tenminste één speler hoger is en de payoff voor geen enkele speler lager bij de oplossing (a3,b2).

Tot nu toe hebben we bimatrix spelen bekeken als one-shot games. Dit wil zeggen dat beide spelers het spel eenmaal speelden. Bimatrix spelen kunnen echter ook herhaald worden gespeeld. Dit noemen we een herhaald spel of repeated game. Een strategie bij een herhaald spel kan veel ingewikkelder zijn onder andere omdat de spelers hun spelkeuze kunnen laten afhangen van wat de tegenstander in het verleden heeft gespeeld. Een strategie zou kunnen zijn om altijd te spelen wat de tegenstander de allereerste game heeft gespeeld.

We hebben in dit hoofdstuk kennisgemaakt met de bimatrix spelen. We hebben dit aan de hand van een voorbeeld gedaan. In het volgende hoofdstuk bekijken we het bimatrix spel genaamd het prisoners dilemma. Het prisoners dilemma zal gebruikt worden in het onderzoek.

3 Het prisoners dilemma

Het prisoners dilemma is één van de bekendste spelen uit de speltheorie.
Het verhaal wat vaak bij dit spel wordt verteld is als volgt:
Twee criminelen A en B worden opgepakt na het plegen van een misdaad. De politie heeft echter niet genoeg bewijs om de criminelen op te sluiten voor de misdaad. Wel hebben de criminelen allebei nog openstaande geldboetes waarvoor de politie de criminelen kort kan opsluiten. Bij het verhoor krijgen allebei

de criminelen de mogelijkheid om de ander te verraden, of niet te verraden. De criminelen krijgen de deal aangeboden dat als ze de andere crimineel verraden dat dan hun openstaande geldboetes worden kwijtgescholden. De volgende vier situaties kunnen nu plaatsvinden:

- Allebei de criminelen verraden elkaar niet:
  In deze situatie krijgen allebei de criminelen een kleine straf vanwege de openstaande boetes, maar er is dan niet genoeg bewijs om de criminelen langdurig op te sluiten.

- Allebei de criminelen verraden elkaar:
  In deze situatie worden de geldboetes van criminelen kwijtgescholden, maar moeten wel beide criminelen de gevangenis in. Dit is voor beide criminelen een slechtere situatie dan als ze elkaar niet hadden verraden.

- Crimineel A verraadt crimineel B terwijl crimineel B wel zijn mond houdt:
  In deze situatie krijgt crimineel A helemaal geen straf omdat er geen bewijs is dat hij de misdaad heeft gepleegd en zijn boetes hem worden kwijtgescholden omdat hij crimineel B verraadde. Voor crimineel A is dit de beste situatie. Crimineel B krijgt echter een hele grote straf: hij moet zowel de gevangenis in vanwege de misdaad die hij gepleegd heeft en hij moet ook nog de boetes betalen. Dit is de slechtste situatie voor crimineel B.

- Crimineel B verraadt crimineel A terwijl crimineel A wel zijn mond houdt:
  Dit is dezelfde situatie als de net alleen nu krijgt B geen straf en A de ergste straf.

We kunnen dit spel nu vertalen naar een bimatrixspel. De spelers zijn A en B. Allebei de spelers kunnen twee dingen doen: niet verraden en wel verraden. We duiden deze twee respectievelijk aan met C (van cooperate) en D (van defect). Nu definiëren we de volgende payoff-functie voor speler A:
$Pay_A(C,C) = 3, Pay_A(C,D) = 0, Pay_A(D,C) = 0, Pay_A(D,D) = 1$
En speler B krijgt de volgende paypff functie:
$Pay_B(C,C) = 3, Pay_B(C,D) = 5, Pay_B(D,C) = 0, Pay_A(D,D) = 1$

Met deze verdeling wordt voldaan aan regels die we hebben gezien bij de vier situaties. Want (C,C) is voor beide spelers beter dan (D,D). Ook is (D,C) beter voor de rijspeler dan (C,C) en slechter voor de kolomspeler dan (D,D). En we zien dat (C,D) beter is voor de kolomspeler dan (C,C), en slechter voor de rijspeler dan (D,D).
De bijbehorende gamematrix is:

		Speler B	
		C	D
Speler A	C	$(3,3)$	$(0,5)$
	D	$(5,0)$	$(1,1)$

We bezien vanaf nu het spel zoals het in de gamematrix wordt weergegeven. We zien dat voor de rijspeler A het altijd beter is om D te spelen, ongeacht wat de kolomspeler B speelt. Want $Pay_A(D,C) > Pay_A(C,C)$ want $5 > 3$ en $Pay_A(D,D) > Pay_A(C,D)$ want $1 > 0$. Zo zien we ook dat voor de kolomspeler B het altijd beter is om D te spelen, ongeacht wat de rijspeler A speelt. Want $Pay_B(C,D) > Pay_B(C,C)$ want $5 > 3$ en $Pay_B(D,D) > Pay_B(D,C)$ want $1 > 0$.
Het enige Nash-equilibria is dus (D,D) want dan kunnen allebei de spelers geen betere actie spelen.

