Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Andreas Moe Hatlø

Å lære No-Press Diplomacy fra Selvspill

Dyp Reinforcement Learning med Fokus på Samarbeid mellom Agenter

Masteroppgave i Datateknologi
Veileder: Keith L. Downing
Juni 2022

NTNU
Norwegian University of
Science and Technology

Andreas Moe Hatlø

Å lære No-Press Diplomacy fra Selvspill

Dyp Reinforcement Learning med Fokus på Samarbeid mellom Agenter

Masteroppgave i Datateknologi
Veileder: Keith L. Downing
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Norwegian University of
Science and Technology

Sammendrag

I løpet av de siste årene har brettspillet Diplomacy fått økt oppmerksomhet innen forskningsfeltet reinforcement learning. Dette skyldes at Diplomacy har trekk som gjør spillet utfordrende å lære en datamaskin. I Diplomacy spiller opp mot sju spillere mot hverandre, og spillerne gjør trekkene sine samtidig. Kombinert med at hver spiller har mange valgmuligheter for hver runde, gjør dette at antallet mulige felleshandlinger pr. runde kan overstige 10^{64}. Spillet har også svært mange mulige brettkonfigurasjoner.

LOLA er en policy-gradient-algoritme som tar hensyn til at andre agenter lærer. En LOLA-agent forsøker å maksimere sin egen prestasjon ved å se på hvordan endringer i egen policy kan påvirke andre agenters policy. Denne algoritmen har tidligere vist seg å kunne generere samarbeid mellom agenter. A2C er en enklere utgave av policy-gradient-algoritmer, hvor det ikke tas hensyn til at andre agenter kan endre sin policy.

I Diplomacy har spillere som evner å samarbeide en stor fordel. Selv om tidligere forsøk på å lære en agent å spille Diplomacy har vært vellykket, har de ikke innebåret å anvende en algoritme som tidligere har vist seg å kunne generere samarbeid. I denne rapporten blir det derfor testet om LOLA egner seg bedre til å lære en agent å spille Diplomacy enn A2C.

LOLA har blitt testet på originalbrettet til Diplomacy, samt en mindre utgave, kalt “Pure”. Forsøk viser at algoritmen krever for mye ressurser til å kunne anvendes på originalbrettet. Forsøk viser også at en LOLA-agent lærer å spille “Pure”-utgaven av Diplomacy, men presterer dårligere enn en A2C-agent. LOLA-agenten lykkes heller ikke i å bli bedre til å samarbeide med sine motspillere enn A2C-agenten.

Abstract

During the last years the board game Diplomacy's popularity as a research topic within reinforcement learning has increased. This is due to Diplomacy having traits which makes it challenging for computers to learn. In Diplomacy up to seven players play against each other. The players do their moves simultaneously, and each player has many moves to choose amongst at every turn. This makes the number of joint actions pr. turn sometimes exceed 10^{64}. Diplomacy also has many board configurations.

LOLA is a policy gradient algorithm which consideres the learning of other agents. A LOLA agent tries to maximize its performance by considering how a change in its own policy will affect the policy of the other agents. Previously, this algorithm has proved to generate cooperation between agents. A2C is another, simpler policy-gradient algorithm, and does not possess the capability of generating cooperation between agents.

Players who cooperate will have a benefit in Diplomacy. Even though previous attempts to learn a computer to play Diplomacy have been successful, none of the attempts have involved using an algorithm which generates cooperation amongst agents. Therefore, in this report it is tested whether LOLA is better than A2C at learning an agent to play Diplomacy.

LOLA has been tested on the standard map in Diplomacy. In addition, the algorithm has been tested on a smaller map called "Pure". The test results shows that LOLA is computational demanding; It is not suitable to learn an agent to play games on the standard map. The test results also shows that the LOLA-agent learns to play a game on the "Pure" map, but it performs worse than the A2C-agent. The LOLA-agent also seems to be worse at cooperating when playing Diplomacy, compared to the A2C-agent.

Forord

Reinforcement learning sitt potensiale til å løse et bredt spekter av oppgaver, gjør det til et interessant emne å studere. Selv om det i de siste årene har vært gjort stor fremgang innen reinforcement learning, er det mange interessante utfordringer som gjenstår.

Brettspillet Diplomacy innehar noen egenskaper som gjør spillet til en stor utfordring for en reinforcement learning agent. Dette har ført til at Diplomacy de siste årene har fått økt oppmerksomhet innen forskning på reinforcement learning. Ved å arbeide med kombinasjonen av reinforcement learning og Diplomacy, får man muligheten til å fordype seg i et problem som ligger i ytterkant av dem man hittil har klart å løse med reinforcement learning.

Oppgaven har vært både utfordrende og lærerik, og har gitt meg nyttig erfaring i å arbeide med komplekse problemstillinger innen kunstig intelligens.

Jeg ønsker å takke Keith L. Downing for god veiledning.

Andreas Moe Hatlø
Ålesund, 27. mai 2022

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

Dette kapittelet starter med en presentasjon av bakgrunn og motivasjon[1] for masteroppgaven. Deretter følger en beskrivelse av mål og forskningsspørsmål, før det gis en oversikt over metoden brukt til å besvare forskningsspørsmålene. Til slutt gis det en oversikt over rapportens innhold.

1.1 Bakgrunn og Motivasjon

En av hovedutfordringene innen kunstig intelligens er å lage en agent som innehar kunstig generell intelligens. En slik agent vil kunne løse et bredt spekter av oppgaver, inkludert de fleste oppgaver et menneske klarer å utføre. Enkelte forskere hevder at reinforcement learning (RL) har potensiale til å kunne oppnå kunstig generell intelligens [Silver et al., 2021], men eksisterende RL-algoritmer har vist seg å være langt unna å skape en slik form for intelligens.

Spill er mye brukt til å teste ut ulike algoritmer innen reinforcement learning. Så langt har man innen reinforcement learning oppnådd stor suksess i flere typer spill. Dette gjelder blant annet sjakk, Go og flere Atari-spill [Schrittwieser et al., 2020].

No-Press Diplomacy er et brettspill for opptil sju spillere, der spillerne spiller som en av sju stormakter i Europa på begynnelsen 1900-tallet. Spillbrettet er et kart over Europa, hvor verdensdelen er delt opp i provinser. Målet med spillet er å ta kontroll over et gitt antall provinser. Selv om kun én spiller kan vinne i No-Press Diplomacy, er det en stor fordel for spillerne å kunne samarbeide imens spillet pågår. Spillerne gjennomfører trekkene sine samtidig, og No-Press Diplomacy har

[1] Delkapittelet Bakgrunn og motivasjon er tematisk likt med tilsvarende delkapittel i forfatterens upubliserte fordypningsrapport [Hatlø, 2021].

svært mange mulige trekk pr. runde. Der sjakk og Go henholdsvis har 35 og 250 mulige trekk pr. runde i gjennomsnitt [Sutton og Barto, 2018], er det estimert at gjennomsnittlig antall trekk i No-Press Diplomacy er $10^{45.8}$ [Anthony et al., 2020]. Disse egenskapene gjør No-Press Diplomacy til en interessant utfordring innen reinforcement learning.

I de siste årene har det blitt gjort flere forsøk på å lære en RL-agent å spille No-Press Diplomacy [Paquette et al., 2019; Anthony et al., 2020; Gray et al., 2021; Bakhtin et al., 2021]. I to av forsøkene lærte agenten å spille No-Press Diplomacy på et nivå som overgår de fleste mennesker [Gray et al., 2021; Bakhtin et al., 2021]. I tre av disse forsøkene har agentene først blitt trent på data fra spill spilt av mennesker, før en RL-algoritme eller en søkealgoritme har blitt brukt til å forbedre agentens prestasjoner. I ett av forsøkene, klarte Bakhtin et al. [2021] å lære en agent en utgave av No-Press Diplomacy for to spillere, uten å bruke data fra spill spilt av mennesker.

En agent som skal kunne mestre et bredt spekter av oppgaver, vil ofte ikke ha tilgang på data som viser hvordan andre har løst oppgavene. For at en slik agent skal kunne bli en realitet, vil det derfor være nødvendig å se videre på hvordan agenter kan lære utelukkende fra RL-metoder. Av den grunn vil fokuset i denne rapporten være på hvordan man kan trene opp en Diplomacy-agent utelukkende fra selvspill.

1.2 Mål og Forskningsspørsmål

Målet og forskningsspørsmålene har lagt føringen for arbeidet i dette prosjektet. I forrige avsnitt ble det fastslått at det kan være interessant å se på metoder hvor en agent lærer No-Press Diplomacy fra selvspill. Av dette følger målet for dette prosjektet:

Mål*[2]*: *Å lage en reinforcement learning-agent som lærer å spille No-Press Diplomacy fra selvspill.*

Når en agent skal lære å spille No-Press Diplomacy fra selvspill, er det nærliggende å bruke reinforcement learning. For at reinforcement learning skal kunne anvendes på et problem, er det flere spørsmål som må besvares. Det mest åpenbare er hvilken algoritme som vil fungere best til å løse problemet. Derav kommer forskningsspørsmål 1:

Forskningsspørsmål 1: *Hvilken reinforcement learning-algoritme egner seg best*

[2] Mål og forskningsspørsmål er hentet fra forfatterens upubliserte fordypningsrapport [Hatlø, 2021].

til å lære No-Press Diplomacy?

Om algoritmen skal anvendes på et stort problem, vil det ofte være nødvendig å bruke kunstige nevrale nettverk som funksjonstilnærmere. Ettersom No-Press Diplomacy har mange mulige trekk pr. runde, og mange ulike brettkonfigurasjoner, vil det være nødvendig å bruke kunstige nevrale nettverk som policy- og verdinettverk. Forskningsspørsmål 2 er bestemt på bakgrunn av dette:

Forskningsspørsmål 2: *Når No-Press Diplomacy skal læres, hvilken nettverksarkitektur egner seg best til å lære policy- og verdifunksjoner?*

1.3 Metode

Til å besvare forskningsspørsmål 1, har en algoritme kalt LOLA blitt sammenliknet med A2C. LOLA er en policy-gradient-algoritme som forsøker å påvirke andre agenters policy ved å endre sin egen policy. A2C, som også er en policy-gradient-algoritme, mangler denne egenskapen, og tar ikke hensyn til at andre agenter kan endre sin policy. LOLA har tidligere vist seg å kunne generere samarbeid mellom agenter, og ettersom samarbeid er viktig i No-Press Diplomacy, har det blitt testet om LOLA fungerer bedre til å lære No-Press Diplomacy enn A2C. Det har også blitt testet i hvilken grad LOLA bidrar til å øke samarbeid mellom agenter som spiller Diplomacy.

Til å besvare forskningsspørsmål 2, har det blitt sett på hvilke nettverksarkitekturer som tidligere er brukt til å lære No-Press Diplomacy. Det har blitt gjort en vurdering av hvilke deler av de ulike nettverkene som har bidratt til å gi et godt resultat. Disse delene har blitt brukt til å bygge et policy- og verdinettverk. Nettverkene har blitt brukt i kombinasjon med LOLA og A2C for å lære agentene å spille No-Press Diplomacy.

1.4 Rapportens Oppbygging

Denne rapporten består av seks kapitler. I kapittel 2 blir det gjennomgått teori som er relevant for å forstå fremgangsmåten brukt til å besvare forskningsspørsmålene. Relatert arbeid er beskrevet i kapittel 3. Metoden brukt til å besvare forskningsspørsmål er beskrevet i kapittel 4. I kapittel 5 vil resultatet av å anvende metoden bli presentert og analysert. Til slutt vil arbeidet bli konkludert i kapittel 6. Konklusjonen innebærer blant annet en vurdering av i hvilken grad arbeidet har gitt et svar på forskningsspørsmålene.

Kapittel 2

Bakgrunn

I dette kapittelet vil det bli sett på teori[1] som er relevant for å forstå metoden som er beskrevet i kapittel 4. Teorien vil også være til hjelp med å forstå relatert arbeid presentert i kapittel 3.

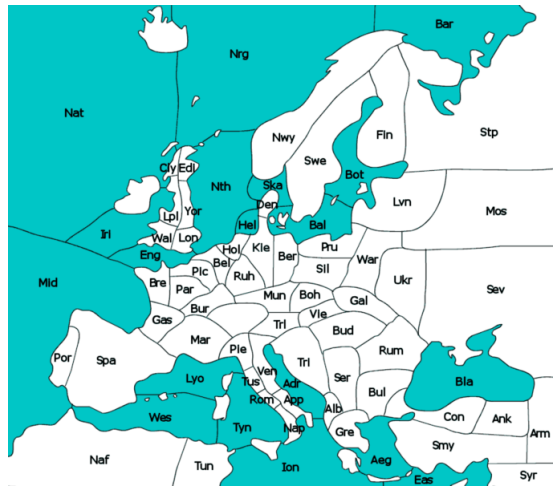
2.1 Diplomacy

Diplomacy er et brettspill for to til sju spillere. Hver spiller spiller som en av stormaktene England, Tyskland, Russland, Tyrkia, Italia, Frankrike eller Østerrike. Som vist i figur 2.1, foregår spillet på et kart over Europa, hvor Europa er inndelt i provinser. Provinsene kan enten være landområde, slik som Paris (PAR), kystområde, slik som London (LON), eller havområde, slik som nordsjøen (NTH). I provinsene har landene mulighet til å plassere brikker. Det finnes to ulike typer brikker i spillet, arméer og flåter, og det kan aldri være mer enn én brikke plassert i hver provins. Arméer kan plasseres i land- og kystprovinser, mens flåter kan plasseres i kyst- og vannprovinser.

34 av provinsene er forsyningssentre, og spilleren som først får kontroll over 18 av forsyningssentrene vinner spillet. I starten av spillet har hvert land enten en armé eller en flåte plassert i forsyningssentrene som ligger i landenes geografiske område. For eksempel starter Tyskland med brikker i Berlin (BER), München (MUN) og Kiel (KIE).

I Diplomacy starter man på begynnelsen av 1900-tallet. Hvert år består av to runder, en vårrunde og en høstrunde, som igjen er delt opp i faser. Selv om Diplomacy er et spill hvor kun én spiller kan vinne, vil man kunne ha store

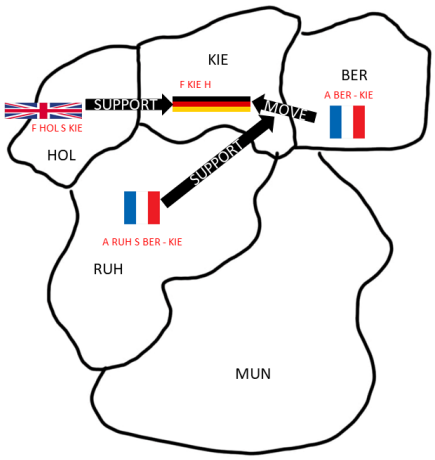
[1] Noe av teorien som blir beskrevet i dette kapittelet var kort beskrevet i forfatterens upubliserte fordypningsrapport [Hatlø, 2021]. Dette gjelder avsnitt 2.1, 2.2, 2.4, 2.5, 2.6, 2.7. Selv om innholdet er tematisk likt med fordypningsrapporten, er innholdet i stor grad ulikt.



Figur 2.1: Spillbrett brukt i Diplomacy [Porge og Edinborgarstefan, 2006]

fordeler av å samarbeide underveis i spillet. Vårrunden starter derfor med en diplomatisk fase hvor spillerne har mulighet til å kommunisere med hverandre og inngå allianser.

Etter den diplomatiske fasen, begynner man å skrive ned ordrer. Det finnes fire typer ordre: "HOLD", "MOVE", "SUPPORT" og "CONVOY". En "HOLD"-ordre gitt til en armé eller en flåte betyr at brikken skal, om mulig, bli værende i sin provins. "A PAR H" er et eksempel på en "HOLD"-ordre som sier at arméen i Paris skal forsøke å bli værende i provinsen. En "MOVE"-ordre sier at man ønsker at en brikke skal flyttes fra en provins til en annen. For eksempel betyr ordren "F ENG - IRI" at en ønsker å flytte flåten som befinner seg i Den engelske kanal, til Irskesjøen. En brikke kan kun flyttes fra en provins til en naboprovins. Dersom provinsen man ønsker å flytte til ikke er okkupert av en annen brikke, vil "MOVE"-ordren bli utført. Derimot, om det finnes en brikke i provinsen man forsøker å flytte til, er en avhengig av støtte fra en annen brikke for å kunne utføre "MOVE"-ordren. Støtte mottar man ved at en annen brikke gis en "SUPPORT"-ordre. Et eksempel på en konflikt er vist i figur 2.2. Den tyske flåten som befinner seg i Kiel har fått en "HOLD"-ordre ("F KIE H"). Samtidig har den franske arméen stasjonert i Berlin fått en ordre om å flytte til Kiel ("A BER - KIE"). Dersom ingen av brikkene har støtte, vil den tyske flåten bli liggende i Kiel, mens den franske arméen fremdeles vil være stasjonert i Berlin. Men dersom den franske arméen i Berlin får støtte av den franske arméen i Ruhr ("A RUH S A BER - KIE"), vil franskmennene være i flertall og "MOVE"-ordren kan gjennomføres.



Figur 2.2: Bildet viser en konflikt som kan oppstå i Diplomacy. Frankrike forsøker å flytte en armé, med støtte fra en armé plassert i Ruhr, fra Berlin til Kiel. I Kiel forsøker en tysk flåte å bli værende. Den tyske flåten får støtte fra en engelsk flåte. Resultatet er at arméen i Berlin ikke får flytte til Kiel. Avlange flagg representerer flåter, mens kvadratiske flagg representerer arméer.

Om den tyske flåten i Kiel får støtte fra den engelske arméen i Holland ("A HOL S F KIE"), vil styrkeforholdet igjen være utjevnet og både flåten i Kiel og arméen i Berlin blir stående. I enhver konflikt er det brikken som mottar mest støtte som får utført sin ordre.

Den siste ordretypen, "CONVOY", kan kun gis til flåter som befinner seg i vannprovinser. En "CONVOY"-ordre brukes til å flytte en armé fra en kyst til en annen via vannprovinsene. Dersom alle vannprovinsene mellom to kyster er okkupert av flåter, kan disse flåtene i samarbeid flytte en armé. For eksempel kan en armé flyttes fra Bretagne til London via den engelske kanal. Dette gjøres ved å gi flåten i den engelske kanal følgende ordre: "F ENG C A BRE - LON".

Etter at ordrene er gitt i ordrefasen, må ordrene utføres i resolusjonsfasen. Siden alle ordrene gjøres samtidig av spillerne, må man i resolusjonsfasen gå over å se på hvilke ordrer som kan gjennomføres. Den fjerde og siste fasen i vårrunden er fasen for tilbaketrekning og oppløsning. I denne fasen må alle brikker som har tapt sin plass i en provins flyttes til en tilgjengelig naboprovins. Dersom ingen naboprovinser er ledige, fjernes brikken fra spillet.

Høstrunden består i tillegg til de samme fasene som vårrunden, av en fase for å anskaffe eller tape enheter. I slutten av høstrunden må alle spillere justere antall brikker slik at det er likt med antallet forsyningssentre spilleren har kontroll

over. Dersom spilleren har flere brikker enn forsyningssentre, må spilleren fjerne noen av sine eksisterende brikker. Dersom spilleren har færre brikker enn forsyningssentre, har spilleren mulighet til å plassere nye enheter i forsyningssentrene som ligger i eget geografisk område. For eksempel kan Tyskland kun plassere nye enheter i forsyningssentrene Kiel, Berlin og München, under forutsetning av at Tyskland har kontroll over disse forsyningssentrene.

No-Press Diplomacy er en utgave av Diplomacy hvor den diplomatiske fasen er utelatt. Videre i rapporten brukes "Diplomacy" om begge varianter.

Avalon Hill sin hjemmeside har en mer detaljert beskrivelse av reglene i Diplomacy [avalonhill.com, ud].

2.2 Spillteori

Spillteori forsøker, ved hjelp av matematikk, å beskrive hvordan rasjonelle agenter handler i ulike situasjoner [Shoham og Leyton-Brown, 2008]. I situasjonene står agentene ovenfor valg, der valgmulighetene medfører ulike konsekvenser for agenten.

Begrepet "nytte" står sentralt i spillteori. Nytte, u, er en funksjon som sier noe om hvor bra et utfall, o, er for en agent. Funksjonen avbilder utfall i et reelt tall ($u : O \rightarrow R$), noe som gjør det mulig for agenten å rangere utfallene etter hvor mye de foretrekkes. Nytten kan også være negativ. Målet for en rasjonell agent vil være å gjøre valg som maksimerer agentens forventede nytte. Et eksempel er en rasjonell agent som skal ta valget om han skal delta i et lotteri. En slik agent står ovenfor to valg: delta i lotteriet eller ikke. Lotteriet kan for eksempel ha to mulige utfall, gevinst (o_1) eller ikke gevinst (o_2), hvor sannsynligheten for gevinst er p_1. Forventet nytte av å ikke delta i lotteriet er 0, ettersom agenten verken taper eller vinner noe på å ikke delta. Forventet nytte av å delta i lotteriet er $p_1 * u(0_1) + (1 - p_1) * u(0_2)$. Avhengig av nytten av gevinsten (f.eks. en stor pengepremie), nytten av å ikke få gevinst (f.eks. kostnaden ved å delta i lotteriet) og sannsynligheten for å vinne, vil den rasjonelle agenten delta i lotteriet, eller ikke. Dersom denne vektede summen er større enn 0, vil den rasjonelle agenten foretrekke å delta i lotteriet. Om den vektede summen er 0, vil agenten være likegyldig til å delta i lotteriet, og dersom summen er mindre enn 0, vil agenten foretrekke å ikke delta i lotteriet.

Eksempelet ovenfor er ikke representativt for alle problemer en agent kan stå ovenfor. I mange tilfeller finnes det andre agenter som påvirker hva agenten kan oppnå. Et eksempel på dette er spillet Diplomacy, hvor man blant annet kan være avhengig av støtte fra andre spillere for å få gjennomført et trekk. Et annet eksempel er spillet "Fangens Dilemma". I Fangens Dilemma avhøres to fanger adskilt fra hverandre. Begge fangene har to valg; De kan enten angi den andre fangen (D), eller ikke angi ham (C). Nytten for de to fangene er gitt i tabell

2.1. I dette tilfellet samsvarer nytten med fengselsstraffen fangene får. Om begge fangene velger å ikke angi den andre (C, C), vil de begge få en fengselsstraff på 1 år. Om begge velger å angi den andre (D, D), blir fengselsstraffen to år. Dersom kun en av fangene angir sin medsammensvorne ((C, D) eller (D, C)), vil den som tyster slippe fri, mens den som tier må sone 3 år i fengsel.

Når det finnes andre agenter, vil beregningen av forventet nytte bli mye vanskeligere. Dette skyldes at utfallet av en handling også bestemmes av handlingene til de andre agentene. Derfor må en finne andre måter enn beregning av forventet nytte for å finne ut hvordan rasjonelle agenter handler. I spillteori brukes løsningskonsept til dette formålet. Et av de mest kjente løsningskonseptene er Nash-ekvilibrium.

I spillteori skiller en mellom to ulike former for strategier: ren strategi og blandet strategi. En ren strategi går ut på at en agent i en gitt situasjon alltid velger den samme handlingen. I Fangens Dilemma tilsvarer dette å alltid velge handling C, eller å alltid velge handling D. Om en agent spiller en blandet strategi, velger han handling tilfeldig etter en gitt sannsynlighetsfordeling. Dette kan for eksempel være å gjøre handling C med 70% sannsynlighet og handling D med 30% sannsynlighet.

En strategiprofil, $s = (s_1, s_2, ..., s_n)$, er en kombinasjon av strategier, én for hver agent. s_i representerer spiller i sin strategi. $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ...s_n)$ er en strategiprofil hvor spiller i sin strategi er utelatt. Nyttefunksjonen, $u_i(s_i, s_{-i})$, beskriver nytten spiller i har av å spille strategi s_i mot strategiprofilen s_{-1}. I henhold til Shoham og Leyton-Brown [2008], defineres et Nash-ekvilibrium som en strategiprofil, s, hvor alle spillere, i, spiller en beste respons, s_i^b, mot s_{-i}. En beste respons vil si at man ikke kan finne en annen strategi, $s_i^{'}$, som har større nytte for agent i når de andre agentene spiller s_{-i} ($s_i^b = \arg\max_{s_i} u_i(s_i, s_{-i})$). Dette innebærer at ingen agenter har insentiv til å bytte strategi når et spill har havnet i et Nash-ekvilibrium. Når et spill ikke er i et Nash-ekvilibrium, kan en eller flere agenter øke sin nytte ved å endre strategi. Dersom spillet spilles av rasjonelle agenter, vil de agentene som tjener på det endre strategi, og spillet vil flyttes mot et Nash-ekvilibrium. Det er derfor forventet at et spill som spilles av rasjonelle agenter vil havne i et Nash-ekvilibrium.

I Fangens Dilemma er (D, D) et Nash-ekvilibrium. Det vil si at når begge fangene velger å vitne mot hverandre, kan ingen oppnå mindre fengselsstraff ved å endre strategi alene.

Fangens Dilemma kan også spilles i en iterert utgave hvor fangene står ovenfor problemstillingen om å angi den andre fangen, eller ikke, gjentatte ganger. En fange som er i stand til å huske handlingene til motspilleren fra forrige runde av Fangens Dilemma, kan utnytte denne informasjonen til sin fordel. Anatol Rapoport har kommet opp med en strategi, kalt "Tit for Tat", som har vist seg å være effektiv i den itererte versjonen av Fangens Dilemma [Axelrod, 1980].

		Fange 2 C	Fange 2 D
Fange 1	C	$-1, -1$	$-3, 0$
	D	$0, -3$	$-2, -2$

Tabell 2.1: Nytte ved ulike utfall i Fangens Dilemma

"Tit for Tat" strategien går ut på å starte første runde med å ikke vitne (C), og deretter gjøre det samme som motspilleren gjorde i forrige runde. "Tit for Tat"-strategiens suksess viser hvordan agenter med hukommelse kan oppnå mer ved å være villig til å samarbeide, men uten å være tilgivende.