Nu bekijken we of de ontmoeting (D,D) ook pareto-optimaal is. Een ontmoeting is pareto-optimaal als er geen andere ontmoeting is waarbij alle spelers er op vooruitgaan.
We zien dat Pay_A(D,D) $= 1$ en Pay_B(D,D) $= 1$. Maar Pay_A(C,C) $= 3$ en Pay_B(C,C) $= 3$. Dus is (C,C) beter dan (D,D) voor beide spelers. En dus is (D,D) niet pareto-optimaal.

Voor beide spelers is het dus het beste om, ongeacht de zet van de tegenstander, D te spelen. Dit is het Nash Equilibrium. Maar tegelijkertijd is het voor beide spelers beter als beide spelers C spelen. Het dilemma in het prisoners dilemma is dat beide spelers logischerwijs D zullen spelen, terwijl ze beter af zouden zijn als ze beide C zouden spelen.

We hebben tot nu toe naar het prisoners dilemma gekeken als one-shot game. We gaan nu het prisoners dilemma bekijken als herhaald spel.
Als herhaald spel zijn er veel meer strategieën mogelijk voor beide spelers. Zo kan een speler altijd C of D spelen. Ook zijn er ingewikkeldere strategieeën mogelijk zoals de tactiek die bekend staat als Grim Reaper : speel C als je tegenstander nooit D heeft gespeeld, en speel anders D. Tegen zichzelf en een speler die altijd C speelt zal deze tactiek altijd C spelen, tegen een strategie als all D na de eerste zet elke beurt alsnog 1 in plaats van 0.
Een andere bekende strategie in de herhaalde vorm van het prisoners dilemma is Tit for tat. Tit for tat speelt als eerste actie C, om in alle spelen na het eerste de actie te spelen die de tegenstander in het voorgaande spel speelde. Tit for tat speelt dus tegen spelers die bereid zijn C te spelen ook C, maar tegen spelers die D spelen, ook D.

Nu is een strategie als tit for tat echter erg simpel. Zo werkt tit for tat al niet meer in andere spelen dan het prisoners dilemma. Het zou beter zijn om een lerende strategie te hebben die zich kan aanpassen aan de strategie van de tegenstander gezien wat de matrix is die gespeeld wordt. We bekijken enkele van deze strategieën in het volgende hoofdstuk.

4 Twee lerende strategieën

4.1 fictitious play

Fictitious play is een lerende strategie voor het eerst beschreven door Brown[3] in 1951. Fictitious play is vooral goed voor het vinden van Nash-equilibria door het tegen zichzelf te laten spelen. Er is onder andere bewezen dat als twee fictious play strategieën tegenelkaar spelen deze met hun acties convergeren als het spel een eindig aantal strategieën heeft en het spel een zero-sum game is.[4] Dit laatste betekend dat voor iedere oplossing (a,b) geldt dat $Pay_A(a,b) = -Pay_B(a,b)$
De fictitious play strategie die geïmplementeerd werd in het experiment was gemaakt voor een bimatrix-spel. Hieronder staat precies hoe het geïmplementeerd is. Het fictitious play algoritme heeft een actieruime genoteerd met Δ^A terwijl de tegenstander B een actieruimte heeft genoteerd met Δ^B.

Fictious play houdt bij hoe vaak iedere actie $b \in \Delta^B$ door de tegenstander gespeeld was. Dit noteren we als $c(b)$. Voor de eerste game plaatsvond werd ingesteld dat $c(b) = 0$ voor alle $b \in \Delta^B$
Ieder spel dat fictitious play een actie moet kiezen werd het volgende gedaan:
Eerst berekent fictious play voor iedere mogelijke actie b van de tegenstander de verwachte kans dat deze mogelijke actie van de tegenstander in een willekeurig spel gekozen wordt. De verwachte kans p(b) dat een actie b $\in \Delta^B$ gekozen wordt is gelijk aan het aantal keer dat deze actie b gekozen is in voorgaande games in de gespeelde speelserie, gedeeld door het totale aantal gespeelde acties van tegenstander B in voorgaande spellen in de afgelopen speelserie. Wanneer

er nog geen spellen zijn gespeeld, en het totale aantal gespeelde acties dus 0 is, wordt geïnstantieerd voor iedere $b \in \Delta^B$ dat $p(b) = \frac{1}{|\Delta^B|}$

Als berekent is voor iedere mogelijke actie b van de tegenstander B wat de verwachte kans p(b) is dat deze actie gespeeld wordt in een game gaat fictitious play berekenen wat de verwachte opbrengsten zijn van de mogelijk zelf te kiezen acties. De verwachte opbrengst $U(a)$ waarbij $a \in \Delta^A$, is gelijk aan de sommatie van de kans dat een tegenstander een actie b speelt keer de opbrengst als fictitious play a speelt en de tegenstander speelt B, voor iedere b $\in \Delta^B$. Oftewel $U_A(a) = \sum_{b \in \Delta^B} Pay_A(a, b) * p(b)$ waarbij $a \in \Delta^A$ als het fictious play algoritme de rijspeler was, en $U_A(a) = \sum_{b \in \Delta^B} Pay_A(b, a) * p(b)$ waarbij $a \in \Delta^A$ als het fictious play algoritme de kolomspeler was.