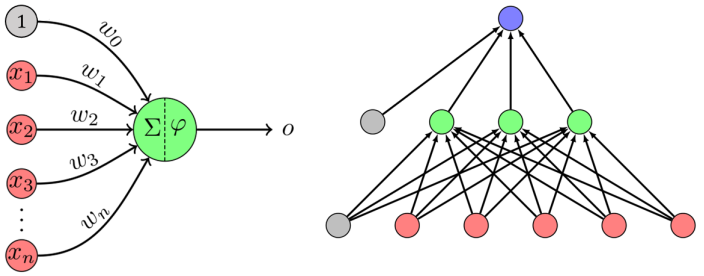
John Nash beviste at alle endelige spill har minst ett Nash-ekvilibrium [Nash, 1951]. Om en setter en øvre grense på hvor mange runder Diplomacy kan vare, vil Diplomacy være et endelig spill. Å finne et Nash-ekvilibrium, kan derfor være en god læringsstrategi for en Diplomacy-spillende agent. I Diplomacy er en agent tjent med å samarbeide med andre spillere. Strategier som oppfordrer til samarbeid, vil også være til agentens fordel.

2.3 Kunstige Nevrale Nettverk

Et kunstig nevralt nettverk består av nevroner som er koblet sammen i et nettverk. Hvert nevron mottar inndata, gjør noen beregninger på inndataen, og sender utdata til dets naboer. Hver kobling mellom nabonevroner i nettverket har en vekt, og når et nevron mottar data fra et annet nevron, multipliseres dataen med vekten [Russell og Norvig, 2016].

Til venstre i figur 2.3 vises en illustrasjon av et nevron. Nevronet tar inn et sett med verdier $(x_1, ..., x_n)$. Av disse verdiene regnes det ut en vektet sum, $\sum_{i=1}^{i=n}(x_i * w_i) + w_0$, som gis til en aktiveringsfunksjon, φ. w_0 er en verdi, kalt bias, som inkluderes i den vektede summen. Nevronet kan sees på som en matematisk funksjon, og ved å endre vektene, endrer man funksjonen som nevronet utgjør. Bias-verdien regnes også som en vekt, og ved å legge w_0 til den vektede summen, er det mulig å endre nevronets utdata når alle inndata-verdiene er lik null. Om φ er en ikke-lineær aktiveringsfunksjon, for eksempel hyperbolsk tangens, vil nevronet utgjøre en ikke-lineær funksjon.

Til høyre i figur 2.3 vises det hvordan nevroner kan settes sammen til et feedforward-nettverk. Feedforward-nettverket er lagdelt, og i det nederste laget finner man nettverkets inndata. De røde sirklene representerer de ulike inndataverdiene, $x_1, ..., x_n$. De grå sirklene representerer verdien 1. I laget i midten finner man tre grønne sirkler som representer nevroner. For hvert nevron beregnes utdata i henhold til prosedyren beskrevet ovenfor. Nevronenes utdata sendes deretter



Figur 2.3: Oppbygging av et feedforward-nettverk. Til venstre: Et nevron [Thoma, 2014][3]. Til høyre: Et feedforward-nettverk satt sammen av nevroner [Thoma, 2013].

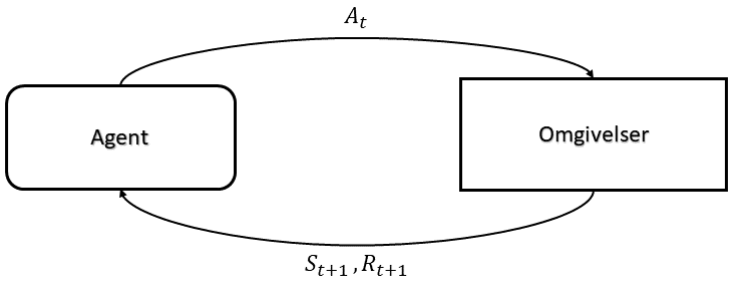
videre til nevronet i det siste laget, representert av en blå sirkel. Dette nevronet bruker utdataen fra det forrige laget til å beregne en verdi som blir hele nettverkets utdata. Antall nevroner i hvert lag og antall lag med nevroner kan variere etter hvilken funksjon man ønsker at nettverket skal utgjøre.

Som regel ønsker man at det nevrale nettverket skal representere en funksjon, f, man ikke har full kjennskap til. I mange tilfeller har man kun tilgjengelig eksempler på inndata med tilhørende utdata for f. Man ønsker i slike tilfeller å justere vektene slik at nettverket på best mulig måte representerer f. Det er i slike tilfeller vanlig å la nettverket inngå i en tapsfunksjon (L), som gir et estimat på hvor likt nettverket er f. Jo mindre tapsfunksjonens funksjonsverdi er, desto mer presist estimeres det at nettverket gjengir f.

$L = (y - \hat{y})^2$ er et eksempel på en tapsfunksjon der y er f sin korrekte funksjonsverdi og $\hat{y}$ er nettverkets utdata. Ved å derivere tapsfunksjonen med hensyn på vektene i nettverket, finner man gradienten til tapsfunksjonen ($\nabla_w L$). Å endre vektene i motsatt retning av gradienten, er en vanlig måte å forsøke å øke prestasjonen til nettverket. På den måten vil man redusere funksjonsverdien til tapsfunksjonen, og dersom tapsfunksjonen gir et godt estimat på hvor likt nettverket er f, vil nettverket bli en bedre tilnærming til f.

Feedforward-nettverk er ikke den eneste formen for kunstige nevrale nettverk. Recurrent Neural Network og Graph Neural Networks er eksempler på andre typer kunstig nevrale nettverk. Disse er beskrevet i henholdsvis avsnitt 2.6 og avsnitt 2.7.

[3]Bildet av nevronet er endret ved å gjøre den øverste sirkelen grå, og sette inn "1" i sirkelen.



Figur 2.4: Agent interagerer med sine omgivelser

.

2.4 Reinforcement Learning (RL)

En RL-agent opererer i omgivelser hvor agenten kan observere omgivelsenes tilstand og gjennomføre handlinger. Basert på hvilke konsekvenser handlingene har, mottar RL-agenten en belønning. I reinforcement learning lærer agenten fra belønningen den mottar. Målet til RL-agenten er å maksimere belønningen den mottar over tid [Sutton og Barto, 2018].

Figur 2.4 illustrer hvordan en agent interagerer med sine omgivelser. Som en kan se av figuren gjør agenten handling A_t i tidssteg t. Handlingene fører til at omgivelsene endres slik at agenten kan observere tilstand S_{t+1} i tidssteg $t+1$. Samtidig blir agenten belønnet med R_{t+1}. Belønningen gjenspeiler hvor bra effekten av handlingen A_t er for agenten. Basert på eksisterende kunnskap, og den nye informasjonen (R_{t+1}, S_{t+1}), vil agenten gjøre en ny handling A_{t+1}.

Policy og verdifunksjoner er viktige elementer innen RL. En policy sier noe om hvilken handling en agent skal gjøre i ulike situasjoner. Ofte vil en policy være en betinget sannsynlighet, $\pi(A_t \mid S_t = s_t)$, over handlinger, $a_t \in A_t$, hvor betingelsen er tilstanden til agentens omgivelser ($S_t = s_t$). Policyen sier dermed noe om hvor sannsynlig det er at agenten velger en gitt handling i en gitt tilstand. Agenten vil forsøke å lære en policy som maksimerer belønningen den får over tid.

En verdifunksjon angir hvilken aggregert og neddiskontert belønning en agent kan forvente å få i fremtiden. Det finnes to ulike verdifunksjoner: en V-funksjon og en Q-funksjon. For å forstå forskjellen mellom de to ulike funksjonene, bruker Sutton og Barto [2018] tre likninger. Likning 2.1 definerer den aggregerte og neddiskonterte belønningen en agent vil få fra tidssteg t. I likningen er γ diskonteringsfaktoren, og R_k belønningen agenten vil få i tidssteg $k, t < k$. T angir det tidssteget oppgaven til RL-agenten er ferdig. Oppgaven en RL-agent står ovenfor

kan enten være episodisk eller kontinuerlig. Dersom oppgaven er episodisk finnes det en tilstand s_T hvor oppgaven kan sies å være ferdig. T i likning 2.1 er derfor det tidssteget hvor agenten observerer S_T. Om oppgaven er kontinuerlig, finnes det ikke en tilstand hvor oppgaven kan sies å være ferdig. Man setter i så fall $T = \infty$.

V-funksjonen er definert ved likning 2.2. V-funksjonen er den forventede verdien til G_t, gitt at agenten observerer tilstand s_t. Q-funksjonen er definert ved likning 2.3, og er den forventede verdien av G_t, gitt at agenten observerer tilstand s_t og gjennomfører handling a_t. V-funksjonens funksjonsverdi kalles tilstandsverdi, mens Q-funksjonens funksjonsverdi kalles handlingsverdi. I tilfeller hvor en eksakt verdifunksjon ikke er tilgjengelig, er det vanlig at agenten lærer seg et estimat av verdifunksjonen.

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.1}$$

$$v(s_t) = \mathbf{E}[G_t \mid S_t = s_t] \tag{2.2}$$

$$q(s_t, a_t) = \mathbf{E}[G_t \mid S_t = s_t, A_t = a_t] \tag{2.3}$$

Videre følger en beskrivelse av noen metoder som blir brukt innen reinforcement learning til å lære en verdifunksjon og en policy.

2.4.1 Temporal Difference (TD) Learning

I TD-learning oppdaterer man agentens verdifunksjon, V, ved å minimere en verdi kalt TD-error. TD-error, vist i likning 2.4, er forskjellen mellom V sine verdier for observert tilstand i tidssteg $t + 1$ og observert tilstand i tidssteg t, pluss belønningen agenten fikk i tilstand $t + 1$. TD-error minimeres ved å øke $V(S_t)$ dersom verdien til R_{t+1} tyder på at nåværende estimat på tilstandsverdien er for lavt. Motsatt reduseres $V(S_t)$ dersom R_{t+1} tyder på at nåværende estimat er for høyt. Endringen i $V(S_t)$, vist i likning 2.5, tilsvarer en konstant α, kalt læringsraten, multiplisert med δ_t.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.4}$$

$$\Delta V(S_t) = \alpha \delta_t \tag{2.5}$$

Ved å bruke δ_t til å oppdatere $V(S_t)$, kan agenten oppdatere verdifunksjonen i det øyeblikket den får en ny belønning. Agenten slipper derfor å vente til den

har tilgang på verdien til G_t, noe den først får når episoden er over. Dette gjør at TD-learning egner seg godt ved kontinuerlige oppgaver.

SARSA er en RL-algoritme som benytter seg av δ_t til å lære en Q-funksjon. I SARSA er V-funksjonen i likning 2.4 byttet ut med en Q-funksjon. Algoritmen oppdaterer $Q(S_t, A_t)$ iterativt med $\alpha\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ [Sutton og Barto, 2018]. Policyen til en SARSA-agent er å velge handlingen som har størst Q-verdi med sannsynlighet $(1 - \epsilon)$. Handlingen med størst Q-verdi er handlingen agenten tror vil gi den største belønningen over tid, G_t. Ettersom agenten må prøve ut de forskjellige handlinger for å finne den beste, velger agenten med sannsynlighet ϵ en tilfeldig handling.

Q-learning er en RL-algoritme som likner på SARSA. Forskjellen mellom Q-learning og SARSA er at $\delta_t = R_{t+1} + \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$ [Sutton og Barto, 2018]. I Q-learning oppdateres ikke Q-funksjonen etter hvilken handling som ble gjort i tidssteg $t + 1$, men etter hvilken handling som har den største Q-verdien i dette tidssteget. Den praktiske forskjellen mellom SARSA og Q-learning er at agenten ved SARSA forsøker å lære en Q-funksjon som er optimal gitt at agenten velger en tilfeldig handling med sannsynlighet ϵ. Ved Q-learning forsøker agenten å lære en Q-funksjon som er optimal gitt at $\epsilon = 0$, uavhengig av den faktiske verdien til ϵ. Dette illustrerer forskjellen mellom to ulike typer algoritmer innen reinforcement learning. SARSA er en "on-policy"-algoritme, hvor policyen som læres er den sammen policyen som blir brukt under læringen. Q-learning er en "off-policy"-algoritme, hvor policyen som læres er forskjellig fra policyen som blir brukt under læringen.

Som en mellomting mellom å oppdatere verdifunksjonen med G_t og å bruke SARSA eller Q-learning, er det mulig å anvende n-stegs-metoder. Ved n-stegs-metoder summerer man den diskonterte belønningen fra de n foregående tidsstegene. Denne summen brukes så til å oppdatere verdifunksjonen for tidssteg t. I n-stegs utgaven av SARSA bruker man $\delta_{t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n Q(S_{t+n}, A_{t+n}) - Q(S_t, A_t))$ [Sutton og Barto, 2018]. Dersom man setter $n = 1$ får man SARSA-algoritmen. Fordelen med n-stegs-metodene er at jo større n er, desto bedre er estimatet på den sanne verdien av verdifunksjonen. En av ulempene er at agenten må vente med å oppdatere verdifunksjonen til nok data er samlet inn.

2.4.2 Policy-Gradient-Metoder

I mange tilfeller er antallet tilstander som agenten kan observere så stort at det ikke er mulig å lagre en policy-verdi, $\pi(A_t \mid S_t = s)$, for hver mulig tilstand s. I slike tilfeller trenger man en parametrisert policy. En parametrisert policy kan for eksempel være et kunstig nevralt nettverk som har færre parametere (vekter) enn mulige tilstander.

I policy-gradient-metoder lærer agenten en parametrisert-policy ved å maksimere en funksjon av policyens parametere, $J(\theta)$. Denne funksjonen angir hvor bra agenten er forventet å yte, gitt policyens parametere. På den måten vil en maksimering av funksjonen føre til en maksimering i forventet ytelse for agenten. Agenten lærer ved å derivere $J(\theta)$ med hensyn på policyens parametere, og deretter flytte parameternes verdier i gradientens retning. I henhold til policy-gradient-teoremet er gradienten proporsjonal med $\mathbf{E}[G_t \nabla \ln \pi(A_t \mid S_t, \theta_t)]$ [Sutton og Barto, 2018]. Dette er utgangspunktet for REINFORCE algoritmen som oppdaterer policyen i henhold til likning 2.6. REINFORCE-algoritmen har den fordelen at den er uavhengig av en verdifunksjon for å kunne lære. Siden man trenger G_t for å oppdatere policyen, er en avhengig av å ha gjennomført en hel episode før man kan oppdatere policyen. En annen ulempe med REINFORCE-algoritmen er at G_t ofte har høy varians, noe som kan føre til at lært policy varierer mye for ulike kjøringer av algoritmen.

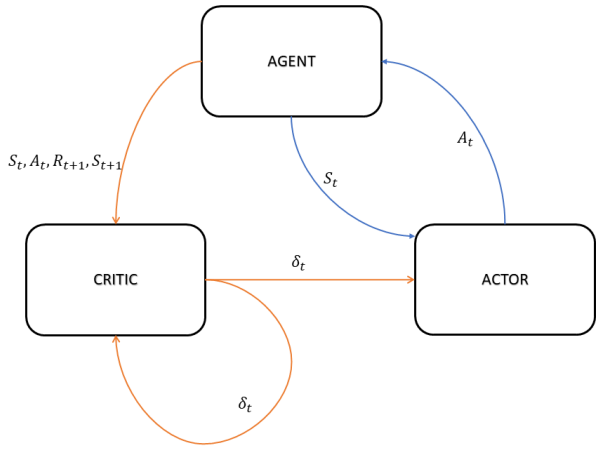
$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \ln \pi(A_t \mid S_t, \theta_t) \tag{2.6}$$

For å håndtere problemet med høy varians, er det mulig å bytte ut G_t i likning 2.6 med δ_t i likning 2.4. I så fall ender man opp med en metode kalt Actor-Critic (A2C). A2C bruker δ_t til å finne ut om en handling ga et resultat som var bedre enn ventet, eller ikke, og endre policyen deretter. Om δ_t blir positiv etter å tatt handling a_t i tilstand s_t, førte a_t til et resultat, $r_{t+1} + \gamma V(s_{t+1})$, som var bedre enn ventet ($V(s_t)$). Dermed bør sannsynligheten for å velge a_t neste gang agenten observerer tilstand s_t økes. Motsatt bør sannsynligheten reduseres om resultatet var dårligere enn forventet. Dette gjøres i A2C.

Figur 2.5 illustrerer hvordan A2C algoritmen fungerer. I A2C er Actor-en ansvarlig for å bestemme policyen, og dermed hvilke handlinger som skal gjøres. Actor-en mottar observert tilstand S_t fra agenten og returnerer en handling A_t. Critic-en er ansvarlig for å vurdere hvor godt agenten presterer. Critic-en holder på verdifunksjonen, og bruker denne til å beregne δ_t. δ_t brukes så til å oppdatere verdifunksjonen og policyen.

2.5 Fiktivt Spill

Fiktivt spill er en metode for å finne Nash-ekvilibrium [Shoham og Leyton-Brown, 2008]. I fiktivt spill går agenten ut i fra at motstanderens strategier ikke endrer seg. Ved å spille mot motstanderne, bygger agenten opp erfaring om hvilke handlinger motstanderen gjør i ulike situasjoner. På den måten kan agenten lære motstandernes strategier av erfaring. I fiktivt spill starter agenten med en antakelse om motstanderens strategi. Basert på denne antakelsen, beregner agenten en beste respons. Når runden er ferdig oppdaterer agenten antakelsene om mot-



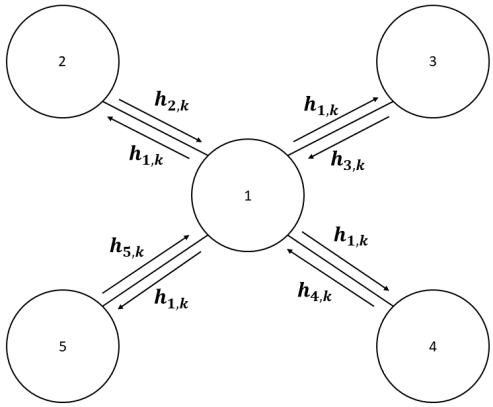
Figur 2.5: Illustrasjon av en A2C-agent. A2C-agenten bruker en Actor for å bestemme hvilken handling A_t som skal gjøres i tilstand S_t. Critic-en brukes til å regnes ut δ_t basert på S_t, A_t, R_{t+1} og S_{t+1}. δ_t brukes så til å oppdatere både Actor og Critic. Oransje linjer representere steg som inngår i læringsprosessen.

standernes strategier ved å se på hvilke handlinger motstanderne gjorde. Agenten antar at en motstanders strategi er den relative frekvensen av handlinger motstanderen har gjort. Basert på de nye antakelsene, beregnes en ny bestes respons. Etter hvert som denne iterative prosessen fortsetter, kan agentens strategi konvergere mot et Nash-ekvilibrium.

2.6 Graph Neural Networks (GNN)

I mange tilfeller kan data struktureres i grafer, hvor data er fordelt over grafens noder og kanter. Hvilke noder som er koblet til hverandre vil kunne si noe om forholdet mellom ulike deler av dataen. Et trivielt eksempel er en graf hvor noder representerer land, og kanter viser hvilke land som er naboer. Om en skal anvende maskinlæring på data strukturert i grafer, er det en fordel å bruke en metode som tar hensyn til grafens struktur. Til dette benyttes "Graph Neural Networks" (GNN) [Hamilton, 2020].

I GNN oppdateres noder iterativt. For hver iterasjon sender hver node informasjon til sine nabonoder, og nabonodene oppdateres med den nye informasjonen. Dermed vil informasjonen som opprinnelig var i én node, spre seg til flere noder i løpet av nettverkets iterasjoner. Denne prosedyren kalles "message



Figur 2.6: "Message passing" mellom noder i et nettverk

passing". I henhold til Hamilton [2020] kan "message passing" bli uttrykt ved likning 2.7. Hver node u har en vektor $\mathbf{h}_u$ som oppdateres iterativt. Oppdateringen skjer via en oppdateringsfunksjon, $UPDATE$, som har forrige iterasjons $\mathbf{h}_u$ og en aggregering av tilsvarende vektorer for nabonodene som input. Både oppdateringsfunksjonen, $UPDATE$, og aggregeringsfunksjonen, $AGGREGATE$, er deriverbare funksjoner. Disse funksjonene er typisk nevrale nettverk. $N(u)$ er mengden av nabonoder til node u. I første iterasjon vil hver nodes vektor, $\mathbf{h}$, være informasjonen som opprinnelig er forbundet med noden.

$$\mathbf{h}_{u,k+1} = UPDATE_k(\mathbf{h}_{u,k}, AGGREGATE_k(\{\mathbf{h}_{v,k}, \forall v \in N(u)\})) \tag{2.7}$$

Figur 2.6 illustrerer hvordan "message passing" foregår mellom noder i et nettverk. I figuren mottar node 1 meldinger, $\mathbf{h}_{v,k}$, fra sine naboer. Samtidig sender node 1 ut en melding, $\mathbf{h}_{1,k}$, til de samme naboene. Meldingene som node 1 mottar aggregeres og brukes til å beregne $\mathbf{h}_{1,k+1}$.

En populær oppdateringsfunksjon, kalt "Graph Convolutional Networks" (GCN), er gitt av Kipf og Welling [2016]. Oppdateringsfunksjonen for GCN er vist i likning 2.8 [Hamilton, 2020][4]. I GCN lar man aggregeringsfunksjonen opptre som oppdateringsfunksjon ved at hver node, u, sender sin vektor, $\mathbf{h_u}$, til seg selv, samt til sine naboer. Summen av alle mottatte vektorer, $\mathbf{h}_v$ multipliseres med en vektmatrise, $\mathbf{W}$. Denne matrisen læres av algoritmen. Deretter anvendes

[4]Kipf og Welling [2016] viser i sin artikkel en mer effektiv måte å beregne grafkonvolusjoner ved å kun bruke multiplisering av matriser. Hamilton [2020] sin likning er tatt med her fordi den på en mer oversiktlig måte viser hvordan grafkonvolusjoner beregnes.

en aktiveringsfunksjon, σ. Alle vektorene som u mottar normaliseres ved å dele vektorene på kvadratroten av antall naboer mottaker har, multiplisert med antall naboer avsenderen har. Normaliseringen bidrar til at vektorene, $\mathbf{h}$, for de ulike nodene ikke får for stor variasjon i størrelse.

$$\mathbf{h}_{u,k+1} = \sigma(\mathbf{W}_k \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_{v,k}}{\sqrt{\mid N(u) \mid\mid N(v) \mid}}) \tag{2.8}$$

GCN kan for eksempel brukes til klassifisering av noder [Kipf og Welling, 2016], eller til å kode informasjon som finnes i en graf [Paquette et al., 2019].

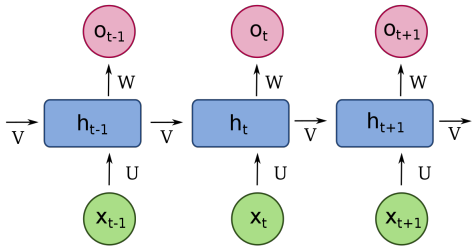
2.7 Recurrent Neural Networks (RNN) og Long Short-Term Memory (LSTM)

Recurrent Neural Networks (RNN) er en type kunstig nevralt nettverk som blir brukt til å behandle sekvensiell data [Goodfellow et al., 2016]. RNN brukes blant annet innen maskinoversettelse, hvor den sekvensielle dataen er setninger. I RNN brukes de samme vektene på alle deler av datasekvensen. Dette kalles parameterdeling, og gjør at RNN kan behandle datasekvenser av ulik lengde.

RNN behandler dataen sekvensielt. Det vil si at man starter med første element i sekvensen, $\mathbf{x}_0$, og behandler dette, før man går videre til neste element, $\mathbf{x}_1$. Dette fortsetter man med til det siste elementet i datasekvensen, $\mathbf{x}_\tau$, er behandlet. Input til RNN i steg t er element $\mathbf{x}_t$ fra datasekvensen, i tillegg til en skjult tilstand $\mathbf{h}_{t-1}$. Den skjulte tilstanden oppdateres av RNN-et for hvert steg og inneholder informasjon om dataen nettverket har sett fram til og med steg $t-1$. Under treningen av RNN-et, håper man at nettverket lærer seg å lagre informasjon i $\mathbf{h}_{t-1}$ som er relevant når de resterende elementene i datasekvensen, $\{\mathbf{x}_{t'} \mid t \leq t^{'} \leq \tau\}$, skal behandles. Ofte vil RNN bli brukt til å gi en output, $\mathbf{o}_t$, for hvert steg.

Likning 2.9 og likning 2.10 viser hvordan RNN behandler data sekvensielt [Goodfellow et al., 2016]. Denne prosessen er også illustrert i figur 2.7. I likning 2.9 kan man se at to vektmatriser, $\mathbf{U}$ og $\mathbf{V}$, brukes til å oppdatere $\mathbf{h}_t$. Matrise $\mathbf{U}$ multipliseres med inputvektoren $\mathbf{x}_t$, og matrise $\mathbf{V}$ multipliseres med den skjulte tilstanden $\mathbf{h}_{t-1}$. Resultatet fra disse multiplikasjonene legges sammen med en bias-vektor, $\mathbf{b}$, og gis som inndata til en aktiveringsfunksjon, σ_h. Likning 2.10 viser hvordan utdata, $\mathbf{o}_t$, genereres ved å multipliseres matrise $\mathbf{W}$ med den skjulte tilstanden $\mathbf{h}_t$. Resultatet fra multiplikasjonen legges til en bias-vektor $\mathbf{c}$, og brukes som inndata til en aktiveringsfunksjon, σ_o.

$$\mathbf{h_t} = \sigma_h(\mathbf{b} + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t) \tag{2.9}$$



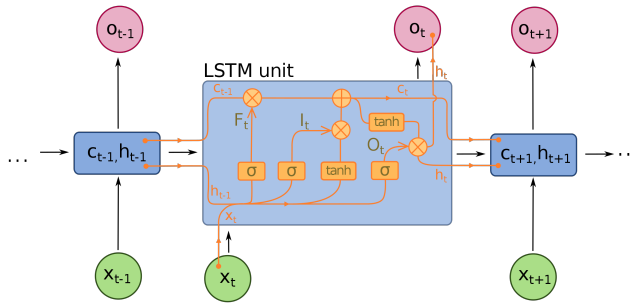
Figur 2.7: Illustrasjon av hvordan RNN behandler data sekvensielt. Denne illustrasjonen er en beskåret utgave av et bilde laget av fdeloche [2017b].

$$\mathbf{o_t} = \sigma_o(\mathbf{c} + \mathbf{W}\mathbf{h}_t) \tag{2.10}$$

Et av problemene med RNN er at gradienten fort kan "eksplodere" eller "forsvinne" dersom datasekvensen er lang. På grunn av kjerneregelen vil en lang datasekvens føre til mange multiplikasjoner av vektene i vektmatrise $\mathbf{V}$ når gradienten til RNN-et skal beregnes. Dersom vektene ikke er lik 1 vil gradienten dermed enten avta eller vokse eksponentielt. Dersom gradienten avtar eksponentielt, og dermed "forsvinner", vil treningen gå tregt. Dersom gradienten vokser eksponentielt, og dermed "eksploderer", vil vektene i nettverket variere stort mellom hver oppdatering.