Als fictious play voor iedere actie $a \in \Delta^A$ de verwachte waarde $U_A(a)$ heeft berekent bepaalt fictious play de actie a* waarvoor de verwachte waarde $U_A(a)$ het hoogst is. Deze actie a* speelt fictious play vervolgens. Als er meerdere acties zijn met dezelfde hoogste waarde, dan speelt fictitious play een willekeurige actie van deze acties.
Als laatste, gegeven de actie b* die de tegenstander dan wil spelen, telde fictitious play bij $c(b*)$ één op omdat de actie b* nu één keer meer gespeeld was door de tegenstander.

4.2 reinforcement learning

De reinforcement learning strategie, die geïmplementeerd werd voor het experiment werkt als volgt.

De reinforcement learning strategie, met kiesbare acties Δ^A, houdt ten alle tijden bij hoeveel iedere actie $a \in Delta^A$ in totaal over alle gespeelde games in de huidige spelserie heeft opgebracht. Dit getal noemen we $U(a)$.
Voordat het eerste spel van een spelserie gespeeld wordt voor iedere actie $a \in \Delta^A$ geïnstantieerd dat $U(a) = 1$.
Iedere game dat het reinforcement algoritme een actie moest kiezen doet het willekeurig één van de volgende twee dingen:

- Met 90% kans, exploiteren:
  Als de reinforcement learning strategie ging exploiteren dan koos het de actie a* die in totaal het meest opgeleverd had over alle games in de afgelopen spelserie. Dus die $a \in \Delta^A$ waarvoor $U(a)$ het hoogst was. Hadden meerdere acties dezelfde totale opbrengst dan werd er via een uniforme verdeling willekeurig uit de acties die deze hoogste totale opbrengst hadden een a* gekozen;

- Met 10% kans, exploreren:
  Bij exploreren kiest het algoritme een willekeurige actie $a \in \Delta^A$ om te spelen waarbij iedere actie $a* \in \Delta^A$ gekozen kon worden met een kans van $\frac{1}{|\Delta^A|}$.

Na geëxploiteerd of geëxploreerd te hebben gaf het reinforcement algoritme dus een actie a* terug die gespeeld ging worden. Als laatste werd, gegeven de actie b* die de tegenstander koos, de opbrengst die dit actiepaar voor het reinforcement learning algoritme opleverde, bij U(a*) opgeteld. Oftewel $U(a*)$ werd $U(a*) + Pay(a*, b*)$ als het reinforcement learning algoritme de rijspeler was, 'of $U(a*)$ werd $U(a*)+Pay(b*, a*)$ als het reinforcement learning algoritme de kolomspeler was.

5 De ontworpen lerende strategie

Eerst zal worden uitgelegd wat Markov modellen zijn want deze worden gebruikt door de lerende strategie die wij ontworpen hebben.

5.1 Markov modellen

Markov-modellen zijn een manier om waargenomen kansvariabelen tot een model te verwerken. Het belangrijke kenmerk van Markov modellen is dat er uit wordt gegaan van de Markov assumptie: Er wordt verondersteld dat een voorspelling van de toekomstige staat alleen afhankelijk is van de huidige staat, en dus niet van de staten die tot de huidige staat geleid hebben.
Een voorbeeld hiervan is de aanname dat de kans dat het morgen regent alleen afhankelijk is van het feit of het vandaag wel of niet heeft geregend. Met een markov model zou men dan de keren tellen dat het regende een dag nadat het de dag ervoor droog was gebleven, hoe vaak het niet regende als de dag ervoor het droog was gebleven, hoe vaak het regende een dag nadat het de dag ervoor regende, hoe vaak het niet regende wanneer de dag ervoor het had geregend.
Stel bijvoorbeeld dat na 100 dagen dit bijhouden de volgende tabel qua metingen zou zijn gevonden:

		Gister	
		Regen	*Droog*
Vandaag	*Regen*	20	10
	Droog	40	30

Nu stellen we ons Markov model op. Aangezien het 60 keer de dag ervoor geregend had en het 20 keer na een regenachtige dag droog was nemen we aan voor ons model $P(VandaagRegen|GisterRegen) = \frac{20}{60} = \frac{1}{3}$. Aangezien het 40 keer droog was na een regenachtige dag nemen we aan $P(VandaagDroog|Gisterregen) = \frac{40}{60} = \frac{2}{3}$. En zo nemen we aan voor na droge dagen: $P(Vandaagregen|Gisterdroog) = \frac{10}{10+30} = \frac{1}{4}$, en $P(Vandaagdroog|Gisterdroog) = \frac{30}{10+30} = \frac{3}{4}$.
Met dit model kunnen we nu voorspellingen doen over nog niet plaatsgevonden evenementen. Regent het bijvoorbeeld vandaag en willen we weten wat

de kans is dat het morgen regent, dan kijken we in ons model en dan vinden we dat $P(VandaagRegen|GisterRegen) = \frac{1}{3}$ en dus nemen we aan dat de kans dat het morgen regent $\frac{1}{3}$ is. Maar we kunnen ook verder vooruitkijken. Regent het bijvoorbeeld vandaag en willen we weten wat de kans is dat het morgen droog blijft en overmorgen regent, dan hoeven we slechts te berekenen $P*(VandaagDroog|GisterRegen)*P(VandaagRegen|GisterDroog) = \frac{1}{3}*\frac{1}{4} = \frac{1}{12}$.