Long Short-Term Memory (LSTM) er en type RNN som håndterer problemet med gradienter som "eksploderer" eller "forsvinner". Når datasekvensene er lange, fungerer derfor et LSTM-nettverk bedre enn RNN-et beskrevet ovenfor.

Figur 2.8 viser hvordan et LSTM-nettverk fungerer. I hvert steg er input til LSTM-nettverket $\mathbf{x}_t$ og $\mathbf{h}_{t-1}$, i tillegg til en tilstand $\mathbf{c}_{t-1}$. I LSTM-nettverket finnes det tre porter: $\mathbf{F}_t$, $\mathbf{I}_t$ og $\mathbf{O}_t$. En port regulerer hvilke informasjon LSTM-nettverket skal se bort i fra. Port $\mathbf{F}_t$ er en vektor som multipliseres elementvis med $\mathbf{c}_{t-1}$, noe som resulterer i vektor $\mathbf{a}_t$. Av likning 2.11 kan man se at $\mathbf{F}_t$ er output fra aktiveringsfunksjonen "sigmoid", der input er en lineærkombinasjon av $\mathbf{h}_{t-1}$ og $\mathbf{x}_t$ [Goodfellow et al., 2016]. Vektmatrisene $\mathbf{U}^f$ og $\mathbf{V}^f$ kan trenes. Det vil si at nettverket gjennom trening kan bli "flinkere" til å sile ut informasjon som er relevant å ta med videre. $\mathbf{b}_f$ er en bias-vektor. Portene $\mathbf{I}_t$ og $\mathbf{O}_t$ beregnes på tilsvarende måten som $\mathbf{F}_t$, men med andre vektmatriser. Port $\mathbf{I}_t$ multipliseres elementvis med det som tilsvarer skjult tilstand $\mathbf{h}_t$ i RNN-et nevnt ovenfor. Resultatet av denne multipliseringen adderes elementvis med $\mathbf{a}_t$. Dette gir den nye



Figur 2.8: Illustrasjon av en LSTM-celle [fdeloche, 2017a]. σ representerer aktiveringsfunksjonen sigmoid, $\otimes$ representer elementvis multiplikasjon , $\oplus$ representerer elementvis multiplikasjon.

tilstanden $\mathbf{c}_t$. Som en kan se av figur 2.8 blir så $\mathbf{h}_t$ en elementvis multiplisering mellom $\mathbf{O}_t$ og resultatet av $\mathbf{c}_t$ som input til aktiveringsfunksjonen *tanh*.

$$\mathbf{F}_t = \sigma(\mathbf{b}^f + \mathbf{U}^f \mathbf{x}_t + \mathbf{V}^f \mathbf{h}_{t-1}) \tag{2.11}$$

Tilstanden $\mathbf{c}_t$ er grunnen til at problemet med gradienter som "eksploderer" eller "forsvinner" er mindre i LSTM-nettverk enn i RNN-et nevnt ovenfor. Ettersom $\nabla_{\mathbf{c}_{t-1}} \mathbf{c}_t = \mathbf{F}_t$, kan gradienten propagere bakover i nettverket uten at vektene i $\mathbf{V}$ må multipliseres mange ganger etter hverandre.

2.8 Embedding

Embedding blir brukt til å lære vektorrepresentasjoner av kategorisk data [Geron, 2019]. Vektorrepresentasjonene lagres i en matrise. Et eksempel på dette kan være matrise $\mathbf{E}$, gitt i likning 2.12. Matrisen inneholder vektorrepresentasjoner for m ulike kategorier. Hver radvektor representerer én kategori. For å hente ut riktig vektorrepresentasjon til en kategori, multipliseres matrisen med en one-hot-koding av kategorien. Et eksempel på dette er vist i likning 2.13, hvor radvektoren for kategori 2 hentes ut.

Embedding kan utgjøre ett eller flere lag i et kunstig nevralt nettverk. I Embedding-laget omgjøres en kategorisk variabel til en vektor, slik som i likning 2.13. Vektoren brukes deretter som inndata i det etterfølgende laget. På den måten kan embedding være et alternativ til å ha en "one-hot"-koding av den kategoriske dataen som inndata til nettverket.

La f være et kunstig nevralt nettverket som blant annet har matrise $\mathbf{E}$ som en del av sine parametere, $f(\mathbf{x}; \mathbf{E}, \theta)$. $\mathbf{E}$ initialiseres med tilfeldige verdier, likt

vektene til et kunstig nevralt nettverk. Elementene i $\mathbf{E}$ kan også læres på samme måte som vektene i et kunstig nevralt nettverk. f kan for eksempel inngå i en tapsfunksjon, L, slik som beskrevet i avsnitt 2.3. Det er mulig å derivere tapsfunksjonen med hensyn på elementene i $\mathbf{E}$, $\nabla_{\mathbf{E}} L$. Ved å bevege seg i motsatt retning av denne gradienten, vil tapsfunksjonens verdi kunne reduseres. Dette gjør det mulig å lære en vektorrepresentasjon av den kategoriske dataen som er bedre enn en "one-hot"-koding.

$$\mathbf{E} = \begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,n} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ e_{m,1} & e_{m,2} & \cdots & e_{m,n} \end{bmatrix} \tag{2.12}$$

$$\mathbf{e}_2 = \begin{bmatrix} 0 & 1 & \cdots & 0 \end{bmatrix} \begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,n} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ e_{m,1} & e_{m,2} & \cdots & e_{m,n} \end{bmatrix} = \begin{bmatrix} e_{2,1} & e_{2,2} & \cdots & e_{2,n} \end{bmatrix} \tag{2.13}$$

2.9 Batchnormalisering

Batchnormalisering blir ofte brukt i forbindelse med dype nevrale nettverk. Teknikken går ut på å normalisere inndata til hvert lag i det kunstige nevrale nettverket, slik at inndataens fordeling blir mer uavhengig av nettverkets vekter. Batchnormalisering gjør at en endring i vektene for ett lag i mindre grad påvirker hvordan de andre lagene i nettverket presterer. Batchnormaliseringen beregnes i fire steg [Ioffe og Szegedy, 2015]:

1. $\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i$
2. $\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)$
3. $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
4. $y_i = \gamma \hat{x}_i + \beta$

I steg 1 beregnes gjennomsnittet for elementene i batchen. Steg 2 innebærer å beregne batchens standardavvik. I steg 3 normaliseres hvert element i batchen, før hvert element lineærtransformeres i steg 4. I likningene er m antall elementer i batchen, x_i er element i i batchen, og ϵ er en liten verdi for å unngå at $x_i - \mu_B$ deles på 0. γ og β er verdier som kan læres. I stedet for gjennomsnitt og varians for en bestemt batch, er det vanlig å beregne et glidende gjennomsnitt og en

glidende varians over flere batcher. På den måten kan hele treningssettet brukes til å beregne gjennomsnitt og varians.

2.10 Dropout

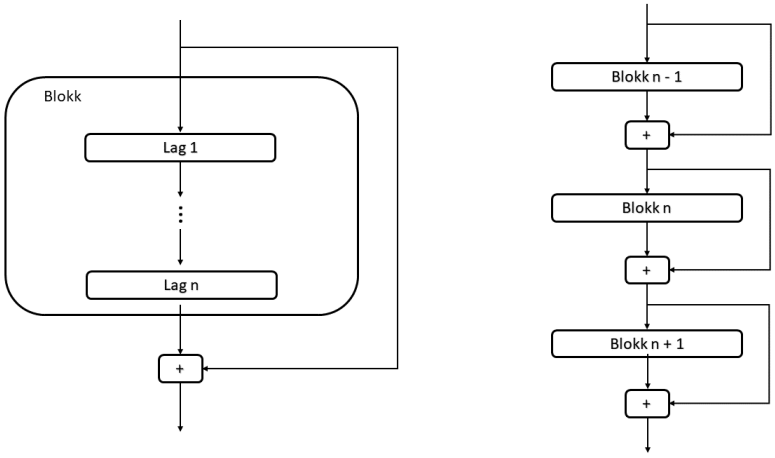
Dropout er en teknikk brukt til å redusere overfitting i kunstige nevrale nettverk [Hinton et al., 2012]. Overfitting vil si at nettverket blir god til å behandle data det er trent på, på bekostning av evnen til å behandle data nettverket ikke har blitt trent på. Når dropout brukes fjernes hvert nevron i nettverket med sannsynlighet ϵ i hvert treningssteg. På den måten trenes et nytt kunstig nevralt nettverket for hver runde med trening. Når nettverket skal gjøre en prediksjon, brukes hele det opprinnelige nettverket. Utdata fra hvert nevron justeres for å ta hensyn til at det under treningen har vært del av et nettverk som er $1 - \epsilon$ ganger så stort som det opprinnelige nettverket. Når nettverket skal predikere, brukes det dermed et slags gjennomsnitt av alle de trente nettverkene som manglet enkelte nevroner.

2.11 Skip Connections

I mange tilfeller kan et dypt nevralt nettverk deles opp i blokker. Hver blokk kan bestå av ett eller flere lag som utfører visse operasjoner på dataen. Ved å legge sammen inndataen til en blokk med utdataen til blokken, får man det som kalles en "skip connection" [He et al., 2015]. "Skip connections" brukes for å unngå et problem ved dype nevrale nettverk hvor nettverkene har en tendens til å prestere dårligere desto flere lag som legges til. I utgangspunktet skulle ikke et dypt nevralt nettverk være dårligere enn et grunt nevralt nettverk, da hvert lag i teorien kan lære seg å la utdataen være den samme som inndataen. "Skip connections" løser dette problemet ved å gjøre det enklere for det nevrale nettverket å la en blokks utdata være likt blokkens inndata. "Skip connections" er illustrert i figur 2.9.

2.12 Oppsummering

I dette kapittelet har det blitt sett på teori som er relevant for forstå metoden presentert i kapittel 4 og relatert arbeid presentert i kapittel 3. Kapittelet startet med en introduksjon til Diplomacy. Deretter fulgte en kort introduksjon til spillteori. I spillteori forsøker man å beskrive hvordan rasjonelle agenter handler i ulike situasjoner. En Diplomacy-spillende agent befinner seg i en situasjon som kan beskrives av spillteori, hvor agenten deltar i et spill mot andre agenter. Innen spillteori brukes ofte løsningskonsept til å forklare hvordan rasjonelle agenter handler i slike situasjoner. Mange algoritmer for "multi-agent reinforcement learning" forsøker å finne et løsningskonsept, slik som Nash-ekvilibrium. I dette



Figur 2.9: Illustrasjon av "skip connection". *Til venstre:* en "skip connection" forbi en blokk. Blokkens inndata blir lagt til blokkens utdata. *Til høyre:* Flere etterfølgende blokker med "skip connections".

kapittelet har det også blitt beskrevet en metode for å finne Nash-ekvilibrium, kalt Fiktivt Spill.

Kapittelet inneholder også en beskrivelse av kunstige nevrale nettverk. Når man skal lære en datamaskin å spille spill som Diplomacy, er det vanlig å bruke kunstige nevrale nettverk til å blant annet lære en parametrisert policy.

I dette kapittelet har det også blitt sett på reinforcement learning. I reinforcement learning lærer en agent ved å forsøke å maksimere belønningen den får fra sine omgivelser. Reinforcement learning egner seg godt når man ikke har tilgjengelige eksempler på hvordan agenten skal oppføre seg, og når det er mulig å belønne agenten i henhold til hvor godt den presterer. Når en agent skal lære å spille Diplomacy fra selvspill, er det derfor naturlig å bruke metoder innen reinforcement learning.

Til slutt i kapittelet ble det sett på ulike former for, og bestanddeler i, kunstige nevrale nettverk. Graph Neural Networks (GNN) brukes til å behandle data som er organisert i grafer. Siden Diplomacy spilles på et kart over Europa, vil GNN kunne brukes når Diplomacy skal læres. Recurrent Neural Networks (RNN) og Long Short-Term Memory (LSTM) brukes til å behandle sekvensiell data. Embedding brukes til å lære vektorrepresentasjoner av kategorisk data. Batchnormalisering og skip connections er teknikker som forbedrer prestasjonen til dype nevrale nettverk. Dropout brukes til å unngå overfitting.

Kapittel 3

Relatert Arbeid

I dette kapittelet vil det bli presentert arbeid som er relatert til forskningsspørsmålene denne rapporten forsøker å svare på[1]. Det relaterte arbeidet omhandler både RL-metoder anvendt på Diplomacy, og andre spill som likner Diplomacy, samt metoder som lærer agenter ved selvspill. Kapittelet starter med en beskrivelse av protokollen brukt i den strukturerte litteraturgjennomgangen. Denne protokollen viser hvilke kriterier som har blitt brukt til å velge ut artiklene som er beskrevet i dette kapittelet.

3.1 Protokoll for strukturert litteraturgjennomgang (SLG)

For å finne relatert arbeid har metoden "Backward Snowballing" blitt brukt. Utgangspunktet for "Backward Snowballing" er et fåtall artikler som ansees å være relevant for forskningsspørsmålene som skal besvares. Fra referanselisten til disse artiklene, brukes forhåndsbestemte kriterier til å velge ut nye artikler. Denne prosessen gjentas med de nyvalgte artiklene, og fortsetter frem til man ikke finner flere artikler som samsvarer med de forhåndsbestemte kriteriene [Kofod-Petersen, 2021].

Tabell 3.1 viser hvilke kriterier som har blitt benyttet i utvelgelsen av relatert arbeid. Det har blitt brukt to ulike typer kriterier. Den ene typen er IC-kriterier. IC-kriteriene bestemmer om en artikkel skal velges fra en referanseliste eller ikke. De artiklene som oppfyller IC-kriteriene, blir vurdert opp mot den andre typen

[1] Store deler av dette kapittelets innhold var en del av forfatterens upubliserte fordypningsrapport [Hatlø, 2021]. Dette gjelder avsnitt 3.1, 3.2, 3.3, 3.4, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11 og 3.12. Avsnittene er skrevet om og noe av innholdet er endret.

kriterium, QC-kriterium. QC-kriteriene brukes til å vurdere kvaliteten på artiklene. Om en artikkel vurderes til å ha for lav kvalitet, vil den ikke bli valgt.

IC-kriteriene forsøker å gjenspeile det som er relevant for å besvare forskningsspørsmålene gitt i kapittel 1. Det første kriteriet, IC 1, følger direkte av forskningsspørsmål 1. For å besvare dette forskningsspørsmålet, er det viktig å finne ut hvilke maskinlæringsmetoder som tidligere har blitt brukt til å lære en agent å spille Diplomacy. Ved å ta med IC 1, vil en blant annet finne RL-metoder som har blitt brukt tidligere. På den måten kan man få en oversikt over hvilke metoder som har gitt gode resultater, og hvilke som ikke har det. Det vil også være relevant å få en oversikt over andre maskinlæringsmetoder som har vært brukt til å lære Diplomacy, ettersom det kan være mulig å kombinere disse med RL-metoder.

Det kan også være lurt å kartlegge metoder som har vært anvendt på andre, men liknende, problem som Diplomacy. Dette gjenspeiles i de tre neste IC-kriteriene. IC 2 er tatt med som et kriterium fordi Diplomacy er et spill for flere spillere. Det vil derfor være lurt å se på artikler som omhandler "Multi-Agent Reinforcement Learning". Ettersom målet, beskrevet i kapittel 1, er å lære en RL-agent å spille No-Press Diplomacy fra selvspill, er det lurt å se på artikler som omhandler RL-metoder hvor agenten lærer fra selvspill. Derfor er IC 3 tatt med som et kriterium. IC 4 er tatt med fordi metoder anvendt på spill som likner Diplomacy kanskje kan brukes på Diplomacy også.

Det antas at kriteriene nevnt ovenfor også kan brukes til å besvare forskningsspørsmål 2. Dette skyldes at nettverksarkitektur ofte er drøftet i artikler som oppfyller de ovennevnte kriteriene.

QC-kriteriene er valgt etter hva som skal til for at en artikkel har høy kvalitet. Det er viktig at resultatet kan reproduseres. Dette er viktig for å kunne verifisere resultatet. På den måten kan man være sikrere på at den beskrevne metoden kan anvendes. Om metoden beskrevet i artikkelen skal anvendes, gjør et reproduserbart resultat det enklere å implementere metoden. Derfor er kriteriet QC 1 tatt med. For å enklere kunne vurdere om metoden er riktig å ta i bruk, er det viktig at fremgangsmåten er godt begrunnet. Dette gjenspeiles i kriterium QC 2. Om artikkelen referer til relatert arbeid av høy kvalitet, vil det i mange tilfeller være enklere å forstå den fremgangsmåten som artikkelen beskriver. I tillegg vil det være enklere å finne annet relatert arbeid. QC 3 er derfor tatt med som et kriterium. Til slutt er det viktig at artikkelen har med en kritisk vurdering av løsningen, metoden eller resultatet den omhandler. Dette gjør det enklere å vurdere hvorvidt artikkelens innhold kan brukes til å besvare forskningsspørsmålene. Derfor er QC 4 tatt med som et kriterium.

I den strukturerte litteraturgjennomgangen er det tatt utgangspunkt i fire artikler som omhandler maskinlæringsmetoder brukt til å lære en agent å spille Diplomacy. Metodene beskrevet i disse artiklene har alle oppnådd gode resultater

ID	Kriterium
IC 1	Artikkelen omhandler maskinlæringsmetoder for å lære en agent å spille Diplomacy
IC 2	Artikkelen omhandler Multi Agent Reinforcement Learning
IC 3	Artikkelen omhandler RL-metoder hvor agenten lærer utelukkende fra selvspill
IC 4	Artikkelen omhandler RL brukt til å lære spill som likner Diplomacy.
QC 1	Resultatet kan reproduseres
QC 2	Fremgangsmåten er godt begrunnet
QC 3	Artikkelen refererer til relatert arbeid av høy kvalitet
QC 4	Artikkelen inneholder en kritisk vurdering av løsning/metode/resultat

Tabell 3.1: Kriterier brukt i strukturert litteraturgjennomgang. Tabellen er hentet fra forfatterens upubliserte fordypningsrapport [Hatlø, 2021].

og regnes som "the state of the art" blant maskinlæringsmetoder anvendt på Diplomacy.

3.2 No-Press Diplomacy: Modelling Multi-Agent Gameplay

Paquette et al. [2019] presenterer i sin artikkel en metode for å lære en agent Diplomacy. Metoden kombinerer supervised learning og reinforcement learning. Først trenes en modell på data samlet inn fra spill spilt av mennesker. Deretter brukes A2C-algoritmen til å forbedre modellen.

Siden spillbrettet i Diplomacy kan representeres som en graf, bruker Paquette et al. et graph neural network (GNN) til å kode informasjon om spillbrettet. I grafen representerer noder provinser og kyster, og kanter representerer naboskap mellom provinser. Ettersom samarbeid kan lønne seg, brukes GNN også til å kode informasjon om hvilke ordrer som ble gjort i forrige runde av spillet. På den måten er det mulig for agenten å lære seg å gjenkjenne samarbeidssituasjoner.

Output fra GNN brukes til å beregne policy. Til dette brukes Long Short-Term Memory (LSTM). Ettersom en handling gjort i én provins har betydning for hvilken handling som er lurt å gjøre i de andre provinsene, velges en ordre for én og én provins om gangen. Rekkefølgen av provinser når ordrer skal velges gjøres etter en topologisk sortering av provinsene. Informasjon om hvilken ordre som ble valgt i en provins, tas med når en ordre skal velges i den neste provinsen.

Paquette et al. testet ut en agent som bare hadde blitt trent ved supervised learning, og en agent som i tillegg var trent med reinforcement learning. Paquette et al. fant ingen signifikant forskjell mellom disse agentene. Sammenliknet med tidligere forsøk på å lære en maskin å spille Diplomacy, var agenten til Paquette et al. en betydelig forbedring. Paquette et al. målte også om agenten lærte seg å samarbeide. Dette var gjort ved å måle hvor stor andel av "SUPPORT"-ordrene som var gitt til en motstander som faktisk hjalp motstanderen. Agenten som var trent ved supervised learning viste seg å være best å samarbeide. Dette skyldes trolig at denne agenten i større grad gjenspeiler måten mennesker spiller på. Agenten trent ved reinforcement learning så ut til å avlære evnen til å gi effektiv støtte.

SLG-vurdering: Denne artikkelen er en av de fire artiklene, nevnt i avsnitt 3.1, som demonstrerer "the state of the art" blant maskinlæringsmetoder anvendt på Diplomacy. Artikkelen har en god beskrivelse av fremgangsmåten. I tillegg er tilhørende kildekode og datasett åpent tilgjengelig via GitHub[2]. Dette gjør resultatet reproduserbart. Fremgangsmåten er delvis begrunnet ved resonnement. GNN og LSTM er blant annet valgt fordi Diplomacy spilles på et kart. Andre deler av fremgangsmåten er valgt på bakgrunn av empiri. Dette gjelder for eksempel hva GNN skal ha som input. Artikkelen har referanser til relatert arbeid av høy kvalitet. Dette gjelder blant annet artiklene beskrevet i avsnitt 3.6 og i avsnitt 3.7. Artikkelen inneholder i liten grad en vurdering av mangler ved fremgangsmåten, men viser at RL-agenten er dårligere på å samarbeide enn agenten trent ved supervised learning.

3.3 Learning to Play No-Press Diplomacy with Best Response Policy Iteration

Anthony et al. [2020] har i sin artikkel tatt utgangspunkt i løsningen til Paquette et al. [2019]. I stedet for å bruke A2C for å trene opp agenten ved selvspill, har Anthony et al. valgt å bruke en utgave av policy iteration (PI).

I PI trenes en policy iterativt. I første iterasjon av algoritmen brukes en policy og en verdifunksjon som er trent på data fra menneskelig spill. I hver iterasjon av algoritmen velges en tidligere utgave av policyen. PI-algoritmen forsøker deretter å finne en ny policy, som er bedre enn den tidligere utgaven. Til dette brukes en "improvement operator". Basert på selvspill med den nye og forbedrede policyen lages et nytt datasett som agentens policynettverket trenes på.

Anthony et al. bruker en "improvement operator" kalt "Sampled Best Response" (SBR). SBR tar utgangspunkt i en base-policy og en kandidat-policy.

[2]GitHub-repository for Paquette et al. [2019]: `https://github.com/diplomacy/research`

Kandidat-policyen brukes til å gjøre et utvalg av handlinger for agenten. Base-policyen brukes til å gjøre et utvalg av felleshandlinger for agentens motstandere. Deretter beregnes den beste responsen blant kandidat-handlingene. Den beste responsen er den handlingen som gir størst forventet belønning, gitt at motstanderne spiller en av de utvalgte felleshandlingene. Verdifunksjonen, trent på data fra menneskelig spill, brukes til å finne agentens forventede belønning når den beste responsen skal beregnes.

Det finnes ulike måter å velge ut base-policyen og kandidat-policyen i SBR. Anthony et al. bruker tre metoder: Iterated Best Response (IBR), FPPI-1 og FPPI-2. FPPI-1 og FPPI-2 er begge basert på fiktivt spill. I IBR brukes forrige iterasjons policy både som base-policy og kandidat-policy. I FPPI-1 og FPPI-2 brukes forrige iterasjons policy som base-policy. I FPPI-1 velges det ut en tidligere iterasjon, t. Fra iterasjon t velges kandidat-policyen. En fordel med denne metoden sammenliknet med IBR, er at policyen som velges er mindre forutsigbar for motstanderne. En ulempe er at agenten ved å bruke en tidligere utgave av kandidat-policyen går glipp av den nye kunnskapen som finnes i siste iterasjons policy. I FPPI-2 gjøres det motsatt. Der blir base-policyen trukket fra den tidligere iterasjonen, t, mens kandidat-policyen er siste iterasjons policy.

Anthony et al. bruker mye av den samme nettverksarkitekturen som Paquette et al. En av forskjellene er at valg av handlinger gjøres med en GNN-dekoder i stedet for et LSTM-nettverk. I tillegg legges brettets tilstand i forrige runde til GNN-koderens input. Disse endringene ga et bedre resultat for SL-agenten til Anthony et al., sammenliknet med SL-agenten til Paquette et al. Anthony et al. bruker også output fra GNN-koderen som input til verdinettverket.

Endringen av RL-algoritme fra A2C til PI, gjorde at løsningen til Anthony et al. presterte bedre enn Paquette et al. sin løsning. Av de ulike utgavene av PI-algoritmen, var det FPPI-2 som presterte best.

SLG-vurdering: Denne artikkelen er en av de fire artiklene, nevnt i avsnitt 3.1, som demonstrerer "the state of the art" blant maskinlæringsmetoder anvendt på Diplomacy. Kun deler av kildekoden er tilgjengelig via GitHub[3]. Koden mangler blant annet en implementering av de ulike utgavene av PI-algoritmen. Deler av fremgangsmåten er godt beskrevet, men for å forstå enkelte deler av fremgangsmåten er man avhengig av å se på artikler som beskriver liknende metoder. Dette gjør det vanskeligere å reprodusere resultatet til Anthony et al. Begrunnelsen for fremgangsmåten er god, men inneholder i liten grad en kritisk vurdering av fremgangsmåten eller resultatet. Artikkel beskrevet i avsnitt 3.8, er et eksempel på artikler av høy kvalitet som Anthony et al. referer til.

[3]GitHub-repository for Anthony et al. [2020]: `https://github.com/deepmind/diplomacy`

3.4 Human-Level Performance in No-Press Diplomacy via Equilibrium Search

Gray et al. [2021] beskriver i sin artikkel en fremgangsmåte for å oppnå spill på menneskelig nivå i Diplomacy. Fremgangsmåten er basert på arbeidet til Paquette et al. [2019]. I likhet med Paquette et al. og Anthony et al. [2020], har Gray et al. trent opp et policy- og verdinettverk med data fra spill spilt av mennesker. Men i stedet for å ha nettverkene som utgangspunkt for reinforcement learning, bruker Gray et al. nettverkene sammen med en søketeknikk kalt "regret matching".