In een Markov model kan er ook vanuit worden gegaan dat een toekomstige staat niet alleen afhankelijk is van de laatste staat, maar ook die ervoor, of die daar dan ook weer voor, enzovoort. In het regen-voorbeeld betekend dit dat het model ook zou kunnen berekenen wat de kans is dat het vandaag regent afhankelijk van een tupeltje van of het gister regende en of het eergister regende. Een kans zou er bijvoorbeeld uitzien als $P(VandaagDroog|GisterRegen, EergisterDroog)$.

5.2 De markov-learner

Een groot probleem van de fictitious play strategie en de reinforcement learning strategie is dat deze strategieën niet bij hun actiebeslissing rekening houden, met wat hun actie in een bepaalde game in een spelserie voor reactie uitlokt bij de tegenstander. De algoritmes houden geen rekening met de toekomst.
Voor het experiment hebben we een nieuw lerend algoritme ontwikkeld voor het gebruik in bimatrix repeated games. We noemen deze de markov-learner omdat hij gebruikt maakt van een Markov model.
Het doel was een lerende strategie te maken die in herhaalde spelen wel rekening kon houden met wat voor effect een actie zou hebben op de toekomstige acties van de tegenstander. We hebben dit op twee manieren proberen te realiseren:
Ten eerste laat de markov-learner niet alleen zijn actiekeuzes afhangen van wat de verwachte opbrengst is de aankomende game, maar ook wat voor verwachte opbrengst de actie in het volgende spel zal opleveren. Het idee hierachter is dat het beter is om te kijken naar de opbrengst op de langere termijn, dan de opbrengst op één moment.
Ten tweede kijkt de markov-learner naar de acties gespeeld door zowel hemzelf als de tegenstander de afgelopen twee games. Er is hier voor gekozen zodat de markov-learner informatie van zijn tegenstander kan gebruiken, over wat deze mogelijk wil gaan spelen de volgende zetten.
Deze twee ideeën hebben we geïmplementeerd door gebruik te maken van een Markovmodel dat bijhoudt wat de verwachte kansen zijn dat de tegenstander bepaalde acties speelt gegeven de afgelopen 2 games gespeelde acties van zowel de markov-learner zelf, als de acties van de tegenstander. Het model bestond dus uit kansen van de vorm
P(actie Tegenstander tijdstip t|((actie markov-learner op t-2 , actie tegenstander op t-2),(actie markov-learner op t-1 , actie tegenstander op t-1)).

Hieronder volgt eerst een precieze beschrijving hoe de markov-learner precies werkt, om daarna aan te geven waarom er gekozen is voor deze instellingen.

De markov-learner werkt als volgt:
De markov-learner hield een markov model bij. Voor ieder mogelijk spelverloop van 2 games hield de markov learner bij hoevaak na deze vorm van spelverloop iedere actie was gespeeld in een spelserie. Een spelverloop zullen we noteren als ((a,b),(a′,b′)). Het eerste tupeltje hierin zijn de acties die de markov-learner en de tegenstander speelde op tijdstip t, en het tweede tupeltje zijn de acties die de markov-learner en de tegenstander in de game erna op tijdstip t+1 speelde. Specifieker is a hier de actie die de markov-learner speelde op tijdstip t en b hier de actie die de tegenstander speelde op tijdstip t. a′ is actie die de markov-learner speelde de game erna op tijdstip t+1 en b′ is de actie die de tegenstander speelde op tijdstip t+1. Een spelverloop bijvoorbeeld was dat op tijdstip t de markov-learner actie g had gespeeld en de tegenstander v, en op tijdstip t+1 de markov-speler actie h gespeeld en de tegenstander actie w, het spelverloop was dan ((g,v),(h,w)).
Voor ieder mogelijk spelverloop hield de markov-learner bij hoevaak iedere mogelijke actie $b \in \Delta^b$ van de tegenstander hier op volgde. Deze aantallen geven we aan met C((A,B),(A′,B′)).
Kwam bijvoorbeeld in een spelserie het spelverloop ((g,v),(h,w)) 8 keer voor, waarbij na dit spelverloop de game erna de tegenstander op tijdstip t+2 5 keer de actie v speelde en 3 keer de actie w speelde, dan als $\Delta^B = \{v, w\}$, dan was C((g,v),(h,w)) = (5,3).