For hver runde i spillet brukes "regret matching" til å finne en ny, midlertidig policy. "Regret matching"-algoritmen starter med at policynettverket brukes til å velge ut N kandidathandlinger for hver agent. Dette er handlinger som i henhold til policynettverket har størst sannsynlighet for å bli valgt av agentene. Deretter oppdateres en midlertidig policy for hver agent iterativt. Den midlertidige policyen er i første iterasjon en uniform sannsynlighetsfordeling over agentens kandidathandlinger. I hver iterasjon av algoritmen velger agentene en handling etter sine midlertidige policyer. For å finne ut hvor bra de utvalgte handlingene er for agentene, simuleres først en runde i spillet med de utvalgte handlingene. Deretter simuleres noen runder hvor agentene antas å spille etter policynettverket. Til slutt brukes verdinettverket til å estimere hvor bra agentenes utvalgte handlinger var. Dette estimatet brukes til å oppdatere agentenes midlertidige policyer. Når alle iterasjonene er gjennomført, brukes den midlertidige policyen til å velge en handling i det virkelige spillet.

Gray et al. har tatt utgangspunkt i nettverksarkitekturen til Anthony et al.. Den viktigste forskjellen mellom nettverksarkitekturene er hvordan policyen beregnes. For å beregne policyen, bruker Gray et al. en multiplikasjon av output fra LSTM og en lært koding for hver handling. Resultatet av multiplikasjonen brukes til å beregne sannsynlighetene som er assosiert med de forskjellige handlingene.

I spill mot tilsvarende agenter som Anthony et al. og Paquette et al. lagde, presterte agenten til Gray et al. best. I tillegg ble løsningen til Gray et al. testet mot spillere på nettstedet `webdiplomacy.net`. Agenten til Gray et al. viste seg å spille like godt som de 2% beste spillerne på dette nettstedet.

SLG-vurdering: Denne artikkelen er en av de fire artiklene, nevnt i avsnitt 3.1, som demonstrerer "the state of the art" blant maskinlæringsmetoder anvendt på Diplomacy. Kildekoden er åpent tilgjengelig via GitHub[4]. I tillegg er fremgangsmåten godt beskrevet. Dette gjør det enkelt å reprodusere resultatet. Artikkelen referer også til arbeid av høy kvalitet. Dette gjelder blant annet artikkelen beskrevet i avsnitt 3.9. Fremgangsmåten kunne vært bedre begrunnet.

[4]GitHub-repository for Gray et al. [2021]: `https://github.com/facebookresearch/diplomacy_searchbot`

Artikkelen beskriver noen mangler ved løsningen. Blant annet spiller agenten dårligere i sluttspillet, noe som kan skyldes at det finnes for lite treningsdata fra sluttspill.

3.5 No-Press Diplomacy from Scratch

Bakhtin et al. [2021] presenterer i sin artikkel en algoritme for å lære en agent å spille Diplomacy utelukkende fra selvspill.

Til å lære agenten å spille Diplomacy, anvender Bakhtin et al. en algoritme kalt DORA. DORA består blant annet av en annen algoritme, kalt Deep Nash Value Iteration. Denne brukes til å lære en verdifunksjon og en policy. Til å oppdatere verdifunksjonen, bruker Deep Nash Value Iteration blant annet et beregnet Nash-ekvilibrium. Siden antallet mulige handlinger for hvert trekk er svært stort i Diplomacy, er det ikke mulig å beregne et eksakt Nash-ekvilibrium. Derfor brukes et policynettverk til å plukke ut kandidathandlinger som skal inngå i beregningen av Nash-ekvilibriet. Til å beregne Nash-ekvilibriet anvendes "regret matching"-algoritmen, som også ble brukt av Gray et al. [2021]. Etter at Nash-ekvilibriet er beregnet, brukes dette til selvspill. Under selvspillet velges handlinger etter sannsynlighetsfordelingen bestemt av Nash-ekvilibriet. I tillegg gjøres det noen ganger tilfeldige handlinger for å utforske handlinger som ikke er en del av i Nash-ekvilibriet. Data fra selvspillet brukes til å oppdatere policy- og verdinettverket.

Bakhtin et al. har tatt utgangspunkt i nettverksarkitekturen til Gray et al., men har gjort noen endringer. Blant annet er GNN-koderen byttet ut en "transformer"-koder. Nettverket er også delt i to separate nettverk: ett for policy og ett for verdi. Dette er i motsetning til nettverkene til Gray et al. og Anthony et al. [2020], hvor koderen er en del av både policy- og verdinettverket.

Agenten har blitt trent opp på en utgave av Diplomacy for to spillere, i tillegg til standardutgaven med sju spillere. I utgaven for to spillere oppnådde agenten et høyere ferdighetsnivå enn mennesker. I spillet for sju spillere var DORA-agenten bedre enn andre agenter når seks DORA-agenter spilte mot en annen agent. Når én kopi av DORA-agenten spilte mot seks andre agenter, presterte DORA-agenten dårligere enn de andre agentene. Ettersom de andre agentene helt eller delvis var trent opp på data fra spill spilt av mennesker, konkluderer Bakhtin et al. med at DORA-agenten lærer seg å spille et ekvilibrium som mennesker normalt ikke spiller.

SLG-vurdering: Denne artikkelen er en av de fire artiklene, nevnt i avsnitt 3.1, som demonstrerer "the state of the art" blant maskinlæringsmetoder anvendt på

Diplomacy. Kildekoden er tilgjengelig via GitHub[5]. I tillegg er fremgangsmåten godt beskrevet, blant annet med en detaljert pseudokode. Derimot brukte Bakhtin et al. store beregningsressurser for å oppnå resultatet. Resultatet er derfor enkelt å reprodusere, dersom man har tilgang til nok datakraft. Artikkelen referer til arbeid av høy kvalitet, deriblant artikkelen beskrevet i avsnitt 3.13. Artikkelen innholder i liten grad en kritisk vurdering av egen fremgangsmåte, og ingen forslag til forbedringer.

3.6 Learning a Game Strategy Using Pattern-Weights and Self-Play

Shapiro et al. [2002] sin fremgangsmåte bruker forhåndsdefinerte mønstre til å lære en agent å spille Diplomacy. Mønstrene brukes til å gjenkjenne ulike situasjoner, og beskrives blant annet av hvilken type ordre som er gitt, hvilken type enhet som mottar ordren eller informasjon om hvem som eier en naboprovins. Agenten lærer ved hjelp av selvspill og TD-learning vekter som assosieres med de ulike mønstrene. Beslutning om hvilken ordre agenten skal gjennomføre, tas på bakgrunn av vektenes størrelser. Først ser agenten på hvilke ordre som passer med hvilke mønstre. For hver ordre regnes det ut en verdi basert på mønstrenes vekter. Deretter velges en av ordrene som har fått tildelt høyest verdi. Vektene blir oppdatert etter hvor godt resultatet av handlingene er. Suksessen til en handling bestemmes blant annet av hvor mange forsyningssentre agenten har kontroll over etter at handlingen ble utført.

Shapiro et al. sin agent ble målt på hvor lik agentens åpningstrekk var menneskers åpningstrekk. Agenten gjorde i mange tilfeller like trekk som et menneske ville gjort, men ikke alltid.

SLG-vurdering: Denne artikkelen er tatt med fordi den blant annet oppfyller kriterium IC 1. Fremgangsmåten er godt beskrevet, så selv om det ikke finnes åpent tilgjengelig kildekode, antas det at resultatet er enkelt å reprodusere. Artikkelen inneholder få referanser til andre artikler. Artikkelen beskriver mangler ved fremgangsmåten. Blant annet kreves det veldig mange spill for at agenten skal lære gode strategier.

[5]GitHub-repository for Bakhtin et al. [2021]: `https://github.com/facebookresearch/diplomacy_searchbot`

3.7 Learning with Opponent-Learning Awareness

Foerster et al. [2018] presenterer i sin artikkel en ny policy-gradient-metode for situasjoner hvor flere agenter samhandler. Hver agent forsøker å maksimere en funksjon av alle agenters parametere. Denne funksjonen, J, er et mål på hvor godt agenten presterer. Funksjonens parametere tar også hensyn til at motstanderne lærer. I en setting med to agenter vil agent 1 forsøke å maksimere følgende funksjon: $J_1(\theta_1, \theta_2 + \Delta\theta_2)$. Her representerer $\theta_2 + \Delta\theta_2$ parameterverdiene til agent 2 etter at agent 2 har gjennomført et læringssteg. Agent 1 går ut i fra at motstanderen, agent 2, ikke tar hensyn til at agent 1 lærer.

Foerster et al. foreslår både en versjon hvor agenten har tilgang på motstandernes faktiske parameterverdier, og en versjon hvor agentene lærer seg et estimat av disse. Estimatet gjøres ved å observere hvilke handlinger motstanderen gjør. Basert på observasjonene antar man at motstanderens parameterverdier er verdiene som med størst sannsynlighet forårsaker motstanderens oppførsel.

Foerster et al. har testet sin agent på flere spill, blant annet "Iterated Prisoner's Dilemma". Resultatet viser at to agenter som spiller mot hverandre, og som bruker Foerster et al. sin algoritme, lærte seg en "tit-for-tat"-strategi. Dette viser at metoden til Foerster et al. [2018] kan lære agenter å samarbeide.

SLG-vurdering: Denne artikkelen oppfyller kriterium IC 2. Kildekoden er tilgjengelig via GitHub[6], og resultatet antas derfor å være enkelt å reprodusere. Fremgangsmåten er også godt begrunnet, blant annet ved en detaljert utledning. Det nevnes noen mangler ved fremgangsmåten. En av disse er at man antar at alle agenter har lik oppbygging av sine policynettverk.

3.8 Deep Reinforcement Learning from Self-Play in Imperfect-Information Games

Heinrich og Silver [2016] sin artikkel omhandler en metode for å lære en agent å lære fra selvspill i spill med imperfekt informasjon. I et spill med imperfekt informasjon har ikke spillerne all informasjon om spillets tilstand. Dette gjelder blant annet i poker, hvor en spiller ikke har oversikt over motstandernes kort.

Heinrich og Silver beskriver i sin artikkel en algoritme for fiktivt spill kalt "Neural Fictitious Self-Play" (NFSP). I NFSP samles erfaring som agenten gjør seg i to forskjellige minner. Det ene minnet brukes til å trene opp agenten ved reinforcement learning. Dataen i dette minnet blir brukt til å trene opp et Q-nettverk. Dette nettverket blir brukt til å velge handlinger grådig. Det vil si at handlingen som velges er handlingen med størst handlingsverdi. Det andre minnet

[6] GitHub-repository for Foerster et al. [2018]: `https://github.com/alshedivat/lola`

brukes til å trene opp agenten ved supervised learning. Dette minnet blir brukt til å trene opp et policynettverk. Når agenten spiller, velges det tilfeldig om en handling skal velges etter Q-nettverket eller policynettverket.

Heinrich og Silver testet NFSP på to pokerspill: "Leduc Poker" og "Limit Texas Hold'em" Resultatet av testene var at NFSP lærte et tilnærmet Nash-ekvilibrium. I "Limit Texas Hold'em" fikk NFSP-agenten et ferdighetsnivå som overgår mennesker. Sammenliknet med en mye brukt algoritme kalt "Deep Q-network", har NFSP den fordelen at handlinger velges etter en sannsynlighetsfordeling. På den måten blir det vanskelig for motstanderne å forutse hvilke handlinger agenten vil gjøre.

SLG-vurdering: Denne artikkelen oppfyller blant annet kriterium IC 3. Koden til Heinrich og Silver sitt arbeid er ikke offentlig tilgjengelig, men ettersom fremgangsmåten er godt beskrevet antas resultatet å være reproduserbart. Artikkelen inneholder i liten grad en kritisk vurdering av resultatet, og inneholder heller ikke forslag til forbedringer. Det refereres til relatert arbeid av høy kvalitet.

3.9 Superhuman AI for multiplayer poker

Brown og Sandholm [2019] beskriver i sin artikkel en pokerspillende agent, kalt Pluribus. Pluribus ble trent til å spille "No-limit Texas Hold'em" for seks spillere, og oppnådde et høyere ferdighetsnivå enn profesjonelle spillere. Pluribus lærte fra selvspill.

På grunn av det store antallet mulige handlinger i poker, benytter Brown og Sandholm seg av en teknikk kalt abstraksjon. Abstraksjon sørger for at handlinger som likner hverandre blir slått sammen. På den måten reduseres kompleksiteten i spillet. I poker kan dette for eksempel være å slå sammen handlinger hvor summene som agenten kan by er like. I tillegg kan spillets tilstand abstraheres ved å slå sammen kortkombinasjoner som er like. Ettersom abstraksjon kan føre til at viktig informasjon sees bort i fra, brukes denne teknikken bare når agenten ser på fremtidige situasjoner.

Pluribus lærer en policy ved hjelp av en algoritme kalt "Monte Carlo Counterfactual Regret Minimization" (MCCFR). I MCCFR oppdateres en agents policy ved å første simulere et spill. Deretter ser man på hvilke alternative handlinger agenten kunne tatt i løpet av spillet. For hver alternativ handling simuleres et delspill som starter i situasjonen hvor den alternative handlingen kunne tas, og ved at agenten tar den alternative handlingen. Deretter oppdateres agentens policy etter hvor mye bedre eller dårligere agenten kunne ha gjort det ved å velge de alternative handlingene. De alternative handlingene som ville ført til et bedre resultat, får en høyere sannsynlighet, mens det motsatte skjer for de alternative handlingene som ville gitt et dårligere resultat. Denne prosedyren gjentas for nye

alternative handlinger agenten kunne gjort i simuleringen av delspillet. I faktisk spill brukes policyen i kombinasjon med søk.

SLG-vurdering: Artikkelen er inkludert blant annet fordi den oppfyller kriterium IC 3. Fremgangsmåten er detaljert beskrevet, og selv om det ikke finnes åpen kildekode tilgjengelig, antas det at resultatet er lett å reprodusere. Fremgangsmåten er begrunnet med de resultater den har gitt. Artikkelen mangler en kritisk vurdering av fremgangsmåten.

3.10 Modelling Others using Oneself in Multi-Agent Reinforcement Learning

Raileanu et al. [2018] viser i sin artikkel hvordan en agent kan lære seg målet til andre agenter. Når en agent kjenner til hva andre agenter i omgivelsene ønsker seg, blir det lettere for å agenten å ta en riktig beslutning om hvordan den skal handle.

Raileanu et al. bruker det samme nettverket til å beregne policy og tilstandsverdi. Input til nettverket er observert tilstand, agentens eget mål, samt et estimat på motstanderens mål. I tillegg til å beregne policy og tilstandsverdi, brukes dette nettverket til å forbedre estimatet på motstanderens mål. For å forbedre estimatet på motstanderens mål, bytter agentens eget mål og estimatet på motstanderens mål plass i rekkefølgen av inndata til nettverket. På den måten forsøker agenten å beregne policy og tilstandsverdi for motstanderen. Deretter observerer agenten hvilke handlinger motstanderen faktisk gjør, og endrer sitt estimat på motstanderens mål slik at nettverket gir en større sannsynlighet for å velge de observerte handlingene.

Raileanu et al. testet læringsalgoritmen på tre ulike spill, som krever ulik grad av samarbeid mellom spillerne. I mange tilfeller klarte agenten å lage et estimat som var nært motstanderens virkelige mål.

SLG-vurdering: Artikkelen er inkludert fordi den oppfyller kriterium IC 2. Det finnes ikke åpen kildekode, men fremgangsmåten er detaljert beskrevet. Resultatet antas derfor å være enkelt å reprodusere. Fremgangsmåten er godt begrunnet, og artikkelen referer til relatert arbeid av høy kvalitet. Blant de refererte artiklene finner man artikkelen beskrevet i avsnitt 3.12. Artikkelen inneholder en kritisk vurdering av fremgangsmåten, hvor det blant annet påpekes at metoden krever mye trening for å fungere.

3.11 Multi-Agent Actor-Critic for Mixed Cooperative Competitive Environments

Lowe et al. [2020] presenterer i sin artikkel en metode for å få actor-critic-metoder til å prestere bedre i omgivelser med flere agenter. Treningsresultatet til actor-critic-metoder brukt i omgivelser med flere agenter har ofte høy varians. Dette skyldes at handlingene til de andre agentene har betydning for hvilken tilstand omgivelsene havner i etter at agenten har utført en handling. Policyen til de andre agentene vil kunne endre seg under treningen.

Løsningen til Lowe et al. består av å bruke en sentralisert critic under trening av policyen. Den sentraliserte critic-en bruker informasjon om motstandernes policy til å beregne tilstandsverdier, noe som gjør det enklere å trene agenten. Informasjon om motstandernes policyer vil kun være tilgjengelig for agenten under trening. Alternativt er det mulig for agenten å lage et estimat av motstandernes policyer basert på observasjoner av motstandernes oppførsel. For at agenten enklere skal kunne håndtere endringer i motstandernes policyer, brukes K forskjellige del-policyer. Disse del-policyene trenes opp i forskjellige episoder, og brukes deretter i treningen av agentens faktiske policy.

Lowe et al. testet sin løsning på flere ulike spill. Noen av spillene krevde kommunikasjon og samarbeid mellom agentene, mens agentene i andre spill var motstandere. Testresultatene viser at fremgangsmåten til Lowe et al. prestere bedre enn A2C i disse spillene.

SLG-vurdering: Denne artikkelen er tatt med fordi den oppfyller kriterium IC 2. Fremgangsmåten er godt begrunnet. Tilhørende kildekode er åpent tilgjengelig via GitHub[7], i tillegg er fremgangsmåten godt beskrevet. Resultatet antas derfor å være enkelt å reprodusere. Artikkelen har med en kritisk vurdering av fremgangsmåten, og forslag til fremtidig arbeid.

3.12 Asynchronous Methods for Deep Reinforcement Learning

Mnih et al. [2016] beskriver i sin artikkel asynkrone versjoner av fire populære RL-metoder: one-step SARSA, one-step Q-learning, n-step Q-learning og A2C.

Metoden til Mnih et al. går i hovedsak ut på å la agenter operere parallelt. I stedet for å trene direkte opp sitt eget policy- og verdinettverk, akkumulerer de parallelle agenten gradienter for sine nettverk. Med jevne mellomrom oppdateres et globalt nettverk med de akkumulerte gradientene. Etter at de globale nettver-

[7] GitHub-repository for Lowe et al. [2020]: `https://github.com/openai/maddpg`

kene er oppdatert, settes de parallelle agentenes nettverksparametere til å være lik de globale nettverksparameterne.

En av utfordringene med de ikke-parallelle metodene nevnt ovenfor er at hver oppdatering av policyen gjøres med data fra et lite antall observasjoner. Observasjonene er i tillegg ikke uavhengige av hverandre, ettersom de f.eks. er hentet fra etterfølgende runder i et spill. Agenten får i mindre grad utforsket ulike tilstander av spillet, noe som kan gi høyere varians i treningsresultatet. Ved å trene opp flere agenter parallelt, vil man få utforsket flere tilstander av spillet. Dette igjen vil føre til at treningen av de globale nettverkene blir mer stabil. I tillegg vil tiden det tar å trene de globale nettverkene reduseres.

Atari-spill ble brukt til å teste de asynkrone metodene. Resultatene viste at de metodene lærte å spille spillene raskere på 16 CPU-kjerner enn en annen populær metode kalt "Deep Q-Network" gjorde på en GPU. Den asynkrone actor-critic-metoden lærte å spille spillene dobbelt så raskt som metoder som ansees som "the state of the art".

SLG-vurdering: Artikkelen er tatt med fordi den oppfyller kriterium IC 3. Tilhørende kildekode er ikke tilgjengelig, men fremgangsmåten er godt beskrevet. Det antas derfor at resultatet er enkelt å reprodusere. Fremgangsmåten er også godt begrunnet. Artikkelen inkluderer en kritisk vurdering av fremgangsmåten, og forfatterne kommer med forslag til forbedringer.

3.13 IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

Espeholt et al. [2018] sin artikkel omhandler en metode, kalt IMPALA, brukt til å lære én enkelt agent å spille flere typer spill. Til dette brukes en sentralisert agent som lærer fra data generert av andre agenter. I tillegg presenterer Espeholt et al. en off-policy-versjon av actor-critic, kalt V-trace.

IMPALA likner den asynkrone A2C-algoritmen beskrevet i avsnitt 3.12. En av forskjellene mellom metodene er at de parallelle agentene sender observerte tilstander, utførte handlinger og mottatte belønninger i stedet for å sende gradienter til den sentraliserte agenten. Den sentraliserte agenten lærer deretter fra dataen den mottar og sender oppdaterte parameterverdier tilbake til de parallelle agentene.

Ettersom agentens forventede belønning under policyen som produserer data, kan være annerledes enn agentens forventede belønning under policyen som trenes, trenger man å justere verdifunksjonen slik at den representerer riktig forventningsverdi. I V-trace brukes "importance sampling" for å oppnå dette.

IMPALA ble testet på diverse Atari-spill. IMPALA oppnådde gode resultater og var mer effektiv til å lære enn den asynkrone A2C-algoritmen beskrevet i avsnitt 3.12.

SLG-vurdering: Denne artikkelen er inkludert fordi den oppfyller kriterium IC 3. Kildekoden for Espeholt et al. er tilgjengelig via GitHub[8]. Fremgangsmåten er også detaljert beskrevet. Dette gjør at resultatet trolig er enkelt å reprodusere. Artikkelen referer til arbeid av høy kvalitet, blant annet artikkelen beskrevet i avsnitt 3.12. Artikkelen har i liten grad en kritisk vurdering av fremgangsmåten.

3.14 Oppsummering

I dette kapittelet har fem forskjellige maskinlæringsmetoder for å lære en agent å spille Diplomacy blitt beskrevet. I metodene til Paquette et al. [2019], Anthony et al. [2020] og Gray et al. [2021] ble data fra spill spilt av mennesker brukt til å trene opp en agent ved supervised learning. Deretter forsøkte Paquette et al. og Anthony et al. å forbedre agenten med reinforcement learning. Bakhtin et al. [2021] lærte en agent å spille Diplomacy utelukkende fra selvspill, men agenten ble ikke bedre til å spille Diplomacy enn agentene til de ovennevnte forfatterne. Shapiro et al. [2002] sin metode med "pattern-weights" var ikke tilstrekkelig til å lære en agent å spille Diplomacy.

I Gray et al. sin løsning ble agentens policy- og verdinettverk brukt som utgangspunkt for søk etter en bedre policy. Siden denne metoden ikke anvender reinforcement learning, vil ikke en metode tilsvarende Gray et al. sin være egnet til å besvare forskningsspørsmål 1. Shapiro et al. [2002] sin metode vil heller ikke kunne brukes, da den ikke var tilstrekkelig til å lære en agent å spille Diplomacy.

I dette kapittelet har det også blitt sett på metoder som egner seg til "Multi Agent Reinforcement Learning". Lowe et al. [2020] viser i sin artikkel hvordan en agent kan forsøke å påvirke læringen til andre agenter. Denne algoritmen har tidligere vist å generere strategier for samarbeid. Raileanu et al. [2018] beskriver hvordan en agent kan lære seg - og nyttiggjøre kunnskapen om - andre agenters mål. Lowe et al. [2020] viser i sin artikkel hvordan en sentralisert critic med informasjon om motstandernes policy, gjør det enklere å trene en agent i "Multi Agent"-settinger.

Paquette et al. nevner i sin artikkel at A2C-algoritmen gjorde Diplomacy-agenten dårligere til å gi effektive "SUPPORT"-ordre. Paquette et al. foreslår derfor å anvende Lowe et al. sin LOLA-algoritme på Diplomacy. Ingen av løsningene til Anthony et al., Gray et al. eller Bakhtin et al. har hatt et spesielt fokus på å

[8]GitHub-repository for Espeholt et al. [2018]: `https://github.com/deepmind/scalable_agent`

oppnå samarbeid mellom agenter. Ettersom agenter som evner å inngå samarbeid med hverandre vil ha en stor fordel i Diplomacy, er det derfor en mulighet for at LOLA-algoritmen har en fordel når Diplomacy skal læres.

En av ulempene ved metoden til Raileanu et al. [2018] er at det er vanskelig å finne en meningsfull representasjon av agentens mål. I tillegg har alle agenter i Diplomacy det samme målet; De ønsker å vinne ved å få kontroll over halvparten av forsyningssentrene. Det kan tenkes at Raileanu et al. sin metode kan brukes til å lære delmål i Diplomacy, men siden disse trolig endres hyppig, antas de å være vanskelige å lære.

Metoder for selvspill, og "reinforcement learning" brukt til å lære spill som likner Diplomacy, har også blitt beskrevet i dette kapittelet. Heinrich og Silver [2016] beskriver en metode som kombinerer Q-nettverk og policynettverk til å lære agenter å spille poker. Brown og Sandholm [2019] sin artikkel omhandler også poker, og beskriver en metode som lærer ved å betrakte alternative handlinger i spilltreet. Mnih et al. [2016] og Espeholt et al. [2018] beskriver to beslektede måter for sentralisert læring. I Mnih et al. [2016] sin løsning brukes agent-prosesser til å generere gradienter som brukes til å oppdatere sentraliserte policy- og verdinettverk. I Espeholt et al. [2018] sin løsning brukes agent-prosessene til å generere data som sendes til å en sentralisert actor-critic, som lærer fra dataen.

Ettersom spilltreet til Diplomacy er veldig stort, er det ikke realistisk å bruke tilsvarende metode som Brown og Sandholm. Da det finnes svært mange mulige trekk pr. runde i Diplomacy, er det heller ikke realistisk å lære en Q-funksjon, slik som i metoden til Heinrich og Silver.

IMPALA [Espeholt et al., 2018] og asynkron actor-critic [Mnih et al., 2016] kan brukes for å forbedre stabiliteten i treningen, i tillegg til å redusere treningstiden.

Artiklene til Paquette et al., Anthony et al., Gray et al. og Bakhtin et al. kan brukes til å finne svar på forskningsspørsmål 2. Policy- og verdinettverket til disse forfatterne har alle elementer som kan anvendes i dette prosjektet.

Kapittel 4

Metode

I dette kapittelet beskrives fremgangsmåten brukt til å oppnå resultatene i kapittel 5. Dette inkluderer valg av spillbrett i Diplomacy, en beskrivelse av de anvendte reinforcement learning algoritmene, samt en oversikt over arkitekturen til policy- og verdinettverket.

4.1 Omgivelser

For en agent som spiller Diplomacy, er omgivelsene spillbrettet det spilles på og agentens motstandere. I tillegg har man belønningssignalet som forteller hvor bra agenten presterer. I denne delen vil det bli sett på hvilket spillbrett som er valgt, hvilken grense som er satt for lengde på spillet, samt hvordan belønningssignalet er designet.

4.1.1 Spillbrett

Den samme spillmotoren[1] som Paquette et al. [2019] benyttet seg av, har blitt brukt i dette prosjektet. Spillmotoren tilbyr flere ulike typer spillbrett som varierer både når det kommer til antall provinser og antall spillere som kan delta. Paquette et al., Anthony et al. [2020] og Gray et al. [2021] har alle brukt det originale brettet, beskrevet i avsnitt 2.1, i sine artikler.