Om een actie te kiezen voor een game op tijdstip t doet de markov-learner het volgende:

- stap 0:
  Als er nog geen twee games zijn gespeeld in de huidige spelserie gaat de markov-speler er van uit dat ieder spelverloop de aankomende 2 games met een evengrote kans kan gebeuren. Hierna gaat de markov-learner door met stap 4. Als er wel 2 games zijn gespeeld in de huidige spelserie begint de markov-learner bij stap 1.

- Stap 1:
  De markov-learner kijkt wat het spelverloop is geweest over de laatst gespeelde 2 games. Dit spelverloop noemen we S^{t-2} omdat het begon op tijdstip t-2.

- Stap 2:
  Hierna berekent de markov-learner voor ieder mogelijk spelverloop de kans dat deze plaatsvindt op S^{t-1} voor iedere actie die de markov-learner kan spelen. Bijvoorbeeld: Stel dat $\Delta^A = \{a,\ a'\}$ en $\Delta^B = \{b,\ b'\}$ en $S^{t-2} = ((a, b), (a, b'))$. Dan berekent de markov-learner de kansen dat $S^{t-1} =$

$((a,b'),(a,b)), S^{t-1} = ((a,b'),(a,b')), S^{t-1} = ((a,b'),(a',b)), S^{t-1} = ((a,b'),(a',b'))$.
De kans dat $S^{t-1} = ((A,B),(A',B'))$ is gelijk aan het aantal keren dat B' is gezien na het spelverloop S^{t-2} gedeeld door het totale aantal keren dat het spelverloop S^{t-2} is gezien. De kans is dus 0 voor ieder spelverloop dat niet begint met het tuple (A,B), want op tijdstip t-1 staat vast dat de markov-learner A speelde en de tegenstander B speelde. Ook is de kans niet afhankelijk van A' want de tegenstander laat zijn keuze niet afhangen van wat nog niet gebeurt is. Wanneer een spelverloop nog nooit is gezien is de kans op iedere actie die de tegenstander erna kan spelen gelijk aan $\frac{1}{|\Delta^B|}$. Na deze stap heeft de markov-learner dus een verzameling van mogelijke spelverlopen met een bijbehorende kansfunctie die we zullen aanduiden met $P_{t-1}(S^{t-1})$

- Stap 3:
  Hierna berekent de markov-learner voor ieder mogelijk spelverloop de kans dat deze plaatsvindt op S^t. Dit doet de markov-learner door dezelfde berekening uit te voeren als bij stap 2, maar dan voor ieder mogelijk spelverloop voor S^{t-1}. Met alleen dit verschil dat nadat berekent is wat de kans is dat een bepaalde S^t volgt op een bepaalde S^{t-1} de uiteindelijke kans $P_t(S^t)$ gelijk is aan de kans dat S^t volgt op S^{t-1} keer de kans dat S^{t-1} volgt op $S^(t-2)$ oftewel $P_{t-1}(S^{t-1})$. Ter voorbeeld: Als $P_{t-1}(S^{t-1})$ $= 0.5$, en de kans dat S^t volgt op $S^{t-1} = 0.3$ dan $P_t(S^t) = 0.5 * 0.3 = 0.15$. Na deze stap houdt de markov-learner een verzameling van mogelijke spelverlopen op tijdstip t genaamd Ω^T over met een bijbehorende kansfunctie die we zullen aanduiden met $P_t(S^t)$.

- Stap 4:
  De markov-learner berekent de verwachte opbrengst van iedere spelverloop $s \in \Omega^T$ aangeduid door $U(s)$ door de opbrengsten die de markov-learner krijgt bij dit spelverloop te vermenigvuldigen met de kans dat dit spelverloop plaatsvind. Bijvoorbeeld, voor het spelverloop ((a,b),(a′,b′)) geldt dat U(((a,b),(a′,b′))) = (Pay(a,b) + Pay(a′,b′)) * $P_t((a,b),(a',b'))$

- Stap 5:
  De markov-learner berekent voor iedere actie $a \in \Delta^A$ wat de sommatie is van die $s \in \Omega^T$ waarbij geldt dat als actie van de markov-learner, s in het eerste tuple a heeft.
  De markov-learner kiest de actie a* waarvoor deze sommatie het hoogst is. Als er meerdere acties zijn waarvoor deze sommatie de hoogste waarde heeft, dan kiest het algoritme een willekeurige van deze acties.

We hebben gekozen voor deze precies instellingen omdat we wilden bereiken dat het algoritme niet altijd koos voor de beste actie in een losse game, maar ook keek naar wat op de langere termijn meer winst zou opleveren. Er is gekozen om de keuze te laten afhangen van de laatste twee acties van beide spelers zodat het algoritme ingewikkelder patronen kon ontwikkelen en meer informatie heeft over

de tegenstander. Er is gekozen om twee acties vooruit te kijken zodat het algoritme over een langere termijn kan beslissen wat er moet gebeuren. We hebben er voor gekozen om de actie te spelen met de hoogste totale waarde over alle mogelijke spelverlopen, omdat het algoritme nog niet weet wat de tegenstander op tijdstip t gaat spelen. De strategie hadden we ook kunnen instellen zodanig dat het zou kijken wat gegeven een actie a op tijdstip t de beste actie zou zijn op tijdstip t+1, om zodanig dat als de verwachte waarde te zien op tijdstip t voor de actie. Maar dan houden we geen rekening mee met dat een zwakkere actie op t+1 weer een betere payoff kan geven op t+2. Iets wat op t+1 pas bedacht kan worden door de markov-learner.