En av årsakene til at det originale spillbrettet er en interessant utfordring innen reinforcement learning er at spillet har et svært stort spilltre. Det er ikke bare antall mulige tilstander som er stort; Antall mulige handlinger pr. runde kan overstige 10^{20} [Bakhtin et al., 2021]. En annen utfordring med det originale spillbrettet er at sju spillere spiller mot hverandre. Flere agenter innebærer flere

[1] GitHub-repository for Diplomacy-spillmotor: `https://github.com/diplomacy/diplomacy`

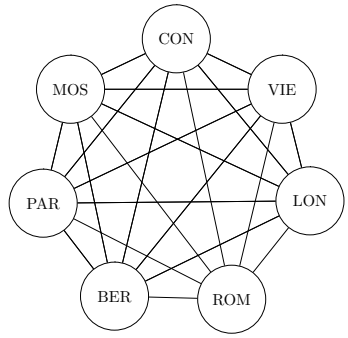
policyer som kan endre seg, og det blir vanskeligere å lære dynamikken i spillet. Om policy-gradient-metoder brukes, vil dette kunne gi høyere varians i læringen [Lowe et al., 2020].

Bakhtin et al. [2021] bruker i sin artikkel en variant av Diplomacy for to spillere kalt "France vs Austria". Som navnet tilsier er Frankrike og Østerrike de eneste nasjonene som spiller på brettet. Selve brettet er likt som i originalutgaven. Det vil si at det er like mange provinser og forsyningssentre, og det er fremdeles slik at en nasjon må ha kontroll over 18 forsyningssentre for å stikke av med seieren. Dette medfører at "France vs Austria" vil være et mindre spill i starten, med færre enheter på brettet og færre mulige handlinger pr. spiller. Senere i spillet vil antall enheter kunne bli like stort som i originalutgaven. Bakhtin et al. valgte å bruke denne utgaven av spillet fordi den gir de samme utfordringen med et stort spilltre som i originalutgaven, samtidig som man unngår utfordringene med mange spillere. Av samme grunn har "France vs Austria" blitt brukt i dette prosjektet.

I spillmotoren finner man også et brett kalt "Pure". "Pure" består av kun 7 provinser, som alle er forsyningssentre. Alle provinsene er naboer til hverandre, og i starten av spillet har hver nasjon hver sin armé i en av provinsene. Figur 4.1 illustrerer hvordan spillbrettet "Pure" er bygd opp. "Pure" er på mange måter en motsetning til "France vs. Austria". "Pure" har et betydelig mindre spilltre, selv om antall mulige handlinger pr. trekk fremdeles kan være stort. Men fordi det er et spill for sju spillere, har man de samme utfordringene med mange spillere som i originalutgaven. På grunn av forskjellen i størrelse, kan det forventes at treningen av en agent som spiller "Pure" krever mindre beregningsressurser enn treningen av en agent som spiller "France vs. Austria". I "Pure" er man i starten avhengig av å få støtte fra en medspiller for å kunne avansere. Denne utgaven passer derfor til å teste i hvilken grad LOLA-algoritmen, som ble nevnt i avsnitt 3.7 og som blir beskrevet i større detalj i avsnitt 4.2, genererer samarbeid mellom agentene. På grunn av de nevnte egenskapene, har også "Pure" blitt brukt i dette prosjektet.

4.1.2 Maksimal Lengde på Spill

I teorien kan man oppleve at en runde Diplomacy kan vare svært lenge. Når man har utrente agenter som spiller mot hverandre, er det ikke utenkelig at alle agentene kun velger "SUPPORT"-ordrer. En kan dermed ende opp med et spill som aldri tar slutt. For å unngå dette har det blitt satt en maksgrense på hvor lenge et spill kan vare. Anthony et al. [2020] estimerte at gjennomsnittslengden på et spill er 20 ordrefaser. Det har blitt valgt å sette den maksimale lengden på et spill til det dobbelte av gjennomsnittslengden. Det antas at dette gjør det mulig for agenten å lære fra spill som utvikler seg på mange ulike måter. For



Figur 4.1: "Pure"-utgaven av Diplomacy. Spillbrettet består av forsyningssentrene Konstantinopel, Wien, London, Roma, Berlin, Paris og Moskva.

"Pure" finnes det ikke data som gjør det mulig å estimere en gjennomsnittslengde. Ettersom "Pure" er et betydelig mindre spill enn spillene som spilles på det originale spillbrettet, settes maksgrensen for "Pure"-spill til 10 ordrefaser.

4.1.3 Belønningssignal

Ettersom en RL-agent forsøker å maksimere belønningen den mottar, vil hvordan en velger å belønne agenten ha innvirkning på hvilket mål agenten faktisk har. En agent som spiller Diplomacy ønsker man naturligvis at skal lære seg å vinne spillet. Designet av belønningssignalet bør derfor gjenspeile dette. Ettersom kun én spiller kan gå seirende ut av spillet, gis seier en belønning på 1. Tap gis en belønning på 0. I og med at spillet har fått en maksimal lengde, vil man også kunne oppleve situasjoner hvor spillet ender uten at en vinner kan kåres. Situasjonene hvor en vinner ikke kan kåres, er ikke alle like. I noen situasjoner vil noen av spillerne være nærmere å ha vunnet enn andre spillere. Man trenger derfor et belønningssignal som favoriserer spillerne som er nærmest å ha vunnet. En vanlig løsning for slike situasjoner er å bruke en "Sum-of-Squares"-score (SoS). Denne metoden er blant annet brukt i onlinespill på nettstedet `webdiplomacy.net`, og i arbeidet til Gray et al. [2021]. SoS-score for spiller i er vist i likning 4.1. c_i er antall forsyningssentre spiller i har kontroll over.

$$SoS_i = \frac{c_i^2}{\sum_j c_j^2} \tag{4.1}$$

4.2 RL-algoritmer

Basert på den strukturerte litteraturgjennomgangen og artiklene beskrevet i kapittel 3, ble det konkludert med at det ville være interessant å se på hvor godt LOLA-algoritmen egner seg til å lære en agent å spille Diplomacy. Sammenliknet med de andre algoritmene som tidligere har blitt brukt til dette formålet, tar LOLA-algoritmen hensyn til at motstandere også lærer. En hypotese er derfor at dette gir LOLA-algoritmen en fordel over de andre algoritmene. På bakgrunn av dette har LOLA-algoritmen blitt sammenliknet med A2C, som ble beskrevet i kapittel 2. A2C har en del likhetstrekk med LOLA, men tar ikke hensyn til motstandernes læring. LOLA-algoritmen ble kort beskrevet i avsnitt 3.7. I et forsøk på å øke hastigheten og stabiliteten på læringen har deler av IMPALA-algoritmen (avsnitt 3.13) blitt brukt. Videre følger en mer detaljert beskrivelse av LOLA og IMPALA.

4.2.1 LOLA

Learning with Opponent Learning Awareness (LOLA) er en Policy-Gradient-metode [Foerster et al., 2018]. I LOLA-algoritmen er funksjonen som skal maksimeres ikke bare en funksjon av agenten, i, sine parametere, men også motstandernes parametere ($J_i(\theta_1, ..., \theta_i, ..., \theta_n)$). En LOLA-agent forsøker å maksimere J-funksjonen etter at motstanderne har gjennomført et læringssteg ($J_i(\theta_1 + \Delta\theta_1, ..., \theta_i, ..., \theta_n + \Delta\theta_n)$). Likning 4.2 viser hvordan parametrene til agent i skal oppdateres i henhold til LOLA-algoritmen. En utledning av denne likningen kan sees i Foerster et al. [2018] sin artikkel. α_i er agent i sin læringsrate. Gradientene er kolonnevektorer. Det første leddet i likningen tilsvarer den opprinnelige policy-gradienten. De resterende leddene beskriver hvordan agent i kan påvirke motstandernes policy, og på den måten forbedre sin egen prestasjon. Den første faktoren i hvert ledd viser hvordan en endring i motstanderens parametere har innvirkning på agentens prestasjoner. Den andre faktoren viser hvordan en endring i egne parametere har innvirkning på læringen til motstanderen.

$$\Delta\theta_i = \nabla_{\theta_i} J_i(\theta_1, ..., \theta_i, ...\theta_n)\alpha_i + \sum_{j \in \{1,2,...,n\} \setminus \{i\}} (\nabla_{\theta_j} J_i(\theta_1, ..., \theta_i, ...\theta_n))^T \nabla_{\theta_i} \nabla_{\theta_j} J_j(\theta_1, ..., \theta_i, ...\theta_n)\alpha_i\alpha_j \quad (4.2)$$

En oppdatering av agentens policy i henhold til likning 4.2, krever at agenten har tilgang til gradientene til motstandernes J-funksjoner. Om disse ikke er tilgjengelig, kan man bruke tilnærminger. Tilnærmingene er beskrevet i likningene 4.3, 4.4 og 4.5. I likningene er π_i policyen til agent i. $A_{i,t}$ er en handling gjort av agent i i tidspunkt t. S_t er tilstanden til omgivelsene i tidspunkt t. $G_{i,t}$

er summen av fremtidige neddiskonterte belønninger, definert i likning 2.1, for agent i. I likhet med de "ordinære" policy-gradient-metodene beskrevet i avsnitt 2.4.2, kan $G_{i,t}$ byttes ut med $R_{i,t+1} + \gamma V_i(S_{t+1}) - V_i(S_t)$. På den måten får man en actor-critic-oppdatering av parametrene. En slik oppdatering er brukt i dette prosjektet.

$$\nabla_{\theta_i} J_i(\theta_1, ..., \theta_i, ...\theta_n) \approx \mathbf{E}[\sum_{t=0}^{T} \nabla_{\theta_i} log\pi_i(A_{i,t} \mid S_t)\gamma^t G_{i,t}] \tag{4.3}$$

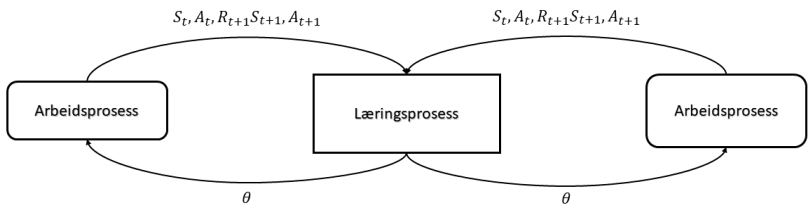
$$\nabla_{\theta_j} J_i(\theta_1, ..., \theta_i, ...\theta_n) \approx \mathbf{E}[\sum_{t=0}^{T} \nabla_{\theta_j} log\pi_j(A_{j,t} \mid S_t)\gamma^t G_{i,t}] \tag{4.4}$$

$$\nabla_{\theta_i}\nabla_{\theta_j} J_j(\theta_1, ..., \theta_i, ...\theta_n)\alpha_i\alpha_j \approx \mathbf{E}[\sum_{t=0}^{T} \gamma^t R_{j,t}(\sum_{l=0}^{t} \nabla_{\theta_i} log\pi_i(A_{i,l} \mid S_l)) (\sum_{l=0}^{t} \nabla_{\theta_j} log\pi_j(A_{j,l} \mid S_l))^T] \tag{4.5}$$

Dersom agenten ikke har tilgang til en motstanders parametere, er det mulig å estimere disse. De estimerte parameterne er de som med størst sannsynlighet vil generere handlingene som er observert hos motstanderen. For å redusere tiden det tar å trene en agent, gis agenten tilgang på motstandernes parametere i dette prosjektet.

4.2.2 IMPALA

For å trene en agent med IMPALA brukes en eller flere læringsprosesser. Dette er prosesser som har som oppgave å ta i mot data, oppdatere policy- og verdinettverket, og deretter sende de oppdaterte parameterne til en eller flere arbeidsprosesser. Arbeidsprosessen gjennomfører én episode, dvs. spiller én runde med Diplomacy, og sender data fra episoden til læringsprosessen. Etter at arbeidsprosessen har mottatt oppdaterte parameterverdier, gjentas denne prosedyren. Figur 4.2 illustrerer hvordan de ulike prosessene kommuniserer. De ulike agentene i de ulike prosessene har, i dette prosjektet, alltid de samme policyene. Dette gjøres ved at arbeidsprosessene venter med å generere data til de har fått oppdaterte parametere, og læringsprosessen venter med å lære til den har fått data fra alle arbeidsprosessene. På den måten blir algoritmen en "on-policy"-algoritme, og man slipper å bruke "importance sampling". De ulike arbeidsprosessene vil likevel utforske ulike deler av tilstandsrommet, da policyen er stokastisk.



Figur 4.2: Kommunikasjon mellom læringsprosesser og arbeidsprosesser i IMPALA. Arbeidsprosessene sender informasjon fra en episode til læringsprosessen. I retur mottar arbeidsprosessene oppdaterte parametere (θ).

4.3 Policy- og verdinettverk

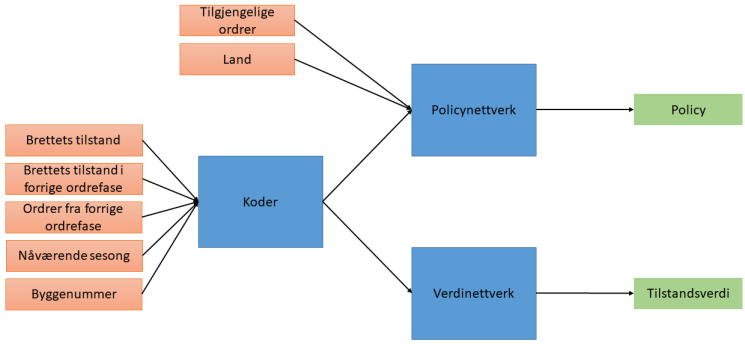
I A2C og LOLA trenger man et verdinettverk og et policynettverk. Nettverkene som er brukt i dette prosjektet er inspirert av nettverkene til Paquette et al. [2019], Anthony et al. [2020], Gray et al. [2021] og Bakhtin et al. [2021].

Figur 4.3 viser hovedbestanddelene i nettverkene. Begge nettverkene bruker den samme koderen til å kode informasjon om nåværende fase og informasjon fra forrige ordrefase. Utdata fra koderen gis deretter til policynettverket og verdinettverket, som beregner henholdsvis policy og tilstandsverdi. Til å beregne policy, bruker policynettverket også informasjon om hvilke ordrer som er tilgjengelig.

Videre følger en mer detaljert beskrivelse av de ulike delene av nettverkene.

4.3.1 Inndata

Å ha brettets tilstand som input til policy- og verdinettverket er en svært vanlig tilnærming når en agent skal lære å spille et brettspill. Ofte er denne informasjonen nok til å avgjøre hvilke handlinger som skal tas. Siden evne til samarbeid kan være avgjørende for å gjøre det godt i Diplomacy, er det viktig å også ta hensyn til data som kan avsløre hvilke spillere som har inngått allianser med hverandre. Hvilke ordrer som ble gitt i forrige ordrefase, vil kunne si noe om allianser blant spillerne. For eksempel vil en SUPPORT-ordre gitt fra et land til et annet indikere samarbeid mellom landene. Det kan tenkes at jo flere foregående ordrefaser man betrakter, desto lettere blir det å oppdage samarbeid. Land som støtter hverandre over flere runder, har større sannsynlighet for å ha inngått samarbeid enn land som støtter hverandre i bare én runde. Samtidig er samarbeid dynamisk; To land som samarbeider i den ene runden, kan være fiender i den neste. Derfor har det i dette prosjektet blitt valgt å kun ta med informasjon om den siste ordrefasen. Dette er likt med inndataen i nettverket til Paquette et al. [2019], Anthony et al. [2020], Gray et al. [2021] og Bakhtin et al. [2021].



Figur 4.3: Oversikt over verdi- og policynettverk. Oransje bokser representerer inndata. Blå bokser representerer nettverkets bestanddeler. Grønne bokser representerer utdata.

Inndataen til koderen til Paquette et al. skiller seg fra inndataen til koderen til Anthony et al., Gray et al. og Bakhtin et al. Paquette et al. valgte å bruke forhåndsbestemte features for å representere ordrene gitt i forrige ordrefase. De andre forfatterne har benyttet seg av lærte "embeddings" (avsnitt 2.8) av ordrene. Anthony et al. har følgende inndata: den nåværende tilstanden til brettet, tilstanden til brettet i forrige ordrefase, ordre fra forrige ordrefase, nåværende sesong, landet som nettverket skal beregne policy for, og forskjellen mellom antall forsyningssentre og antall enheter hvert enkelt land har (byggnummer). Koderen brukt i dette prosjektet har samme inndata som Anthony et al., bortsett fra at den ikke har med hvilke land som det skal beregnes policy for. I følge artikkelen til Anthony et al. presterer agenten bedre om denne inndataen brukes, sammenliknet med Paquette et al. sine forhåndsbestemte features.

Spillbrettets tilstand er representert av en matrise hvor hver rad representerer en provins eller en kyst (se figur 4.4). For provinser som har to kyster vil det ha betydning hvilken kyst en flåte befinner seg ved. Dette gjelder provinsene "SPA", "STP" og "BUL". Det er lagt til en rad for hver av kystene til disse provinsene, slik at informasjon som er spesifikk for disse kystene også kan kodes. Informasjon om hver provins og kyst "one-hot"-kodes. For provinsene som kun har én kyst, representere den samme raden både kysten og provinsen. "One-hot"-kodingen inneholder informasjon om hvilken enhetstype som befinner seg i provinsen, hvilket land som eier enheten, hvorvidt det kan bygges i provinsen, hvorvidt enheten som befinner seg i provinsen kan fjernes, hvilken enhetstype

som eventuelt har blitt fjernet fra provinsen, hvilket land som eventuelt har fått fjernet sin enhet fra provinsen, hvilken type provinsen er (land, vann, kyst) og, dersom provinsen er et forsyningssenter, hvem som eier forsyningssenteret. Denne måten å kode brettet på er likt med arbeidet til de fire ovennevnte forfatterne.

Ordrer som er tilgjengelig for en spiller blir "one-hot"-kodet likt med arbeidet til Gray et al. [2021][2]. Ordrene kodes med informasjon om ordretype, underliggende ordretype, underliggende enhetstype, underliggende startprovins og underliggende destinasjonsprovins. Dersom ordren som skal kodes er en "SUPPORT"- eller en "CONVOY"-ordre, er ordren som kommer etter bokstavene "S" eller "C" en underliggende ordre. I ordren "A LON S F EDI - YOR" er "F EDI - YOR" en underliggende ordre. Dermed er ordretypen "SUPPORT", underliggende ordretype er "MOVE", underliggende startprovins er "EDI", underliggende destinasjonsprovins er "YOR" og underliggende enhetstype er "FLEET". Dersom ordren ikke har en underliggende ordre, som f.eks. "A LON - YOR", kodes ordren som om den skulle vært sin egen underliggende ordre. Det vil si at ordretype er "MOVE", underliggende ordretype er "MOVE", underliggende startprovins er er "LON", underliggende destinasjonsprovins er "YOR" og underliggende enhetstype er "ARMY". Dersom ordren ikke har en underliggende destinasjonsprovins, slik som "A LON H", kodes provinsen ("LON") både som start- og destinasjonsprovins. De kodede ordrene er en del av inndataen til policynettverket, beskrevet i avsnitt 4.3.3. I tillegg er informasjon om hvilket land det skal bestemmes ordrer for en del av inndataen til policynettverket.

4.3.2 Koder

Koderen består av tre GNN (se figur 4.4), og to "embedding"-lag. "Embedding"-lagene holder på vektorrepresentasjoner for henholdsvis sesonger og ordrer. For spillet "France vs. Austria" består GNN-ene av åtte lag med grafkonvolusjoner. Det vil si at hvert GNN gjennomfører åtte oppdateringer som beskrevet i likning 2.8. Dette er likt med koderen brukt av Paquette et al. [2019], Gray et al. [2021] og Bakhtin et al. [2021]. Årsaken til at åtte lag er brukt, er at ingen provinser er mer enn 8 steg unna hverandre i grafrepresentasjon av spillbrettet. Dette gjør at GNN-ene rekker å propagere informasjon mellom alle noder i grafen. For spillet "Pure" brukes bare ett lag med grafkonvolusjoner pr. GNN. Dette skyldes at i "Pure" er alle provinser naboer med hverandre.

[2]I artikkelen til Gray et al. [2021] kommer det ikke klart frem hvordan ordrer skal kodes. Gitt beskrivelsen til Gray et al. skal f.eks. en "MOVE"-ordre og en "HOLD"-ordre kodes likt. Informasjon om hvor enheten skal flytte kodes blant annet ikke. Derfor vil det i mange tilfeller ikke være mulig å skille mellom slike ordrer. Riktig koding av ordrer er vist i kildekoden: `https://github.com/facebookresearch/diplomacy_searchbot/blob/main/fairdiplomacy/models/diplomacy_model/diplomacy_model.py`

La X være inndata til et lag i GNN-ene. Når "France vs. Austria" spilles, gjennomfører hvert lag i GNN-et følgende operasjoner:

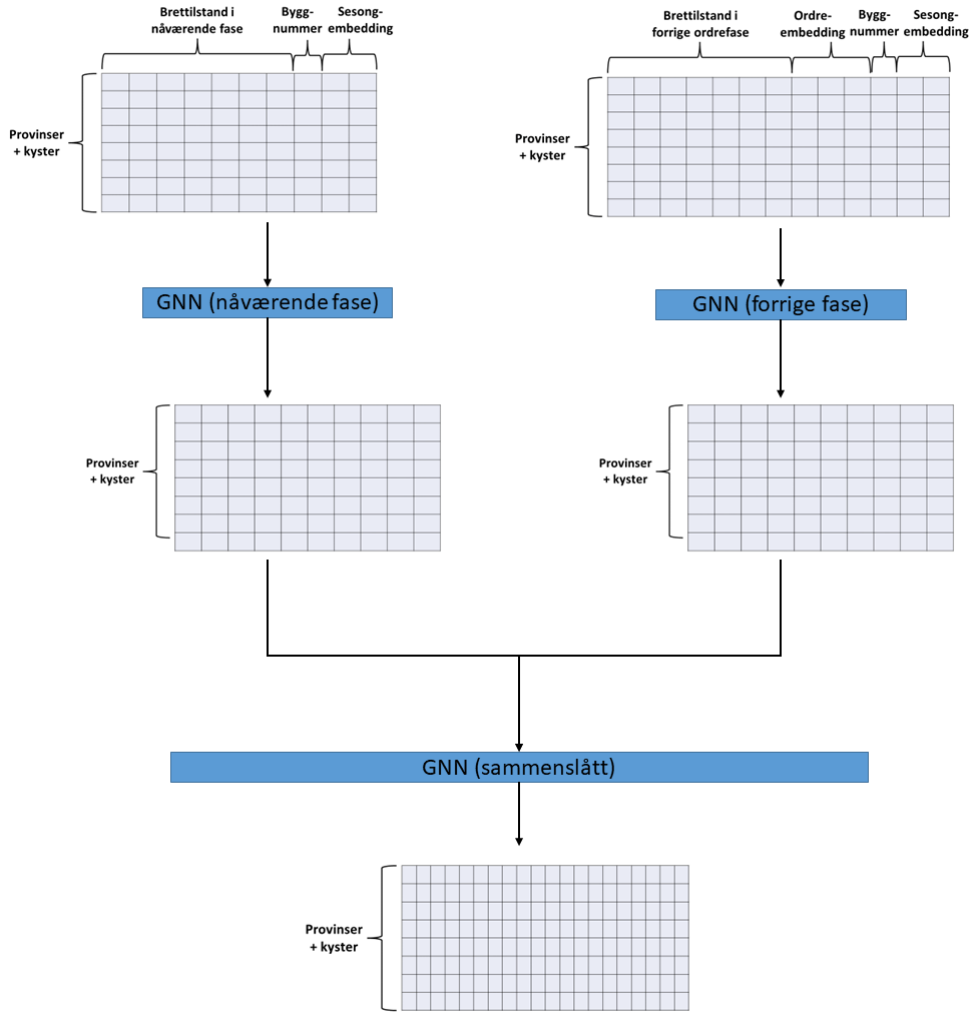
1. $Y_{GraphConv} = GraphConv(X)$
2. $Y_{Linear} = XA$
3. $Y_{Add} = Y_{GraphConv} + Y_{Linear}$
4. $Y_{BatchNormalization} = BatchNormalization(Y_{Add})$
5. $Y_{ReLU} = ReLU(Y_{BatchNormalization})$
6. $Y_{Dropout} = Dropout(Y_{ReLU})$
7. $Y_{Out} = Y_{Dropout} + X$

I steg 1 gjennomføres en grafkonvolusjon. I steg 2 multipliseres X med en matrise A. A består av vekter som kan trenes. I steg 3 adderes resultatet fra grafkonvolusjonen med resultatet fra lineærtransformasjonen i steg 2. I steg 4 utføres batchnormalisering (se avsnitt 2.9). Resultatet fra batchnormaliseringen brukes som inndata til aktiveringsfunksjonen ReLU i steg 5. Deretter anvendes et "dropout"-lag (se avsnitt 2.10) i steg 6. Utdata fra GNN-laget er utdata fra "dropout"-laget addert med X. Det siste steget kalles en "skip-connection", ettersom man lar X hoppe over alle stegene i GNN-laget før det legges til lagets utdata (se avsnitt 2.11). Om GNN-et består av flere lag, blir Y_{Out} inndata for det neste laget. Dette designet av et GNN-lag er likt med arbeidet til Gray et al. [2021] og Bakhtin et al. [2021].

Ettersom batchnormalisering, dropout og "skip-connection" er teknikker som først og fremst er nyttig i nettverk med mange lag, droppes disse når agenten skal lære å spille "Pure".

Et av GNN-ene koder informasjon om brettets nåværende tilstand, byggenummer, og en lært "embedding" av nåværende sesong. "Embedding"-en læres slik som beskrevet i avsnitt 2.8. Denne informasjonen slås først sammen, før den gis til GNN-et. Det andre GNN-et koder informasjon om brettets tilstand i forrige ordrefase, en lært "embedding" av ordrene gitt i forrige ordrefase, byggenummer og en lært "embedding" av nåværende sesong. Utdata fra disse GNN-ene slås sammen og brukes som inndata til det tredje GNN-et. Utdata fra det tredje GNN-et brukes deretter som inndata til policynettverket og verdinettverket.