6 Onderzoeksvraag

We gaan met een experiment onderzoeken hoe de gepresenteerde markov-learner presteert in vergelijking met en in combinatie met andere lerende algoritmes. De onderzoeksvraag is:

Hoe presteert de markov-learner in vergelijking met fictitious play en reinforcement learning in repeated games van het prisoners dilemma?

Om dit te onderzoeken laten we de markov-learner, fictious play algoritme, en reinforcement learning, tegen elkaar het eerder genoemde prisoners dilemma spelen in repeated games. Er is gekozen voor het prisoners dilemma voor de onderzoeksvraag omdat als het prisoners dilemma eenmalig wordt gespeeld de ander verraden (actie D) altijd de beste keuze is. Maar in repeated games waarbij het niet duidelijk is hoelang de game nog doorgaat, kan het interessanter zijn om aan te bieden C te spelen zodat de tegenstander hier ook op in kan gaan. Aangezien het prisoners dilemma een symmetrisch spel is zijn de acties die een rijspeler met strategie A tegen een kolomspeler met strategie B speelt, dezelfde acties als die van de kolomspeler als deze met strategie A tegen een rijspeler speelt die strategie B gebruikt. Het maakt dus niet uit of een speler als rij- of kolomspeler speelt.

7 Methode

Hieronder wordt uitgelegd hoe het gedane experiment kan worden herhaald.
Het experiment bestaat uit het spelen van het eerder beschouwde prisoners dilemma door de drie eerder genoemde lerende strategieën tegen elkaar. De lerende strategien zijn: reinforcement learning, fictitious play en de markov-learner.
Deze strategieën moeten worden geïmplementeerd als beschreven in de voorgaande secties.

Het te spelen herhaalde spel is het prisoners dilemma. De payoffs van dit spel moeten als volgt worden ingesteld:

		Kolomspeelster	
		C	D
Rijspeler	C	$(3,3)$	$(0,5)$
	D	$(5,0)$	$(1,1)$

Laat iedere speelstrategie als rijspeler tegen iedere speelstrategie als kolomspeler, inclusief zichzelf, 1000 spelseries spelen. Iedere spelserie bestaat uit 1000 rondes. Iedere ronde bestaat uit het spelen van één zet door beide spelers.

Voor ieder paar spelers moet het experiment als volgt worden opgesteld:
Allereerst moet er een variabele worden geïnstantieerd met waarde 0. Deze variabele houdt bij wat de totale score is van de rijspeler over het gehele aantal games tegen de kolomspeler.
Laat dan per koppel van rijspeler-kolomspeler 1000 spelseries spelen. Een spelserie begint door allebei de spelers te initiëren zodanig dat ze nog niets weten. Dit betekent dat voor iedere speler, elke zet, C en D, even vaak gespeeld lijkt. Concreet moet worden ingesteld per strategie:

- Dat de markov-learner alle speelvolgordes nog nooit had gezien;
- Dat regret-learning allebei de zetten C en D nul keer had gezien;
- Dat fictitious play allebei de zetten C en D nul keer had gezien;

Laat na de initialisatie bij een spelserie allebei de spelers 1000 rondes spelen. Een ronde bestaat uit de volgende fases:

- Speelfase: Hierbij geven beide spelers de zet door die ze met hun huidige interne model willen spelen.
- Updatefase: Hierbij krijgen de spelers de zet door die hun tegenstander net gespeeld heeft. Hiermee werden de lerende spelers hun modellen geüpdatet.
- Puntentellingsfase: Bij het totaal aantal punten van de rijspeler wordt het aantal punten opgeteld dat de rijspeler zou krijgen gegeven de zetten die de rijspeler en kolomspeler in de laatst gespeelde speelfase teruggaven.

Na 1000 van deze rondes is een spelserie afgelopen. Begin hierna weer een nieuwe spelserie waarbij de spelers weer opnieuw worden geïnitialiseerd en er weer 1000 rondes worden gespeeld.
Zo worden er per paar rijspeler-kolomspeler 1000 van deze spelseries gespeeld. Na deze 1000 spelseries moet het totale aantal punten van de rijspeler gedeeld

worden door het totale aantal rondes, dus door 1000000 (omdat 1000 * 1000 = 1000000). Het getal dat na deling overblijft is de gemiddelde score die de rijspeler heeft gekregen in de speelrondes tegen de kolomspeler.

Laat zo iedere strategie als rijspeler tegen iedere strategie als kolomspeler spelen. Hier houden we de 9 gemiddelde scores van de rijspelers van ieder mogelijk paar rijspeler-kolomspeler aan over. Stop deze 9 gemiddelde scores in een 3x3 2-dimensionale matrix zodanig dat in iedere cel de gemiddelde score staat van een rijspeler tegen een kolomspeler.