Eksempel på utregning: For å gjøre det enklere å forstå metoden som er brukt, vises her et eksempel på hvordan inndata omgjøres til utdata i koderen. I eksempelet består spillbrettet av tre provinser: HOL, KIE og BER. To nasjoner er med i spillet: Tyskland og Frankrike. Tyskland har en flåte i KIE, mens Frankrike



Figur 4.4: Illustrasjon av koder som består av tre GNN og to "embedding"-lag. Først kodes informasjon om nåværende fase og informasjon fra forrige ordrefase med separate GNN. Deretter slås utdata fra disse GNN-ene sammen, og gis til et tredje GNN. Utdata fra det tredje GNN-et blir senere brukt som inndata til policy- og verdinettverket. Embedding-lagene er utelatt, men vektorrepresentasjonene av sesonger og ordrer er lagt til inndataen til GNN for nåværende fase og GNN for forrige ordrefase. Figuren er basert på tilsvarende illustrasjon, samt beskrivelser, gitt av Gray et al. [2021] og Anthony et al. [2020].

Figur 4.5: Spillbrett brukt i eksempel på utregning. Kvadratisk flagg representerer en armé, mens avlangt flagg representerer en flåte.

har en armé i HOL. Denne situasjonen er vist i figur 4.5. I dette tilfellet vil kun informasjon om hvilken type enhet og hvem som eier enheten bli kodet. Fremgangsmåten er lik for de tre GNN-ene. Eksempelet viser derfor kun en beregning for GNN-et som koder informasjon om nåværende tilstand, byggenummer og en lært "embedding" av nåværende sesong. Nåværende sesong i dette eksempelet er "Fall".

Inndata til koderen er matrisen $\mathbf{S}$, som er en "one-hot"-koding av brettets tilstand, vektoren $\mathbf{b}$ som viser byggnummer, og nåværende sesong. I dette tilfellet er alle byggnumrene lik 0, og $\mathbf{b} = \overrightarrow{0}$. $\mathbf{S}$ er vist i likning 4.6. I $\mathbf{S}$ er øverste rad "one-hot"-kodingen for HOL, den midterste raden er "one-hot"-kodingen for KIE, og den nederste raden er "one-hot"-kodingen for BER. Kolonnene representerer, fra venstre til høyre: armé, flåte, ingen enhetstype, Frankrike, Tyskland, ingen nasjon.

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

Det første som gjøres når inndata er gitt til koderen, er å konvertere sesongen til en "embedding". Anta at "embedding" for sesonger er gitt av likning 4.7. Den øverste raden er for "Spring", mens den nederste raden er for "Fall". Ved å "one-hot"-kode "Fall", og multiplisere "one-hot"-kodingen med $\mathbf{E}$, får man "embedding"-en for "Fall". Dette er vist i likning 4.8.

$$\mathbf{E} = \begin{bmatrix} 0.2 & 0.3 \\ 0.1 & 0.5 \end{bmatrix} \tag{4.7}$$

$$\mathbf{e}_{Fall} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0.2 & 0.3 \\ 0.1 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.5 \end{bmatrix} \tag{4.8}$$

Etter at "embedding"-en for sesongen er funnet, slås $\mathbf{S}$, $\mathbf{b}$ og $\mathbf{e}_{Fall}$ sammen og brukes som inndata $\mathbf{X}$ til *GraphConv* i steg 1. $\mathbf{X}$ er vist i likning 4.9.

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0.1 & 0.5 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0.1 & 0.5 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0.1 & 0.5 \end{bmatrix} \tag{4.9}$$

Videre vises hvordan *GraphConv* beregnes etter likning 2.8. I *GraphConv*-beregningen er spillbrettet omgjort til en graf, og hver provins er representert av en node. *GraphConv*-beregningen starter med at hver node, u, i grafen sender sin vektor $\mathbf{x}_u$ til sine nabonoder. Når $\mathbf{x}_{KIE}$ skal oppdateres, er det første som gjøres å normalisere $\mathbf{x}_{KIE}$ og vektorene som skal sendes til KIE: $\mathbf{x}_{HOL}$ og $\mathbf{x}_{BER}$. Normaliseringen av $\mathbf{x}_{HOL}$ gjøres ved å dele $\mathbf{x}_{HOL}$ på kvadratroten av antall naboprovinser til KIE, multiplisert med antall naboprovinser til HOL. Dette blir $\frac{\mathbf{x}_{HOL}}{\sqrt{|N(KIE)||N(HOL)|}} = \frac{\mathbf{x}_{HOL}}{\sqrt{2*1}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0.1 & 0.5 \end{bmatrix}$. Tilsvarende er normaliseringen av $\mathbf{x}_{BER}$ lik $\frac{\mathbf{x}_{BER}}{\sqrt{|N(KIE)||N(BER)|}}$, og normaliseringen av $\mathbf{x}_{KIE}$ lik $\frac{\mathbf{x}_{KIE}}{\sqrt{|N(KIE)||N(KIE)|}}$.

De normaliserte radvektorene til HOL og BER summeres med den normaliserte radvektoren til KIE. Deretter multipliseres den summerte vektoren med en vektmatrise $\mathbf{W}$. Resultatet fra multiplikasjonen gis så til en aktiveringsfunksjon. For dette prosjektet er det brukt en lineær aktiveringsfunksjon som setter utdata lik inndata. Utdata fra aktiveringsfunksjonen blir KIE sin nye vektor $\mathbf{y}_{GraphConv,KIE}$. Utregningen er vist i likning 4.10[3]. Den samme prosedyren gjennomføres også for HOL og BER. Utdata fra *GraphConv* er en matrise som består av de nye vektorene for provinsene, vist i likning 4.11.

$$\begin{aligned} \mathbf{y}_{GraphConv,KIE} = \Bigg(& \frac{\mathbf{x}_{KIE}}{\sqrt{|\ N(KIE)\ ||\ N(KIE)\ |}} + \frac{\mathbf{x}_{HOL}}{\sqrt{|\ N(KIE)\ ||\ N(HOL)\ |}} + \\ & \frac{\mathbf{x}_{BER}}{\sqrt{|\ N(KIE)\ ||\ N(BER)\ |}} \Bigg) * \mathbf{W} = \\ \Bigg(& \frac{1}{2} \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0.1 & 0.5 \end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0.1 & 0.5 \end{bmatrix} + \\ & \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0.1 & 0.5 \end{bmatrix} \Bigg) * \mathbf{W} = \\ & \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{2} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{2} & \frac{1}{\sqrt{2}} & 0 & \frac{0.2\sqrt{2}+0.1}{2} & \frac{\sqrt{2}+0.5}{2} \end{bmatrix} * \mathbf{W} \end{aligned} \tag{4.10}$$

[3] I likning 4.10 har vektoren og matrisen byttet plass, sammenliknet med likning 2.8. I likning 4.10 er vektoren en radvektor, mens vektorene i likning 2.8 er kolonnevektorer.

$$\mathbf{Y}_{GraphConv} = \begin{bmatrix} \mathbf{y}_{GraphConv,HOL} \\ \mathbf{y}_{GraphConv,KIE} \\ \mathbf{y}_{GraphConv,BER} \end{bmatrix} \tag{4.11}$$

Etter at $\mathbf{Y}_{GraphConv}$ er beregnet, utføres trinn 2 - 7, som beskrevet ovenfor. Om GNN-et består av flere lag, brukes utdata fra et lag, $\mathbf{Y}_{Out}$, som inndata til det etterfølgende laget, og trinn 1 - 7 gjennomføres på nytt. Etter at $\mathbf{Y}_{Out}$ er beregnet for det siste laget, slås $\mathbf{Y}_{Out}$ sammen med tilsvarende utdata-matrise for GNN-et som koder informasjon fra forrige ordrefase. Den sammenslåtte matrisen gis deretter som inndata til det siste GNN-et, som vist i figur 4.4.

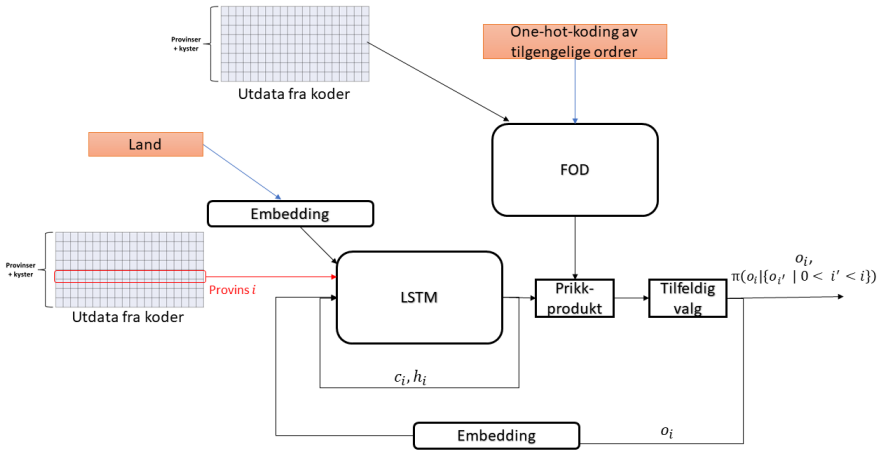
4.3.3 Policynettverk

Policynettverkene til Paquette et al. [2019], Anthony et al. [2020] og Gray et al. [2021] har til felles at de predikerer ordrer sekvensielt, en provins ad gangen. En topologisk sortering bestemmer rekkefølgen av provinsene. Det er kun provinsene som tilhører et bestemt land som inngår i beregningen av policy.

Policynettverket til Paquette et al. bruker et LSTM-nettverk for å bestemme ordrer. LSTM-nettverket til Paquette et al. gir for hver provins en sannsynlighetsfordeling over ordrer som er tilgjengelige for agenten. For å oppnå det samme bruker Anthony et al. en "relational order decoder" som består av GNN-er. I henhold til Gray et al. er metoden til Anthony et al. ganske krevende med tanke på beregningskostnader, sett opp mot hvor mye metoden forbedrer agentens prestasjoner. Gray et al. bruker selv en "featurized order decoder" (FOD) i kombinasjon med en LSTM. Gray et al. sin metode er den som har gitt best resultat, og er også brukt av Bakhtin et al. [2021]. Av den grunn er en FOD i kombinasjon med LSTM blitt brukt i dette prosjektet.

Policynettverket for dette prosjektet er illustrert i figur 4.6. I hver iterasjon i policynettverket tar LSTM-cellen inn informasjon om provinsen det skal velges en ordre for, en lært "embedding" av landet som skal utføre ordren og en lært "embedding" av ordren som ble valgt i forrige iterasjon. I tillegg har man en skjult tilstand som overføres fra en iterasjon til annen, som beskrevet i avsnitt 2.7. Informasjonen om en provins er i policynettverket radvektoren tilhørende provinsen i koderens utdata.

FOD lærer en vektorrepresentasjon av de tilgjengelige ordrene for en gitt provins. Disse vektorene multipliseres deretter med utdata fra LSTM tilhørende provinsen. Resultatet av multiplikasjonen gir en "logit"-verdi for hver tilgjengelige ordre i provinsen. Ved å bruke "logit"-verdiene som inndata til aktiveringsfunksjonen "softmax" får man en sannsynlighetsfordeling over de tilgjengelige ordrene for provinsen.



Figur 4.6: Illustrasjon av policynettverk med LSTM og "featurized order decoder" (FOD)

FOD består av tre lineærtransformasjoner: En for å transformere informasjon om en tilgjengelig ordres underliggende startprovins, en for å transformere informasjon om den tilgjengelige ordrens underliggende destinasjonsprovins og en for å transformere "one-hot"-kodingen av den tilgjengelige ordren (se avsnitt 4.3.1). Informasjon om den underliggende startprovinsen og den underliggende destinasjonsprovinsen er radvektorene tilhørende start- og destinasjonsprovinsen i koderens utdata. Resultatet av lineærtransformasjonene summeres sammen. Denne summen blir FOD-ens utdata.

Policynettverkets utdata er valgt handling, $a = (o_1, ..., o_I)$, og en sannsynlighet for for hver ordre, o_i, den valgte handlingen består av. Ettersom policynettverket predikerer ordrer for provinser sekvensielt, kan sannsynligheten for ordre o_i sees på som en betinget sannsynlighet. Betingelsen er ordrene som er predikert før o_i ($\{o_{i'} \mid 0 < i^{'} < i\}$). Sannsynligheten for handling a blir derfor $\pi(a) = \prod_{i=1}^{I} \pi(o_i \mid \{o_{i'} \mid 0 < i^{'} < i\})$ [Gray et al., 2021].

Policy-gradienten blir beregnet etter at en handling er valgt og man har sett resultatet av å gjennomføre denne handlingen. Til å beregne $\pi(a)$, har policynettverket derfor en funksjon for å bruke allerede valgte ordrer, i stedet for å bestemme nye ordrer, når policy-gradienten skal beregnes.

Et LSTM-nettverk fungerer spesielt godt når det skal behandle lange datasekvenser. I "France vs. Austria" vil LSTM-nettverket måtte predikere ordrer for opptil 18 provinser. Når "Pure" skal spilles, predikerer policynettverket ikke

ordre for mer enn 3 provinser. Derfor byttes LSTM-nettverket ut med et enklere RNN når "Pure"-utgaven blir spilt. Målt i antall parametere blir policynettverket mye mindre med det enklere RNN-et. Dette vil trolig gjøre det raskere å trene nettverket. Færre parametere gjør det også mindre ressurskrevende å regne ut den deriverte av andre orden i LOLA-algoritmen.

Eksempel på utregning:

I dette eksempelet brukes situasjonen vist i figur 4.5. For å gjøre det enklere å forstå hvordan policynettverket virker, blir det her gitt et eksempel på hvordan ordrer bestemmes for provinsen HOL. Anta at Frankrike, som har en armé i HOL, har to tilgjengelige ordrer: Arméen kan enten bli værende i HOL ("A HOL H"), eller angripe flåten som befinner seg i KIE ("A HOL - KIE"). For enkelthetsskyld kodes kun informasjon om ordretype, startprovins og destinasjonsprovins. "One-hot"-kodingen for "A HOL H" er vist i likning 4.12. I vektoren representerer elementene, fra venstre til høyre: ordretypen "HOLD", ordretypen "MOVE", ordretypen "SUPPORT", startprovinsen HOL, startprovinsen KIE, startprovinsen BER, destinasjonsprovinsen HOL, destinasjonsprovinsen KIE og destinasjonsprovinsen BER. "One hot"-kodingen for "A HOL - KIE" er vist i likning 4.13.

$$\mathbf{o}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.12}$$

$$\mathbf{o}_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.13}$$

Det første som gjøres når en ordre skal bestemmes i provins u, er å gi LSTM-cellen koderens utdata for provins u som input. I tillegg er input en "embedding" som representerer landet som velger ordre, og en "embedding" som representerer ordren valgt i den forrige iterasjonen i policynettverket. Koderen har utdata som vist i likning 4.14. "Embedding" for land er vist i likning 4.15. I likning 4.16 vises "embedding" for de n ordrene som kan være tilgjengelig i løpet av hele spillet, i tillegg til en "default embedding" som blir brukt i den første iterasjonen i policynettverket.

Når det skal velges ordrer for provinsen HOL, gis LSTM-cellen $\mathbf{y}_{EncoderOut,HOL}$, $\mathbf{e}_{France}$ og $\mathbf{e}_{default}$ som input. Utdata fra LSTM blir dermed:
$\mathbf{y}_{LSTM} = LSTM(\mathbf{y}_{EncoderOut,HOL}, \mathbf{e}_{France}, \mathbf{e}_{default})$.

$$\mathbf{Y}_{EncoderOut} = \begin{bmatrix} \mathbf{y}_{EncoderOut,HOL} \\ \mathbf{y}_{EncoderOut,KIE} \\ \mathbf{y}_{EncoderOut,BER} \end{bmatrix} \tag{4.14}$$

$$\mathbf{E}_{Power} = \begin{bmatrix} \mathbf{e}_{France} \\ \mathbf{e}_{Germany} \end{bmatrix} \tag{4.15}$$

$$\mathbf{E}_{Order} = \begin{bmatrix} \mathbf{e}_{default} \\ \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \mathbf{e}_n \end{bmatrix} \tag{4.16}$$

Når FOD-en skal beregne vektorrepresentasjoner for ordrer, brukes utdata fra koderen i kombinasjon med "one-hot"-kodingen for ordrene. FOD-en sin utdata for ordren "A HOL - KIE" vil bli beregnet her. Det første som gjøres er å lineærtransformere radvektoren fra koderens utdata, som tilhører ordrens startprovins ($\mathbf{y}_{EncoderOut,HOL}$). Dette er vist i likning 4.17, hvor $\mathbf{A}_{src}$ er en matrise med trenbare vekter. Det samme gjøres for koderens utdata tilhørende destinasjonsprovinsen ($\mathbf{y}_{EncoderOut,HOL}$) og "one-hot"-kodingen for ordren. Dette er vist i henholdsvis likning 4.18 og likning 4.19.

$$\mathbf{y}_{LinearSrc} = \mathbf{A}_{src}\mathbf{y}^T_{EncoderOut,HOL} \tag{4.17}$$

$$\mathbf{y}_{LinearDst} = \mathbf{A}_{dst}\mathbf{y}^T_{EncoderOut,KIE} \tag{4.18}$$

$$\mathbf{y}_{LinearOneHot} = \mathbf{A}_{OneHot}\mathbf{o}^T_2 \tag{4.19}$$

FOD-ens utdata er summen av $\mathbf{y}_{LinearSrc}$, $\mathbf{y}_{LinearDst}$ og $\mathbf{y}_{LinearOneHot}$ ($\mathbf{y}_{FOD} = \mathbf{y}_{LinearSrc} + \mathbf{y}_{LinearDst} + \mathbf{y}_{LinearOneHot}$).

Logit-verdien for ordren "A HOL - KIE" er prikkproduktet mellom utdata fra LSTM og utdata fra FOD-en ($\mathbf{y}_{LSTM} \cdot \mathbf{y}_{FOD}$). Når det er flere tilgengelige ordrer i HOL, beregnes logit-verdien for hver tilgjengelige ordre. Her gjelder dette ordrene "A HOL H" og "A HOL - KIE". Aktiveringsfunksjonen "softmax" brukes deretter til å omgjøre logit-verdiene til en sannsynlighetsfordeling over ordrene. Etter denne sannsynlighetsfordelingen trekkes hvilken ordre som skal utføres i provinsen HOL.

4.3.4 Verdinettverk

Verdinettverket består av et feedforward-nettverk (se avsnitt 2.3) bestående av ett skjult lag med nevroner. Inndata til verdinettverket er utdata fra koderen. Verdinettverket predikerer en tilstandsverdi for alle nasjonene som er med i spillet. Ettersom belønningen agentene mottar summerer til 1, og er aldri større enn 1 eller mindre enn 0, er softmax godt egnet som aktiveringsfunksjon. Dette er likt med Gray et al. [2021] og Bakhtin et al. [2021]. Anthony et al. [2020] valgte å bruke ReLU som aktiveringsfunksjon, noe som ikke fungerer like godt med den valgte måten å gi belønninger.

4.4 Måling av Resultat

I dette prosjektet er det blitt brukt "1 vs 6"-spill for å måle hvor godt agentene spiller. I "1 vs 6"-spill spiller en type agent mot seks kopier av en annen type agent. En slik måte å teste agenter på i Diplomacy er brukt av Paquette et al. [2019], Anthony et al. [2020], Gray et al. [2021] og Bakhtin et al. [2021]. En av fordelene med en slik testmetode er at man kan se hvorvidt den unike agenten kan utnytte svakheter ved spillet til agenten som er kopiert. Dette innebærer også at man kan måle hvor robust strategien til agenten som er kopiert er mot den unike agentens strategi.

To modeller vil bli testet mot hverandre. Den ene er trent opp med LOLA-algoritmen, den andre er trent opp med A2C-algoritmen. Disse modellene vil bli testet i spill mot hverandre, og mot en agent som utfører ordrer etter en uniform sannsynlighetsfordeling.

Ettersom det er viktig å kunne samarbeide med andre spillere i Diplomacy, gjennomføres tester på hvor god agenten har blitt til dette. Samarbeid mellom agenter måles med en metode kalt "koalisjonsanalyse". Denne metoden ble beskrevet av Paquette et al. [2019], og måler hvor ofte og hvor effektivt en agent gir støtte til andre agenter. Hvor ofte en agent gir støtte til en annen agent måles ved å dele antall "SUPPORT"-ordrer som blir gitt til en annen agent på antall "SUPPORT"-ordrer agenten har gitt. Dette forholdstallet kalles en "cross power-support ratio".

At en "SUPPORT"-ordre er effektiv vil si at den har en positiv effekt for agenten som mottar støtte. Om en agent forsøker å flytte til en ny provins ("A BER - KIE"), og agenten trenger støtte for å kunne flytte ("A RUH S A BER - KIE"), vil "SUPPORT"-ordren være effektiv om den fører til at agenten får flyttet til den ønskede provinsen. Hvor effektiv en agent er til å gi "SUPPORT"-ordrer måles ved å dele antall effektive "SUPPORT"-ordrer gitt til en annen agent på antall "SUPPORT"-ordre som har blitt gitt til en annen agent. Dette forholdstallet kalles "effective cross-power support ration".

En "SUPPORT"-ordre gitt til en annen agent regnes kun som effektiv dersom fraværet av "SUPPORT"-ordren ville gitt et annet resultat for motstanderen som mottar støtten. Det vil si at om flere agenter gir "SUPPORT"-ordrer til den samme motstanderen, og motstanderen kun trenger støtte fra én agent, vil ingen av "SUPPORT"-ordrene regnes som effektive. De lave tallene skyldes i stor grad at når en kopi av en agent gir støtte til en annen kopi, gir flere av de andre kopiene støtte til den samme kopien.

Dersom flere agenter gir støtte til samme angrep, og noen av støttene er overflødige, kan man heller ikke se bort i fra de overflødige støttene når "effective cross-power support ratio" skal regnes ut. Dette skyldes at det ikke er mulig å peke ut hvilke "SUPPORT"-ordrer som er overflødige, og hvilke som ikke er det. Det

er kun mulig å regne ut hvor mange av "SUPPORT"-ordrene som er overflødig. Det kan derfor være nødvendig å bruke en tredje form for koalisjonsanalyse, kalt "Coordinated cross-power support move ratio".

"Coordinated cross-power support move ratio" viser hvor stor andel av "SUPPORT"-ordrene som gis for å støtte en motstander i et angrep (f.eks. "F BER S A RUH - MUN"), gis når motspilleren faktisk angriper ("A RUH - MUN"). Problemet med "coordinated cross-power support move ratio", sammenliknet med "effective cross-power support ratio", er at det ikke tas hensyn til hvor mye støtte parten som blir angrepet får. Likevel vil "coordinated cross-power support move ratio" kunne gi en indikasjon på hvor god agentene er til å samarbeide, ved at den måler hvor gode de er til å koordinere angrep.

Paquette et al. [2019] viste at A2C-agenten de trente opp var dårligere til å samarbeide enn en "supervised learning"-agent. Ved å bruke koalisjonsanalysen kan man se om LOLA-algoritmen kan brukes til å gjøre en RL-agent flinkere til å samarbeide enn A2C-agenten.

4.5 Oppsummering

I dette kapittelet har metoden brukt til å oppnå resultatene i kapittel 5 blitt beskrevet. Kapittelet startet med en beskrivelse av omgivelsene de Diplomacy-spillende agentene befinner seg i. Omgivelsene bestemmes av blant annet hvilket spillbrettet det spilles på, makslengden på spillet og belønningssignalet. To ulike spillbrett har blitt brukt: originalbrettet i Diplomacy, samt en enklere utgave kalt "Pure". På originalbrettet har spillet "France vs Austria" blitt spilt. Som navnet tilsier spiller kun landene Frankrike og Østerrike i denne utgaven. I "Pure"-utgaven av Diplomacy spiller sju land mot hverandre, og alle landene starter spillet med å ha kontroll over én provins. Totalt består "Pure"-brettet av 7 provinser. Makslengden på spillet er satt til 40 ordrefaser for "France vs. Austria" og 10 ordrefaser for "Pure". En agent får en belønning på 1 dersom den vinner spillet, og en belønning på 0 dersom den taper spillet. Om spillet ender i uavgjort, beregnes belønningen etter hvor mange forsyningssentre agenten har ved spillets slutt.

To algoritmer har blitt brukt til å lære en agent å spille Diplomacy: A2C og LOLA. I LOLA forsøker agenten å maskimere sin egen belønning ved å påvirke læringen til sine motstandere. A2C deler enkelte trekk med LOLA, men forsøker ikke å påvirke motstanderne sin læring. Deler av IMPALA-algoritmen har også blitt brukt til å gjøre treningen av agentene mer stabil.

Et policynettverk og et verdinettverk har blitt brukt til å lære policy og tilstandsverdier. Begge nettverkene deler på en koder bestående av Graph Neural Networks. For å beregne tilstandsverdier, gis utdata fra koderen til et "feedforward"-nettverk som beregner en tilstandsverdi for alle landene som deltar i spillet. For å

beregne en policy, gis utdata fra koderen og en "one-hot"-koding av tilgengelige ordrer til policynettverket. Policynettverket kombinerer et LSTM-nettverk med en "featurized-order-decoder". Policynettverkets utdata er en handling, satt sammen av ordrer, pluss en betinget sannsynlighet for hver av ordrene som utgjør handlingen.

Kapittel 5

Resultat og Analyse

I dette kapittelet vil resultatet av å anvende metodene beskrevet i kapittel 4 bli presentert og analysert. Kapittelet begynner med en gjennomgang av resultat for "Pure"-utgaven av Diplomacy. Ettersom det ble vanskelig å anvende LOLA-algoritmen på "France vs Austria", blir utfordringer ved denne utgaven deretter analysert. Til slutt vil usikkerheten i resultatene bli drøftet.

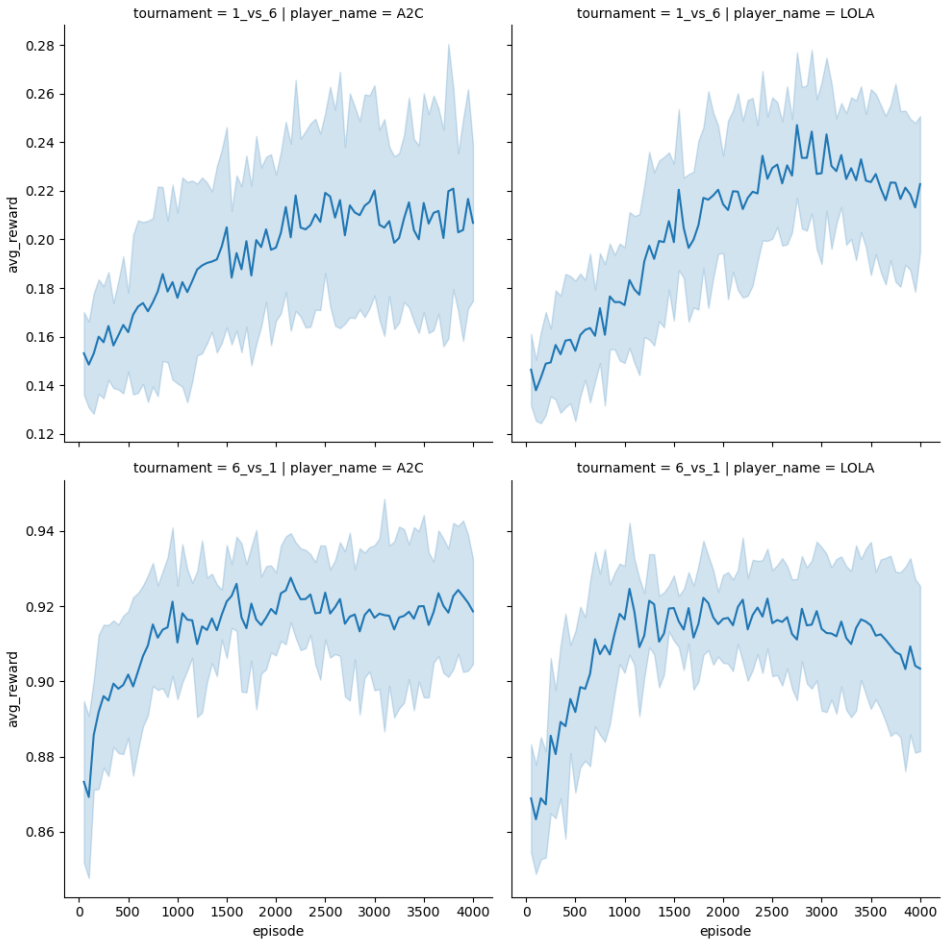
5.1 Pure-utgaven

En agent som spiller "Pure"-utgaven av Diplomacy er avhengig av støtte fra minst én av sine motstandere. Å få støtte fra en motstander er den eneste måten agenten kan avansere i starten av spillet. Dette påvirker hvordan man tolker resultatene fra "Pure"-utgaven av Diplomacy.

5.1.1 Treningsresultat

Treningen av A2C-agenten og LOLA-agenten har foregått over 4000 episoder. Ettersom det ikke finnes noen "benchmark"-agent for "Pure"-utgaven av Diplomacy, ble agenten som trekker handlinger tilfeldig brukt til å måle treningsprogresjon. For hver 50. episode ble det gjennomført "1 vs 6"-spill mellom agenten som ble trent og "Tilfeldig"-agenten. Det har både blitt gjennomført spill hvor agenten som blir trent spiller som en unik agent, og spill hvor agenten som blir trent spiller som seks kopier. Treningsresultatet for agentene er vist i figur 5.1.

Som en kan se av figur 5.1, forbedrer begge agentene sine resultater. En "Tilfeldig"-agent er forventet å score 0.143 når den spiller mot seks andre "Tilfeldig"-agenter. Tilsvarende er seks "Tilfeldig"-agenter forventet å score 0.857 samlet i



Figur 5.1: Treningsresultat. Data er hentet fra 10 kjøringer hvor LOLA-agenten og A2C-agenten for hver 50. episode spiller 100 spill mot en "Tilfeldig-agent" . Grafene til venstre tilhører A2C-agenten, grafene til høyre tilhører LOLA-agenten. De øverste grafene er fra turneringer hvor agentene har spilt mot seks kopier av "Tilfeldig"-agenten. De nederste grafene er fra turneringer hvor agenten har spilt som seks kopier mot én "Tilfeldig"-agent. Den mørkeblå linjen er gjennomsnittsbelønningen til A2C- eller LOLA-agenten. De lyseblå feltene representere standardavviket.

møte med én "Tilfeldig"-agent. I møte med seks "Tilfeldig"-agenter scorer A2C-agentene etter ca. 3000 episoder nesten 0.22 poeng i gjennomsnitt, mens LOLA-agenten scorer enda høyere på sitt beste. Både LOLA-agenten og A2C-agenten scorer rundt 0.92 på sitt beste i gjennomsnitt når de spiller som seks kopier mot en unik "Tilfeldig"-agent. Begge agentene blir bedre enn "Tilfeldig"-agenten etter trening.

Prestasjonen til LOLA-agenten ser ut til å avta etter ca. 3000 episoder, mens prestasjonen til A2C-agenten ser ut til å stabilisere seg. En annen forskjell mellom A2C- og LOLA-agenten er at variasjonen i prestasjonene til A2C-agenten er høyere enn variasjonen i prestasjonene til A2C-agenten. Dette kan sees av de lyseblå områdene i grafene. Disse representerer standardavviket.

5.1.2 Turnering

Til turneringen har det vært nødvendig å plukke ut én "utgave" av A2C- og LOLA-agentene. Treningsresultatet vist i figur 5.1 inneholder data fra flere kjøringer av treningsprosessen. For hver episode, og for hver kjøring av treningsprosessen, har det eksistert en unik "utgave" av agentene. Det er ikke mulig å bestemme hvilken av disse utgavene som best representerer evnen de forskjellige algoritmene har til å lære en agent å spille Diplomacy. Likevel kan det tenkes at to forskjellige metoder kan gi en god indikasjon på dette. Den ene metoden innebærer å velge den utgaven av agentene som har gitt det absolutt beste resultatet. Den andre metoden innebærer å velge ut den utgaven som presterer nærmest det beste utgavene har gjort i gjennomsnitt. Å velge utgaven som har gitt det absolutt beste resultatet, viser et minimum for det beste algoritmen kan få til. Ulempen med denne metoden er at flaks - f.eks. "riktig" frø til slumptallsgeneratorer - trolig har har hatt en betydning for resultatet. For denne turneringen har det derfor blitt valgt utgaver av agentene som presterer i nærheten av det beste utgavene har prestert i gjennomsnitt. Det vil si at A2C-agent bør score ca. 0.22 i møte med seks "Tilfeldig"-agenter, og ca. 0.92 i møte med én "Tilfeldig"-agent. LOLA-agent bør score ca. 0.25 i møte med seks "Tilfeldig"-agenter, og ca. 0.92 i møte med én "Tilfeldig"-agent. For å gjenspeile resultatene vist i tabell 5.1, bør LOLA-agenten score litt mindre enn A2C-agenten i møte med én "Tilfeldig"-agent.

For å sammenlikne prestasjonen til de ulike algoritmene, har det blitt gjennomført "1 vs 6"-spill. To tabeller oppsummerer hvordan de ulike agentene presterte mot hverandre. Tabell 5.1 viser gjennomsnittlig belønninger over 1000 spill for de ulike agentene. Den viser gjennomsnittlig belønning både for den unike agenten og agenten som er kopiert. Tabell 5.2 viser i prosent hvor mange seiere hver agent har hatt mot de andre agentene. I denne tabellen vises det også data for den unike agenten og den kopierte agenten.

LOLA-agenten er ikke forventet å fungere bedre enn A2C-agenten når den som

1x ↓ 6x →	Tilfeldig	A2C	LOLA
Tilfeldig	-	0.067 (0.081) / 0.933	0.086 (0.09) / 0.914
A2C	0.226 (0.186) / 0.774	-	0.203 (0.197) / 0.797
LOLA	0.248 (0.197) / 0.752	0.077 (0.089) / 0.923	-

Tabell 5.1: Tabell over gjennomsnittlig belønning for 1x agent / 6x agent over 1000 spill. Tallene skrevet i parentes er standardavvik.

1x ↓ 6x →	Tilfeldig	A2C	LOLA
Tilfeldig	0.0% / 0.0%	0.0 % / 22.2%	0.0% / 8.4%
A2C	0.7% / 0.0%	-	0.6% / 3.8%
LOLA	0.9% / 0.0%	0.0% / 20.3%	-

Tabell 5.2: Seiers-rate for 1x agent / 6x agent. Tallene er regnet ut fra resultatene fra 1000 spill.

eneste agent spiller mot seks tilfeldige agenter. Dette skyldes at LOLA forsøker å tilpasse sin policy ved å betrakte hvordan motstanderne lærer. "Tilfeldig"-agenten lærer ikke.

Som en kan se av tabell 5.1 er det 0.022 poeng som skiller A2C- og LOLA-agentene når de som eneste agent spiller mot seks "Tilfeldig"-agenter. A2C-agenten har i gjennomsnitt fått en belønning på 0.226, mens LOLA-agenten i gjennomsnitt har fått en belønning på 0.248. Spørsmålet blir så om en kan konkludere med at A2C- og LOLA-agentene er bedre enn en "Tilfeldig"-agent, i møte med seks "Tilfeldig"-agenter. I tillegg må det svares på om LOLA-agenten er bedre enn A2C-agenten, eller om forskjellen i resultatet kan tilskrives tilfeldigheter. For å svare på disse spørsmålene gjennomføres hypotesetesting. Signifikansnivået settes til $\alpha = 0.05$[1].

Hypotese 1: I møte med seks "Tilfeldig"-agenter scorer en A2C-agent i gjennomsnitt høyere enn en "Tilfeldig"-agent.

I møte med seks andre "Tilfeldig"-agenter har en "Tilfeldig"-agent en forventet gjennomsnittsbelønning på 0.143. Den spesifikke nullhypotesen blir dermed $H_0 : \mu = 0.143$. La den stokastiske variabelen X være belønningen en A2C-agent mottar i et spill mot seks andre "Tilfeldig"-agenter. Dette gir den alternative hypotesen $H_1 : \mu_X > 0.143$. Det er gjort 1000 observasjoner. Den observerte gjennomsnittsverdien til X er $\bar{x} = 0.226$, og det observerte standardavviket er $s = 0.186$. Dette gir en observert t-verdi $t_{obs} = \frac{0.226-0.143}{0.186/\sqrt{1000}} \approx 14.11$. Med dette har man en p-verdi $P(T > 14.11) \approx 0$. Dermed forkastes nullhypotesen, og det

[1]I forbindelse med hypotesetestingen brukes samme notasjon som Frisvold og Moe [2004].

konkluderes med at hypotese 1 trolig stemmer.

Hypotese 2: I møte med seks "Tilfeldig"-agenter scorer en LOLA-agent i gjennomsnitt høyere enn en "Tilfeldig"-agent.

La Y være belønningen en LOLA agent mottar i et spill mot seks "Tilfeldig"-agenter. Den spesifikke nullhypotesen er den samme som for Hypotese 1. Den alternative hypotesen er $H_1 : \mu_Y > 0.143$. Observert gjennomsnittsverdi for Y er $\bar{y} = 0.248$, og observert standardavvik er $s = 0.197$. Dette gir en observer t-verdi $t_{obs} = \frac{0.248-0.143}{0.197/\sqrt{1000}} \approx 16.85$. P-verdien blir dermed $P(T > 16.85) \approx 0$. Dermed forkastes nullhypotesen. Det konkluderes med at hypotese 2 trolig stemmer.

Hypotese 3: I møte med seks "Tilfeldig"-agenter scorer en LOLA-agent i gjennomsnitt høyere enn en A2C-agent.

For Hypotese 3 har vi alternativ hypotese $H_1 : \mu_Y - \mu_X > 0$, og spesifikk nullhypotese $H_0 : \mu_Y - \mu_X = 0$. Den observerte t-verdien er $t_{obs} = \frac{(0.248-0.226)-0}{\sqrt{0.197^2/1000+0.186^2/1000}} \approx 2.56$. Dette gir en p-verdi på $P(T > 2.56) \approx 0.006$. Nullhypotesen kan dermed forkastes, og det er grunn for å tro at hypotese 3 er korrekt.

Resultatet fra hypotesetestene tilsier at LOLA-agenten er bedre enn A2C-agenten i møte med seks "Tilfeldig"-agenter. I tillegg tilsier resultatene at A2C-agenten er bedre enn en "Tilfeldig"-agent i møte med seks "Tilfeldig"-agenter. Dette samsvarer med resultatene vist i figur 5.1. Derimot samsvarer ikke resultatet fra hypotese 3 med antakelsen om at en LOLA-agent ikke skal ha noen fordel i møte med en "Tilfeldig"-agent. Dette kan bety at forskjellen mellom agentene faktisk kan tilskrives tilfeldigheter.

Selv om hypotese 1-3 ser ut til å stemme, sier resultatet fra hypotesetestene ikke noe om hvor mye bedre agentene er enn "Tilfeldig"-agenten. Begge agentene scoret noe mer enn en "Tilfeldig"-agent scorer i gjennomsnitt. Som en kan se av tabell 5.2 klarer A2C- og LOLA-agentene å vinne henholdsvis 0.7% og 0.9% av spillene de spiller mot seks "Tilfeldig"-agenter. Det er derfor grunn til å tro at A2C- og LOLA-agentene ikke er så veldig mye bedre enn en "Tilfeldig"-agent i møte med seks andre "Tilfeldig"-agenter.

Det kan tenkes at forklaringen på at A2C-agenten og LOLA-agenten ikke scorer mye høyere, er at det er vanskelig å få effektiv støtte fra de seks "Tilfeldig"-agentene. I starten av et spill har hver agent 43 ordrer å velge mellom. Dersom en av agentene bestemmer seg for å angripe en annen agent, er det 10.12%[2]

[2]For at en agent skal kunne gjennomføre et vellykket angrep mot en av sine motstandere, må den samle mer støtte enn motstanderen som skal angripes. Hver motstander har én tilgjengelig

sannsynlighet for at agenten skal få gjennomført angrepet, gitt at motstanderne er "Tilfeldig"-agenter. Det er enkelt for en optimal agent å vinne mot seks "Tilfeldig"-agenter når den først har erobret én provins. Når agenten har to provinser, blir det mye enklere for agenten å gå til angrep mot de resterende agentene. Det antas derfor at en optimal agent vil vinne minimum 10% av spillene den spiller mot seks "Tilfeldig"-agenter. Dette viser at både LOLA og A2C-agenten har langt igjen til optimalt spill.

I beste fall har LOLA- og A2C-agentene lært seg å koordinere angrep. Det er derfor forventet at disse skal fungere best når de spiller som de seks kopierte agentene. Ettersom LOLA i andre spill har vært bedre enn A2C til å finne strategier for samarbeid [Foerster et al., 2018], forventes det at LOLA-agenten presterer bedre enn A2C-agenten i denne settingen. Som en kan se av tabell 5.1 scorer A2C-agenten høyere enn LOLA-agentene, når de som seks kopier spiller mot én "Tilfeldig"-agent. De seks A2C-agentene scorer samlet i gjennomsnitt 0.933 poeng, mens de seks LOLA-agentene scorer samlet i gjennomsnitt 0.914 poeng. Seks "Tilfeldig"-agenter er forventet å score 0.857 poeng samlet, når de møter én "Tilfeldig"-agent. På bakgrunn av dette gjennomføres tre hypotesetester.

Hypotese 4: I møte med en "Tilfeldig"-agent scorer seks A2C-agenter samlet i gjennomsnitt høyere enn seks "Tilfeldig"-agenter.

La X være den samlede scoren seks A2C-agenter mottar i gjennomsnitt fra spill spilt mot én "Tilfeldig"-agent. Den spesifikke nullhypotesen er $H_0 : \mu = 0.857$. Den alternative hypotesen er $H_1 : \mu_X > 0.857$. Den observerte gjennomsnittsverdien for X er $\bar{x} = 0.933$. Det observerte standardavviket for X er $s = 0.081$. Dette gir en observert t-verdi $t_{obs} = \frac{0.933-0.857}{0.081/\sqrt{1000}} \approx 29.67$. Dermed har man en p-verdi på $P(T > 29.67) \approx 0$. Med dette forkastes nullhypotesen, og det konkluderes med hypotese 4 trolig er korrekt.

Hypotese 5: I møte med en "Tilfeldig"-agent scorer seks LOLA-agenter i gjennomsnitt høyere enn seks "Tilfeldig"-agenter.

La Y være den samlede scoren seks LOLA-agenter mottar i gjennomsnitt fra spill spilt mot én "Tilfeldig"-agent. Den spesifikke nullhypotesen er $H_0 : \mu = 0.857$, mens den alternative hypotesen er $H_1 : \mu_Y > 0.857$. Den observerte gjennomsnittsverdien for Y er $\bar{y} = 0.914$, og det observerte standardavviket er $s = 0.09$. Dette gir en observert t-verdi $t_{obs} = \frac{0.914-0.857}{0.09/\sqrt{1000}} \approx 20.03$. Med dette har

ordre som gir støtte til angrepet, og én ordre som gir støtte til den som blir angrepet. Det finnes tre hendelser hvor agenten som angriper mottar mer støtte enn motstanderen: Dersom motstanderen mottar ingen støtte, dersom motstanderen mottar støtte fra en agent, og dersom motstanderen mottar støtte fra to agenter. Dette gir følgende sannsynlighet for at angrepet skal være vellykket: $\sum_{i=1}^{5} \binom{5}{i,0,5-0-i}(\frac{1}{43})^i(\frac{1}{43})^0(\frac{41}{43})^{5-0-i} + \sum_{i=2}^{4} \binom{5}{i,1,5-1-i}(\frac{1}{43})^i(\frac{1}{43})^1(\frac{41}{43})^{6-1-i} + \binom{5}{i,2,5-2-3}(\frac{1}{43})^3(\frac{1}{43})^2(\frac{41}{43})^{5-2-3} = 0.1012$

man en p-verdi på $P(T > 20.03) \approx 0$, og nullhypotesen forkastes. Det er derfor grunn til å tro at hypotese 5 er sann.

Hypotese 6: I møte med en "Tilfeldig"-agent scorer seks A2C-agenter i gjennomsnitt høyere enn seks LOLA-agenter.

For denne hypotesetesten er den spesifikke nullhypotesen $H_0 : \mu_X - \mu_Y = 0$, og den alternative hypotesen er $H_1 : \mu_X - \mu_Y > 0$. Dette gir en observert t-verdi på $t_{obs} = \frac{(0.933-0.914)-0}{\sqrt{0.09^2/1000+0.081^2/1000}} \approx 4.96$. P-verdien blir dermed $P(T > 4.96) \approx 0$. Med denne p-verdien forkastes nullhypotesen, og det konkluderes med at hypotese 6 trolig er korrekt.

Resultatet fra hypotese 4 og 5 styrker troen på at A2C- og LOLA-agentene presterer bedre enn en "Tilfeldig"-agent. Dette får en også bekreftet når en ser på tabell 5.2. Tabellen viser at seks kopier av LOLA-agenten samlet har vunnet 8.4% av spillene mot en "Tilfeldig"- agent, mens det samme tallet for A2C-agenten er 22.2%. Til sammenlikning endte ikke et eneste spill med seier når kun "Tilfeldig"-agenter spilte mot hverandre.

Det at seks kopier av LOLA- og A2C-agentene presterer såpass mye bedre enn seks kopier av "Tilfeldig"-agenten, styrker forventningen om at de lærer å koordinere angrep. Resultatet viser også at strategien til LOLA- og A2C-agenten er ganske robust i møte med en strategi som velger handlinger etter en uniform sannsynlighetsfordeling. Resultatet fra hypotesetesten for hypotese 6 svekker forventningen om at LOLA-agenten er mer effektiv i koordineringen av angrep enn A2C-agenten.

Som en kan se av tabell 5.1 scorer en "Tilfeldig"-agent betydelig lavere i møte med seks kopier av LOLA-agenten enn det A2C-agenten gjør. "Tilfeldig"-agenten scorer 0.086, mens A2C-agenten scorer 0.203. "Tilfeldig"-agenten scorer i gjennomsnitt 0.067, mens LOLA-agenten i gjennomsnitt scorer 0.077. Basert på disse resultatene gjennomføres det to hypotesetester.

Hypotese 7: I møte med seks kopier av LOLA-agenten scorer A2C-agenten i gjennomsnitt høyere enn en "Tilfeldig"-agent.

La Y være scoren en A2C-agent oppnår i et spill mot seks kopier av LOLA-agenten, og la X være scoren en "Tilfeldig"-agent oppnår i spill mot seks kopier av LOLA-agenten. Den spesifikke nullhypotesen er $H_0 : \mu_Y - \mu_X = 0$. Den alternative hypotesen er $H_1 : \mu_Y - \mu_X > 0$. De observerte gjennomsnittene for X og Y er henholdsvis $\bar{x} = 0.086$ og $\bar{y} = 0.203$. De observerte standardavvikene for X og Y er henholdsvis $s_X = 0.09$ og $s_Y = 0.197$. Dette gir en observert t-verdi på $t_{obs} = \frac{0.203-0.086}{\sqrt{0.197^2/1000+0.09^2/1000}} = 17.08$. Dette gir en p-verdi på $P(T > 17.08) \approx 0$, og nullhypotesen forkastes. Dermed konkluderes det med at hypotese 7 trolig er

sann.

Hypotese 8: I møte med seks A2C-agenter scorer en LOLA-agent i gjennomsnitt høyere enn en "Tilfeldig"-agent.

La X være scoren en "Tilfeldig"-agent oppnår i et spill mot seks kopier av A2C-agenten, og la Y være scoren LOLA-agenten oppnår i spill mot seks kopier av A2C-agenten. Den spesifikke nullhypotesen er $H_0 : \mu_Y - \mu_X = 0$, og den alternative hypotesen er $H_1 : \mu_Y - \mu_X > 0$. De observerte gjennomsnittene for X og Y er henholdsvis $\bar{x} = 0.067$ og $\bar{y} = 0.077$. De observerte standardavvikene for X og Y er henholdsvis $s_X = 0.081$ og $s_Y = 0.089$. Dette gir en observert t-verdi på $t_{obs} = \frac{0.077-0.067}{\sqrt{0.089^2/1000+0.081^2/1000}} = 2.621$ Dermed har man en p-verdi på $P(T > 2.62) \approx 0.001$. Nullhypotesen kan derfor forkastes.

Konklusjonen vedrørende hypotese 7 samsvarer med konklusjonen vedrørende hypotese 1, hvor det ble konkludert med at A2C er bedre enn en "Tilfeldig"-agent. Konklusjonen vedørende hypotese 8 samsvarer med konklusjonen vedørende hypotese 2, hvor det ble konkludert med at LOLA-agenten er bedre enn en "Tilfeldig"-agent. Det at A2C-agenten scorer 0.203 mot seks LOLA-agenter, mens LOLA-agenten kun scorer 0.077 mot seks A2C-agenten, motsier konklusjonen vedrørende hypotese 3. Dette gjør at man ikke uten videre kan rangere LOLA-agenten og A2C-agenten.

Konklusjonen fra "1 vs. 6"-spillene er at A2C- og LOLA-agenten i stor grad har lært seg strategier som fungerer godt når de spiller som den unike agenten. Begge agentene er bedre enn "Tilfeldig"-agenten når de spiller mot seks "Tilfeldig"-agenter. A2C-agenten scorer betydelig høyere i møte med seks LOLA-agenter, enn LOLA-agenten gjør i møte med seks A2C-agenter. I begge tilfeller scorer agenten høyere enn en "Tilfeldig"-agent. Når A2C-agenten og LOLA-agenten spiller som seks kopier av hverandre mot en "Tilfeldig"-agent, scorer de begge høyere enn det en "Tilfeldig"-agent ville ha gjort.

Resultatet fra "1 vs. 6"-spillene tyder på at A2C-agenten er bedre til å lære "Pure"-utgaven av Diplomacy, men det er vanskelig å gjøre en endelig rangering av agentene. Det å spille som en unik agent og det å spille som seks kopier av en agent kan ikke direkte sammenlignes. Det er derfor vanskelig å f.eks. justere scoren for A2C når den spiller som seks kopierte agenter, slik at den kan summeres opp med scoren for A2C når den spiller som unik agent. Om en tok totalscoren for spillene hvor A2C spiller som unik agent, og la den sammen med totalscoren for A2C når den spiller som kopiert agent, ville poengsummen når A2C spiller som kopiert agent få for stor betydning for summen. Det er heller ikke mulig å bruke gjennomsnittsscoren til de kopierte agenten i summen.

For å kunne gjøre en rangering av de forskjellige agentene, har det blitt gjen-

Agent	Gjennomsnittsscore
Tilfeldig	0.103
A2C	0.168
LOLA	0.157

Tabell 5.3: Tabell som viser det totale resultatet for bestemte kopier av A2C-, LOLA- og "Tilfeldig"-agenten. Resultatet er et gjennomsnitt fra 3000 spill, hvor agentene byttet på å ha tre spillende kopier av seg selv. De agentene som ikke spilte med tre kopier av seg selv, spilte med to kopier hver.

nomført en turnering hvor to av agentene spiller som to land og en agent spiller som tre land. De tre agentene bytter på om de spiller som tre land eller to land. I stedet for å se på alle kopiene av agenten som én agent, blir det kun sett på resultatet til én bestemt kopi av agentene. Dette gjør de nevnte utfordringene ved å rangere agentene mindre. Resultatet fra denne turneringen er vist i tabell 5.3. Som en kan se av tabell, scoret A2C-agenten høyest med 0.168 poeng i gjennomsnitt over 3000 spill. LOLA- og "Tilfeldig"-agentene scoret henholdsvis 0.157 og 0.103 poeng i gjennomsnitt. Dette resultatet gir en indikasjon på at A2C-algoritmen kan være bedre til å lære "Pure"-utgaven av Diplomacy, sammenliknet med LOLA-algoritmen. Resultatet kan også tyde på at LOLA-agenten ikke har blitt bedre til å samarbeide enn A2C-agenten.

5.1.3 Koalisjonsanalyse

For å vurdere agentenes evne til samarbeid, har det blitt gjennomført en koalisjonsanalyse. Resultatene fra koalisjonsanalysen vises i tabell 5.4, 5.5 og 5.6. Koalisjonsanalysen er basert på spill hvor spillerne er sju kopier av den samme agenten. For hver agent har det vært spilt 1000 spill. I forbindelse med "Pure"-utgaven, må resultatet fra koalisjonsanalysen tolkes på en litt annen måte enn i forbindelse med "France vs. Austria". I begynnelsen av "Pure" er det kun mulig å gi støtte til motstandere. Det vil si at en agent som aldri avanserer vil ha en "cross-power support ratio" på 100%, gitt at den har gitt minst en "SUPPORT"-ordre. Et lavere tall vil nødvendigvis bety at agenten har klart å avansere.

Som en kan se av tabell 5.4 oppnådde "Tilfeldig"-agenten en "cross-power support ratio" på 98.75%. Det høye tallet til "Tilfeldig"-agenten samsvarer med at denne agenten har vist seg å ha problem med å gjøre store avansement. I løpet av de 1000 spillene "Tilfeldig"-agenten spilte mot seg selv, endte ingen spill i seier.