8 Resultaten en bespreking resultaten

De door het experiment gecreëerde tabel is:

		Kolomspeelster		
		$Mark$	$Fict$	$Reinf$
Rijspeler	$Mark$	1.21	0.999	1.2024
	$Fict$	1.004	1	1.2019
	$Reinf$	0.9603	0.949	1.5872

In iedere cel staat geschreven wat de totale opbrengst van de rijspeler tegen de kolomspeler over het totale aantal gespeelde games was. We zullen nu ingaan op de verschillende gemeten resultaten. Als we spreken van het resultaat van strategie A tegen strategie B, dan bedoelen we de opbrengst van strategie A als rijspeler toen die tegen strategie B als kolomspeler speelde.

De resultaten van de markov-learner tegen fictitious play en fictitious play tegen de markov-learner, lagen allebei zeer dicht bij 1. Bij nadere inspectie van de gespeelde acties bij beide confrontaties bleek dat fictitious play bij iedere speelserie bij iedere game D speelde, terwijl de markov-learner altijd D speelde, behalve iedere 4^e game van iedere spelserie.

De resultaten van de markov-learner tegen reinforcement learning en reinforcement learning tegen de markov-learner waren respectievelijk 1.2024 en 0.9603. Het begin van de spelseries bij deze beide confrontaties verschilden onderling nog veel door het random element van reinforcement learning. Reinforcement learning ging echter al snel bijna altijd over op alleen D spelen. Waarna de markov-learner snel ook overschakelde op alleen D spelen. Reinforcement learning speelde echter door het random element nog steeds soms C, waardoor het minder dan 1 punt gemiddeld kreeg.

De resultaten van fictitious play tegen reinforcement learning en reinforcement learning tegen fictitious play waren respectievelijk 1.2019 en 0.949. Als we na de spelseries keken zagen we dat fictitious play altijd D speelde. Reinforcement learning speelde ook altijd D want C bracht nu eenmaal niets op en ging dus nooit meer in totaal hebben opgeleverd dan D. Dus bij exploitatie werd altijd D gekozen en alleen bij exploratie werd soms C gespeeld.

Het resultaat van fictitious play tegen fictitious play was precies 1. Fictitious play speelde altijd D, en dus leverde iedere game een waarde van 1 op. De gemiddelde opbrengst was dan ook 1 voor fictitious play tegen fictitious play.

Het resultaat van reinforcement learning tegen reinforcement learning was 1.5872. Er waren drie verschillende soorten spelseries te onderscheiden:

- Beide spelers speelde door het random element en door dit te exploiteren aan het begin van het spel in de eerste zetten C, waarna beide spelers voornamelijk C speelde, behalve als er geëxploreerd werd. De gemiddelde opbrengsten van zulke spelseries lagen rond de 3.
- Beide spelers speelde door het random element en door dit te exploiteren aan het begin van het spel in de eerste zetten D, waarna beide spelers voornamelijk D speelde, behalve als er geëxploreerd werd. De gemiddelde opbrengsten van zulke spelseries lag rond de 1.
- Beide spelers speelde door het random element en door dit te exploiteren aan het begin van het spel in de eerste zetten C en bleven dit een tijdje doen. Op een gegeven moment oversteeg door het exploreren de totale opbrengst van D boven die van C voor een van die spelers waarna de speler D begon te spelen voor de rest van de spelserie met af en toe een random C ertussen. De speler die nog C bleef spelen bleef dit soms de hele spelserie doen, maar schakelde soms ook over op D. De gemiddelde opbrengsten van zulke spelseries lagen tussen 0 (voor de speler die nooit of laat naar D overstapte), en de 5(voor de speler die na een paar keer C overstapte naar D) in.

Het resultaat van de markov-learner tegen de markov-learner was 1.21. Beide markov-learners speelde alle speelserie dezelfde volgorde aan acties. De opbrengst van iedere spelserie was dan ook 1.21. Beide spelers speelden dus ook altijd dezelfde actie. De volgorde leek vrij willekeurig. Vaak kwam het patroon terug dat er 5 of 6 keer het patroon dat er 11 of 12 D's afgewisseld werd met een C, waarna het patroon weer meer willekeurig werd voor even.

Samenvattend zien we dus dat bij alle strategieënparen behalve markov-markov en reinforcement-reinforcement, de gevoerde strategie altijd D spelen wordt na enkele games waardoor de gemiddelde score rond de 1 zit voor beide spelers. Zit reinforcement learning echter in het paar krijgt reinforcement learning wat minder dan 1 punt, omdat door het random exploreren, reinforcement

learning dan af en toe C speelt, terwijl de tegenstander altijd D speelt, en dus iets meer punten krijgt gemiddeld.
We zien dat bij reinforcement learning tegen reinforcement learning de gemiddelde waarde het hoogst is, maar ook de verschillende gemiddeldes over de spelseries het meest uiteen lopen.
We zien dat bij de markov-learner tegen de markov-learner de gemiddelde opbrengst even hoog is bij iedere spelserie. Deze opbrengst is ook vrij veel hoger dan 1. De markov-learners doen dus een strategie samen waarbij ze beide beter af zijn dan als ze alleen maar D speelde en dit doen ze in tegenstelling tot reinforcement learning tegen reinforcement learning constant.
Wij denken dat deze consistentie en het toch hoger scoren dan 1 punt gemiddeld de markov-learner tot een interessante lerende strategie maakt.

9 Discussie en conclusie

We hebben een nieuwe lerende strategie, genaamd de markov-learner, gepresenteerd voor gebruik in herhaalde 2 speler spelen. De strategie maakt gebruik van een Markovmodel en kijkt meerdere acties vooruit. Het algoritme is getest door het te laten spelen, en te vergelijken met met twee andere lerende strategieën, namelijk fictitious play en reinforcement learning. Het spel dat gespeeld werd was het prisoners dilemma. Het interessante aan het prisoners dilemma is dat het altijd het beste is om D te spelen ongeacht wat de tegenstander doet, maar het voor beide spelers beter is als beide spelers C spelen, dan als ze allebei D spelen.
Uit het experiment kwam naar voren dat in een herhaald spel van het prisoners dilemma, de meeste strategieën D wilden gaan spelen, behalve in twee gevallen, namelijk als reinforcement learning tegen reinforcement learning speelde, en als de markov-learner tegen de markov-learner speelde.
Bij reinforcement learning tegen reinforcement learning waren de gemiddelde opbrengsten hoger dan bij de markov-learner tegen de markov-learner, maar deze opbrengsten waren wel erg verschillend per spelserie.
Bij de markov-learner tegen de markov-learner was de opbrengst echter constant bij iedere spelserie, terwijl deze nog steeds beter was dan als beide spelers alleen maar D speelde. Dit lijkt er op te wijzen dat de markov-learner strategie mogelijk gebruikt zou kunnen worden om cosntant hogere opbrengsten te verkrijgen in herhaalde spelen dan met andere lerende strategieën mogelijk is.

Opgemerkt moet worden dat er slechts één game, namelijk het prisoners dilemma, gespeeld is. Om een beter inzicht te krijgen in en van de mogelijkheden van de markov-learner zou deze op meer games moeten worden getest. Ook is de markov-learner slechts vergeleken met enkele andere lerende strategieën. Er zijn nog andere lerende strategieën zoals no-regret learrning en reinforcement learning strategieën met andere parameters.
Er is dus nog veel onderzoek nodig om de mogelijkheden en het gedrag van de markov-learner volledig te doorgronden.

10 Bibliografie

References

[1] Peters, Hans, *Game Theory, a Multi-Leveled Approach*, second edition, 2015. Hoofdstukken 2, 3 en 7.

[2] Langrock, Roland; MacDonald, Iain L.; Zucchini, W, *Hidden Markov Models for Time Series: An Introduction Using R*, 2016. Hoofdstukken 1 en 2.

[3] Brown, George W. *Iterative Solutions of Games by Fictitious Play" In Activity Analysis of Production and Allocation*, 1951.

[4] Robinson, J, *An Iterative Method of Solving a Game*, Annals of Mathematics 54, 296 - 301, 1951.

[5] Fudenberg, Drew; Levine, David K, *The Theory of Learning in Games*, 1998. Hoofdstuk 2.

[6] Peyton Young, H *Strategic Learning and its Limits*, 2004. Hoofdstukken 2 en 6.

11 Appendix: URL naar code en README

De code die gebruikt werd bij het experiment is te vinden op GitHub via de volgende URL:

https://github.com/IvardeHaan/BA-Scriptie-Ivar-de-Haan

De code is geschreven in Python 3.6 en is verdeeld over vijf scripts. Deze scripts kunnen via de URL worden ingezien, gekopieerd en gedownload als zip.

Wanneer u het experiment wilt herhalen kunt u de code als volgt runnen:

- Download de code via bovenstaande link als zip door eerst op het knopje "Clone or Download" en daarna op het knopje "Download ZIP" te drukken;
- Unzip de zipfile met een programma als 7-Zip in een folder;
- Run het bestand "final_test.py" met python 3.6. In windows 7 kan dit gedaan worden door in de command prompt te navigeren naar de directory waar de scripts staan en hier de volgende command aan te roepen: "py final_test.py"

De code bestaat uit de volgende vijf scripts:

- "final_test.py": Hoofdscript van het experiment. Hier kan ingesteld worden hoe vaak iedere strategie tegen iedere andere strategie speelt;
- "speelserie.py": Code die een speelserie tussen twee strategieën laat plaatsvinden. Hier kan worden ingesteld hoeveel games iedere spelserie duurt;
- "markov_learner.py": Bevat de code van de markov-learner strategie.
- "reinforcement_learning.py": Bevat de code van de reinforcement learning strategie;
- "Fictitious.py": Bevat de code van de fictitious play strategie.