A2C-agenten oppnådde en "cross-power support ratio" på 86.10%, mens det samme tallet for LOLA-agenten var 94.94%. Spørsmålet blir da om forskjellen kan skylles at A2C-agenten har avansert mer, og dermed hatt færre muligheter

Agent	Cross-power support ratio.
Tilfeldig	98.75%
A2C	86.10%
LOLA	94.94%

Tabell 5.4: Tabell som viser "cross-power support ratio".

Agent	Effective cross power-support ratio'
Tilfeldig	1.16%
A2C	28.16%
LOLA	20.08%

Tabell 5.5: Tabell som viser "effective cross power-support ratio".

til å gi støtte til motspillere, enn LOLA-agenten. I gjennomsnitt varte et spill i 9.62 runder når A2C-agenten spilte. Det samme tallet for LOLA-agenten var 9.87. 27.2% av spillene endte i seier når A2C-agenten spilte, mot 13.1% for LOLA-agenten. Det er derfor en mulighet for at forskjellen i "cross-power support ratio" mellom A2C- og LOLA-agenten skyldes at A2C-agenten har vært mer effektiv i å avansere, og ikke at LOLA-agenten har lært seg å samarbeide bedre enn A2C-agenten.

"Effective cross power-support ratio" er vist i tabell 5.5. Som en kan se av tabellen er "effective cross-power support ratio" lav for "Tilfeldig"-agenten. "Tilfeldig"-agenten har en "effective cross-power support ratio" på 1.16%. Tilsvarende tall for A2C-agenten og LOLA-agenten er henholdsvis 28.16% og 20.08%.

"Coordinated cross-power support move ratio" er vist i tabell 5.6. Som en kan se av tabellen er forholdstallet for "Tilfeldig"-agenten lavt, med 2.45%. LOLA-agenten har et høyere forholdstall med 40.47%, mens A2C-agenten scorer høyest med 59.45%. Dette viser at både A2C- og LOLA-agenten er betydelig bedre til å koordinere angrep enn en "Tilfeldig"-agent. I tillegg viser det at A2C-agenten er bedre til å koordinere angrep enn LOLA-agenten. Disse tallene tilsier at A2C-agenten er noe bedre til å samarbeide enn LOLA-agenten. Man har dermed ikke oppnådd samme resultat som Foerster et al. [2018] hvor LOLA-algoritmen gene-

Agent	Coordinated cross-power support move ratio
Tilfeldig	2.45%
A2C	59.45%
LOLA	40.47%

Tabell 5.6: Tabell som viser "coordinated cross-power support move ratio".

rerte samarbeid mellom agentene. Dette svekker hypotesen om at LOLA-agenten er bedre til lære å samarbeide i Diplomacy enn A2C-agenten.

5.2 France vs. Austria

Gitt tilgjengelige ressurser ble det vanskelig å generere nok data for utgaven "France vs Austria". Den største utfordringen med LOLA er hvor ressurskrevende det er å beregne den andre ordens deriverte. LOLA-oppdateringen kan beregnes som en "one-step"-metode, hvor LOLA agenten oppdateres hver gang den observerer en ny tilstand og får en ny belønning fra omgivelsene, eller som en "batch"-metode, hvor en "batch" av oppsamlet data brukes til å oppdatere agenten. Disse metodene gir ulike utfordringer med tanke på ressursbruk.

Det spesielle med "one-step"-utgaven av LOLA, sammenliknet med "one-step"-utgaven av A2C, er at den andre ordens deriverte (likning 4.5) for tidssteg t krever informasjon om agentens observasjoner og handlinger for tidsstegene $t^{'} < t$. Dette gjør at agenten må lagre data fra tidligere tidssteg.

Den største begrensningen til "one-step"-utgaven av LOLA er tiden det tar å beregne LOLA-oppdateringen. "One-step"-utgaven av A2C krever én gradientberegning pr. steg. Dersom LOLA-oppdateringen gjøres på en måte som ikke krever svært mye minne, må gradienten beregnes tre ganger pr. motstander. I tillegg må man beregne gradienten som brukes i A2C. Denne metoden å beregne LOLA-gradienten (se likning 4.2) gjøres ved å multiplisere radvektoren $(\nabla_{\theta_j} log\pi_j(A_{j,t} \mid S_t)\gamma^t G_{i,t})^T$ fra likning 4.4 med kolonnevektoren $(\gamma^t R_{j,t} \sum_{l=0}^{t} \nabla_{\theta_i} log\pi_i(A_{i,l} \mid S_l))$ fra likning 4.5. Resultatet multipliseres deretter med radvektoren $(\sum_{l=0}^{t} \nabla_{\theta_j} log\pi_j(A_{j,l} \mid S_l))^T$ fra likning 4.5. På den måten trenger man å holde maksimalt to vektorer i minnet samtidig. I A2C og LOLA, utgjør tiden det tar å beregne gradienten en stor del av kjøretiden. For "France vs. Austria", hvor hver spiller har én motstander, gjennomføres LOLA-algoritmen fire ganger så mange gradient-beregninger som A2C. Dette gjør "one-step"-utgaven av LOLA betydelig tregere enn A2C.

Et annet alternativ er å multiplisere gradienten i likning 4.4 med matrisen i likning 4.5. En ulempe med denne metoden er at matrisen krever svært mye minne når antall parametere i policynettverket blir stort. Policynettverket, beskrevet i kapittel 4, har ca. tre millioner parametere. Dette gjør at matrisen får en størrelse på ca. 9000 milliard elementer. Dersom alle parametere er lagret med 32-bits flyttall, vil matrisen alene kreve ca. 36 terabyte med minne. Gitt tilgjengelig maskinvare, er dette en uhåndterbar størrelse.

Det finnes også en tredje måte å beregne LOLA-oppdateringen på. Denne metoden går ut på å beregne gradienten i likning 4.4 og multipliserer den med $(\gamma^t R_{j,t} \sum_{l=0}^{t} \nabla_{\theta_i} log\pi_i(A_{i,l} \mid S_l)) * \prod_{l=0}^{t} \pi_j(A_{l,j} \mid S_l)$, og deretter beregner gradi-

enten med hensyn på motstanderen j sine parametere. Denne metoden er vanskeligere å analysere enn de tidligere nevnte metodene for å regne ut LOLA-gradienten, men forsøk viser at også denne metoden fort krever svært mye minne.

Utfordringen med de to sistnevnte måtene å beregne LOLA-oppdateringen er også lik for batch-utgaven av LOLA. Basert på analysen presentert over virker det ikke realistisk å anvende LOLA på en av de større utgavene av Diplomacy. Om man skal kunne bruke LOLA, er en i så fall avhengig av å bruke et policynettverk som er langt mindre enn det som er foreslått i kapittel 4. Et slikt nettverk ville vært veldig annerledes fra nettverkene brukt av Paquette et al. [2019], Anthony et al. [2020], Gray et al. [2021] og Bakhtin et al. [2021].

5.3 Usikkerhet i resultatene

At det ikke finnes et objektivt mål på en agents prestasjoner, som er uavhengig av agentens motstandere, er ett av de største usikkerhetsmomentene ved resultatet presentert i dette kapittelet. Når agentene ble trent, ble det gjennomført "1 vs. 6"-spill mellom agentene og "Tilfeldig"-agenten. "Tilfeldig"-agenten fungerte som en "benchmark"-agent, og resultatet fra "1 vs. 6"-spillet ble brukt til å gjøre en vurdering av utviklingen i læringen til agentene. Denne metoden vil kun vise hvor godt agentene gjør det mot "Tilfeldig"-agenten, og ikke hvordan agenten presterer mot andre agenter. Når det velges ut en "utgave" av agentene som skal delta i turneringen, forplanter denne usikkerheten seg til turneringsresultatene. Hvordan A2C-agenten presterer i forhold til LOLA-agenten i turneringen, er ikke nødvendigvis det beste en A2C-agent kan få til. Om man hadde en ferdig trent LOLA-agent, som presterer det beste en LOLA-agent kan prestere i møte med en A2C-agent, og brukte LOLA-agenten som "benchmark" under treningen av A2C-agenten, kunne man endt opp med en annen "utgave" av A2C-agenten i turneringen. Turneringsresultatet kunne dermed sett annerledes ut.

Det er også en risiko for at de valgte "utgavene" av LOLA- og A2C-agentene kun presterer likt med gjennomsnittet i møte med en "Tilfeldig"-agent (se figur 5.1). I møte mot hverandre, kan det hende de valgte "utgavene" ikke representerer gjennomsnittet. I teorien kunne noe av denne usikkerheten kanskje blitt fjernet ved å gjennomføre en turnering hvor de ulike "utgavene" av A2C- og LOLA-agentene deltok. En slik turnering er svært ressurskrevende, og dermed ikke realistisk å gjennomføre.

En annen usikkerhet ved treningsresultatene er at de er basert på kun 10 kjøringer av treningsprosessen. Antallet ble valgt på bakgrunn av hvor ressurskrevende det er å gjennomføre en runde med trening. Å kjøre en del flere treningsprosesser kunne gitt et bedre estimat på hvor godt agentene gjør det mot "Tilfeldig"-agenten i løpet av treningen.

Kapittel 6

Konklusjon

Å lære å spille Diplomacy er en utfordrende oppgave for en RL-agent. I Diplomacy har hver spiller et stort antall mulige trekk pr. runde. Alle spillerne gjennomfører trekkene sine samtidig. I tillegg er det en fordel å kunne inngå samarbeid med de andre spillerne. Disse egenskapene kjennetegner mange andre problemer, inkludert problemer en agent med kunstig generell intelligens er forventet å kunne lære seg. Å få en agent til å lære seg Diplomacy, vil derfor kunne være et lite steg i riktig retning av å oppnå kunstig generell intelligens.

På veien til å skape en agent som innehar kunstig generell intelligens er det mange utfordringer som må løses. I denne rapporten har det vært sett på noen av disse utfordringer. Den ene utfordringen innebærer at agenter med kunstig generell intelligens er nødt til å kunne lære oppgaver uten å ha tilgang på data som viser hvordan oppgaven skal løses. For Diplomacy har dette til en viss grad blitt oppnådd tidligere av Bakhtin et al. [2021], som klarte å lære en agent å spille "France vs. Austria"-utgaven av Diplomacy utelukkende fra selvspill.

En annen utfordring er å få agenter til å samarbeide, når dette lønner seg. Som en følge av dette har en algoritme, kalt LOLA [Foerster et al., 2018], blitt testet ut på Diplomacy. LOLA har tidligere vist seg å generere samarbeid mellom agenter. LOLA har ikke tidligere blitt anvendt på Diplomacy, og hypotesen har vært at LOLA skulle ha en fordel over andre algoritmer fordi den har vist seg å generere samarbeid. Tidligere forsøk på å lære en maskin å spille Diplomacy har oppnådd gode resultater, men ingen av forsøkene har innebåret å anvende en algoritme som har vist seg å generere strategier for samarbeid.

I denne rapporten er det forsøkt å besvare to forskningsspørsmål. Det ene forskningsspørsmålet handler om hvilke algoritmer som egner seg best til å lære en agent Diplomacy fra selvspill. Det andre forskningsspørsmålet handler om hvilke nettverksarkitekturer for policy- og verdinettverk som egner seg best når

en agent skal lære seg Diplomacy.

Forskningsspørsmål 1: *Hvilken reinforcement learning-algoritme egner seg best til å lære No-Press Diplomacy?*

LOLA-algoritmen ble sammenliknet med A2C-algoritmen ettersom A2C har likhetstrekk med LOLA, samtidig som at A2C ikke forsøker å påvirke motstandernes læring. Ettersom det tidligere ikke har eksistert en "benchmark"-agent for "Pure"-utgaven av Diplomacy, ble en agent som velger handlinger etter en uniform sannsynlighetsfordeling brukt til dette.

I forsøket på å anvende LOLA på det originale spillbrettet, ble det oppdaget at LOLA-algoritmen hadde visse begrensninger. Avhengig av hvordan en velger å regne ut LOLA-oppdateringen, er problemet med algoritmen at den enten har svært lang kjøretid, eller at den krever for mye minne. Det må derfor konkluderes med at LOLA-algoritmen ikke kan brukes til å lære spill som foregår på originalbrettet i Diplomacy.

LOLA-algoritmen ble også testet på "Pure"-utgaven av Diplomacy. Resultatet fra "Pure"-utgaven viser at både LOLA- og A2C-agenten er bedre enn en "Tilfeldig"-agent. Resultatet viser også at A2C trolig er bedre til å lære "Pure"-utgaven enn LOLA.

Det har også blitt testet i hvilken grad de ulike agentene har lært å samarbeide. Ved å se på hvor godt agentene koordinerte "SUPPORT"-ordrer, konkluderes det med at A2C-agenten er bedre til å koordinere slike ordre, sammenliknet med LOLA-agenten. Av den grunn svekkes hypotesen om at LOLA har en fordel i Diplomacy fordi den genererer samarbeid. Sammenliknet med "Tilfeldig"-agenten, var LOLA-agenten og A2C-agenten betydelig bedre til å koordinere "SUPPORT"-ordrer.

Forskningsspørsmål 2: *Når No-Press Diplomacy skal læres, hvilken nettverksarkitektur egner seg best til å lære policy- og verdifunksjoner?*

Den valgte nettverksarkitekturen er basert på det som tidligere har vist seg å fungere godt til å lære Diplomacy. Det har vært gjort mindre endringer for å tilpasse policy- og verdinettverket til "Pure"-utgaven. Tilpasningen har innebåret å velge vekk funksjonalitet som egner seg godt når nettverkene er spesielt dype. Dette har blant annet vært å bytte ut LSTM med et enklere RNN, samt fjerne dropout-lag og batch-normalisering. Disse endringene har blant annet redusert antall parametere i nettverkene, noe som har gjort det enklere å regne ut policy-gradienten og LOLA-oppdateringen.

6.1 Bidrag

Denne rapporten beskriver teori og arbeid som er relatert til oppgaven med å lage en RL-agent som lærer å spille No-Press Diplomacy fra selvspill.

Rapporten bidrar også ved å beskrive fremgangsmåten ved, og resultatet av, å anvende LOLA-algoritmen på "France vs. Austria"- og "Pure"-utgaven av Diplomacy. Rapporten beskriver hvordan LOLA-algoritmen ikke egner seg til å spille "France vs. Austria"-utgaven, og hvordan den ikke har klart å prestere bedre enn A2C-algoritmen i "Pure"-utgaven. Resultatet av å anvende LOLA-algoritmen på Diplomacy, viser at algoritmen har begrensninger som gjør det nødvendig å finne andre metoder for å generere samarbeid mellom agenter.

Ettersom A2C-agenten og LOLA-agenten presterer bedre enn en "Tilfeldig"-agent, kan disse brukes som "benchmark"-agenter for "Pure"-utgaven av Diplomacy. I fremtidig arbeid kan disse agentene brukes til å vurdere prestasjonen til andre agenter.

6.2 Fremtidig arbeid

Det er identifisert to begrensninger ved LOLA-algoritmen når den anvendes på originalbrettet i Diplomacy: kjøretid og krav til minnestørrelse. Problemet med kjøretid og krav til minne skyldes i hovedsak kombinasjonen av algoritmen og det valgte policynettverket. Det er derfor to områder man kan forbedre om man ønsker å anvende en metode tilsvarende den som er brukt i dette prosjektet; Man kan enten endre algoritmen eller policynettverket. Om en velger å gjøre policynettverket mindre, vil det måtte bli betydelig annerledes enn det som tidligere har vært anvendt på Diplomacy.

Det er mulig å øke antall IMPALA-arbeidsprosesser, og se om dette fører til redusert varians i treningsresultatet. Dette vil eventuelt kreve større beregningsressurser enn det som har vært tilgengelig i arbeidet med denne oppgaven. I tillegg er det mulig å forbedre IMPALA-implementasjonen brukt i denne oppgaven ved å implementere V-trace-algoritmen og la arbeidsprosessene kjøre asynkront.

Ettersom LOLA ikke kunne brukes til å lære originalutgaven av Diplomacy, har det ikke vært enkelt å sammenlikne denne med tidligere forsøk på å lære Diplomacy, slik som arbeidet til Paquette et al. [2019], Gray et al. [2021], Anthony et al. [2020] og Bakhtin et al. [2021]. Det kan derfor være interessant å teste ut metodene til de nevnte forfatterne på "Pure"-utgaven av Diplomacy.

Det kan tenkes at metoder tilsvarende Raileanu et al. [2018] og Lowe et al. [2020] sine metoder kan tilpasses Diplomacy. Raileanu et al. bruker agenters mål i beregningen av policy og tilstandsverdier. Lowe et al. kombinerer Q-nettverk og policynettverk, samt bruker forskjellige del-policyer for å gjøre en agent mer robust mot endringer i motstandernes policyer. Selv om metodene ikke direkte

kan anvendes på Diplomacy, kan det være interessant å se på om tilsvarende metoder gir en fordel når Diplomacy skal læres.

Bibliografi

Anthony, T., Eccles, T., Tacchetti, A., Kramár, J., Gemp, I., Hudson, T. C., Porcel, N., Lanctot, M., Pérolat, J., Everett, R., Werpachowski, R., Singh, S., Graepel, T., og Bachrach, Y. (2020). Learning to play no-press diplomacy with best response policy iteration. ArXiv:2006.04635.

avalonhill.com (u.d.). Avalon hill diplomacy cooperative strategy board game, ages 12 and up, 2-7 players. `https://www.avalonhill.com/en-us/product/avalon-hill-diplomacy-cooperative-strategy-board-game-ages-12-and-up-2-7-players:09A402C7-4CA2-4E9D-9449-4592B2066011`. Lastet ned: 2021-08-31.

Axelrod, R. (1980). Effective choice in the prisoner's dilemma. *Journal of conflict resolution*, 24(1):3–25.

Bakhtin, A., Wu, D., Lerer, A., og Brown, N. (2021). No-press diplomacy from scratch. ArXiv:2110.02924.

Brown, N. og Sandholm, T. (2019). Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., og Kavukcuoglu, K. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. ArXiv;1802.01561.

fdeloche (2017a). Long short-term memory.svg. `https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg`. Lastet ned: 2022-03-23. Lisensiert under CC BY 4.0 (`https://creativecommons.org/licenses/by/4.0/`).

fdeloche (2017b). Recurrent neural network unfold.svg. `https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg`. Lastet ned: 2022-03-23. Lisensiert under CC BY 4.0 (`https://creativecommons.org/licenses/by/4.0/`).

Foerster, J. N., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., og Mordatch, I. (2018). Learning with opponent-learning awareness. ArXiv:1709.04326.

Frisvold, F. og Moe, J. G. (2004). *Statistikk for Ingeniører*. Fagbokforlaget, 1. utgave.

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2. utgave.

Goodfellow, I., Bengio, Y., og Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Gray, J., Lerer, A., Bakhtin, A., og Brown, N. (2021). Human-level performance in no-press diplomacy via equilibrium search. ArXiv:2010.02923.

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

Hatløe, A. M. (2021). Selvlært diplomacy-agent: Forslag til fremgangsmåte.

He, K., Zhang, X., Ren, S., og Sun, J. (2015). Deep residual learning for image recognition. ArXiv:1512.03385.

Heinrich, J. og Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. ArXiv:1603.01121.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., og Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. ArXiv:1207.0580.

Ioffe, S. og Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv:1502.03167.

Kipf, T. N. og Welling, M. (2016). Semi-supervised classification with graph convolutional networks. ArXiv:1609.02907.

Kofod-Petersen, A. (2021). Finding literature - and how to read it. `https://research.idi.ntnu.no/aimasters/files/2021_09_28_SLR.pdf`. Lastet ned: 2022-05-25.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., og Mordatch, I. (2020). Multi-agent actor-critic for mixed cooperative-competitive environments. ArXiv:1706.02275.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., og Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. ArXiv:1602.01783.

Nash, J. (1951). Non-cooperative games. *Annals of mathematics*, pages 286–295.

Paquette, P., Lu, Y., Bocco, S., Smith, M. O., Ortiz-Gagne, S., Kummerfeld, J. K., Singh, S., Pineau, J., og Courville, A. C. (2019). No press diplomacy: Modeling multi-agent gameplay. ArXiv:1909.02128.

Porge og Edinborgarstefan (2006). Diplomacyboard.png. `https://commons.wikimedia.org/wiki/File:DiplomacyBoard.png`. Lastet ned: 2021-10-01. Lisensiert under CC BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0/`).

Raileanu, R., Denton, E., Szlam, A., og Fergus, R. (2018). Modeling others using oneself in multi-agent reinforcement learning. In *International conference on machine learning*, pages 4257–4266. PMLR.

Russell, S. og Norvig, P. (2016). *Artificial Intelligence*. Pearson Education, 3. globale utgave.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Shapiro, A., Fuchs, G., og Levinson, R. (2002). Learning a game strategy using pattern-weights and self-play. In *International Conference on Computers and Games*, pages 42–60. Springer.

Shoham, Y. og Leyton-Brown, K. (2008). *Multiagent Systems*. Cambridge University Press.

Silver, D., Singh, S., Precup, D., og Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.

Sutton, R. S. og Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press.

Thoma, M. (2013). Feedf-forward-perceptron.svg. `https://commons.wikimedia.org/wiki/File:Feed-forward-perceptron.svg`. Lastet ned: 2022-03-24. Lisensiert under CC BY 3.0 (`https://creativecommons.org/licenses/by/3.0/`).

Thoma, M. (2014). Perceptron-unit.svg. https://commons.wikimedia.org/wiki/File:Perceptron-unit.svg. Lastet ned: 2022-03-24 Lisensiert under CC0 1.0 (https://creativecommons.org/publicdomain/zero/1.0/deed.en).

Vedlegg

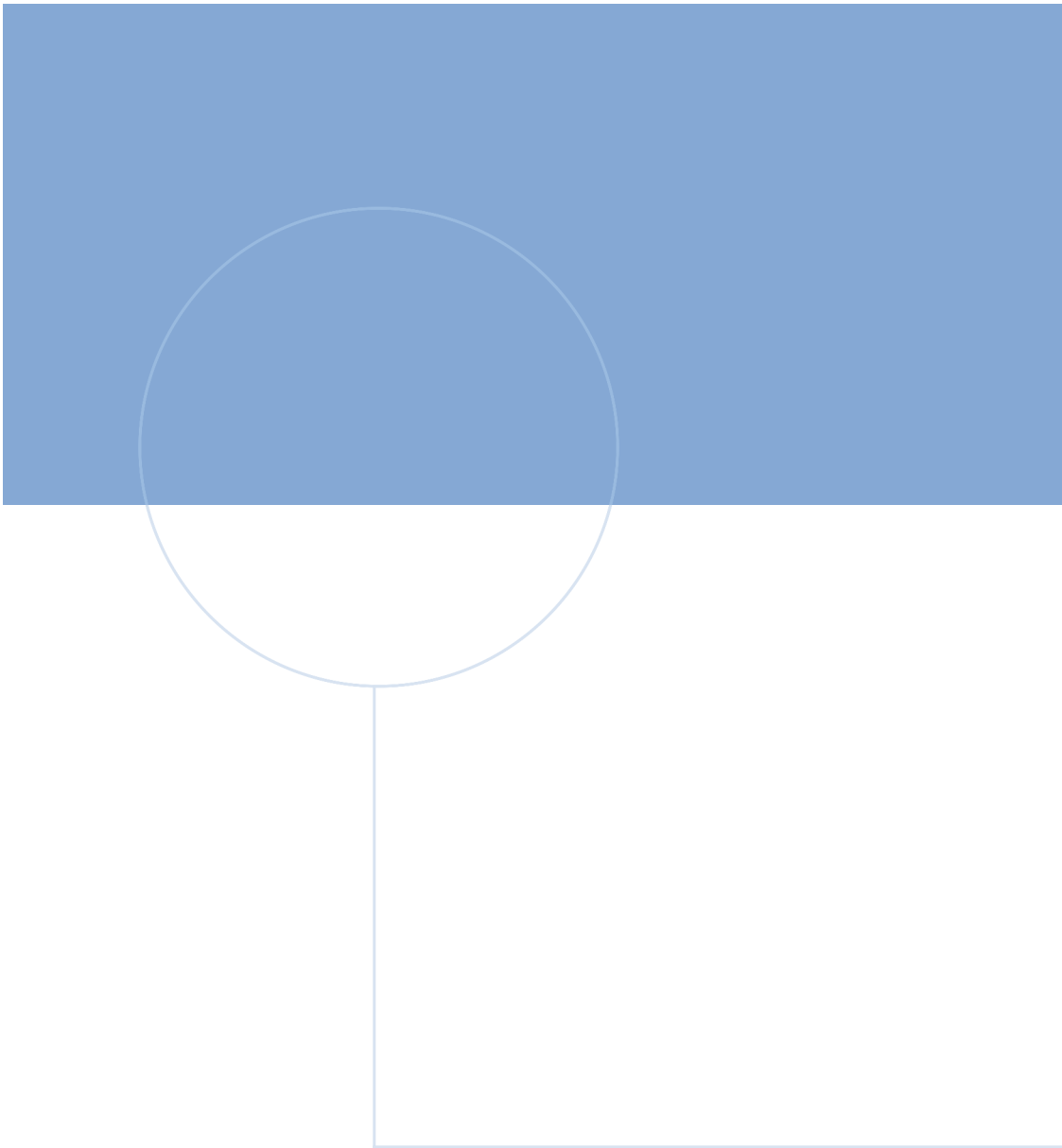
A Hyperparametere

Under vises hyperparameterverdiene brukt til å generere resultatet gitt i kapittel 5.

Parameter	Parameterverdi
Arbeidsprosesser (IMPALA-spesifikk)	3
Diskonteringsfaktor	0.99
Læringsrate motstander (LOLA-spesifikk)	0.005
Opponent modelling (LOLA-spesifikk)	False
Læringsrate verdifunksjon	0.0003
Aktiveringsfunksjon verdifunksjon	softmax
Optimizer verdifunksjon	SGD
Tapsfunksjon verdifunksjon	MSE
Læringsrate policy	0.0003
Optimizer policy	SGD
Embedding-størrelse tidligere ordre	10
Embedding-størrelse land	60
Embedding-størrelse predikert ordre	80
RNN-type	SimpleRNN (Keras)
Embedding-størrelse sesong	16
Dybde GNN-koder for nåværende fase	1
Dybde GNN-koder for forrige ordrefase	1
Dybde samlet GNN-koder	1
GNN-koder for nåværende fase output-størrelse	50
GNN-koder for forrige ordre fase output-størrelse	60
Samlet GNN-koder output-størrelse	110
dropout-rate	0.0
Batch normalisering	False

B Kildekode

Kildekode er vedlagt denne rapporten. I zip-filen med kildekode finnes også den trente LOLA- og A2C-agenten som ble brukt i turneringen.



NTNU

Norwegian University of
Science and Technology