Gitt en arkitektur på et nevralt nettverk. Hvordan kan observasjonsrommet optimaliseres for å oppnå raskere trening?

Av

Espen Kalleberg, Håkon Harnes,
Mathias Heggelund og Svein Jakob Høie

Norges Teknisk-Naturvitenskapelige Universitet

Trondheim, 2. september 2021

Espen Kalleberg

Håkon Harnes

Mathias Heggelund

Svein Jakob Høie

Abstrakt

Hensikten med denne oppgaven er å få erfaring med å løse en eller flere problemstillinger ved hjelp av ulike metoder innen maskinlæring. Oppgaven undersøker hvordan observajsonsrommet kan optimaliseres for å oppnå raskere trening. Helt konkret ser vi på optimaliseringene grayscale, redusert input, frame stacking og action repeat. Disse optimaliseringene er implementert i environmentet CarRacing-v0, som er løst ved hjelp av Proximal Policy Optimization. Resultatet viser at action repeat er helt essensiell for å oppnå raskere trening. Grayscale og frame stacking fører også til raskere trening dersom de benyttes sammen med action repeat. Redusert input virker mot sin hensikt, og fører til tregere trening.

Innholdsfortegnelse

Figurer

1 Introduksjon

Denne oppgaven er gitt i faget *TDAT3025 Anvendt maskinlæring med prosjekt* på studiet *Dataingeniør* ved *Norges teknisk-naturvitenskapelige universitet* i Trondheim. Hensikten med oppgaven er å få erfaring med å løse en eller flere problemstillinger ved hjelp av ulike metoder innen maskinlæring.

Reinforcement learning (RL) er et av feltene innen maskinlæring som er under rask utvikling. Hovedprinsippet går ut på at maskinen, også kalt agenten, lærer gjennom prøving og feiling. Den får tilbakemeldinger i form av rewards. Målet til agenten er å maksimere summen av rewards over tid.

Denne oppgaven ser nærmere på environmentet *CarRacing-v0* fra OpenAI Gym [3]. Det er et enkelt environment der målet er å kontrollere en bil slik at den holder seg på veien. Dersom bilen kjører utenfor veien blir den straffet med negativ reward. For å løse dette problemet har vi brukt Proximal Policy Optimization (PPO). PPO er en av de nyeste algoritmene innen reinforcement learning. Den er enklere å implementere enn tidligere algoritmer, samtidig som den er like effektiv.

Problemstillingen vår går ut på å undersøke hvordan observasjonsrommet kan optimaliseres for å oppnå raskere trening. Først tar vi for oss effekten de ulike optimaliseringene har hver for seg. Deretter ser vi på sammensetninger av disse.

2 Teori

2.1 Begreper

Det er mange begreper innenfor reinforcement learning. Denne seksjonen gir en oversikt over de mest sentrale.

Agenten er selve systemet som samhandler med enviromentet. Her, et nevralt nettverk.

Action er en handling som agenten kan utføre i environmentet, for eksempel svinge til høyre eller venstre.

Staten er tilstanden som environmentet befinner seg i. En agent vil påvirke staten ved å utføre actions.

Policy definerer hvordan en agent skal oppføre seg. Det vil si, hvilken action som skal utføres i en gitt state. Hvert $(state, action)$ par som blir utført av agenten kalles for et **step**.

En **episode** består av en sekvens av **steps**.

Reward $r(s, a)$ defineres som belønningen en agent mottar ved et $(state, action)$ par. Målet med reinforcement learning er å maksimere summen av rewards.

Value function $V(s)$ for en state er den totale summen av forventede rewards en agent vil kunne oppnå fra nevnte state ved slutten av episoden.

Action-value function $Q(s, a)$ er den totale summen av forventede rewards en agent vil kunne oppnå ved å uføre en gitt handling i en gitt state ved slutten av episoden.

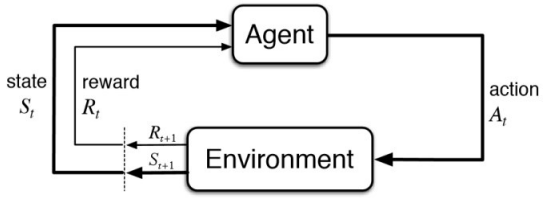
Discount γ definerer om umiddelbare eller framtidige rewards skal prioriteres. En lav discount vil føre til at agenten tenker kortsiktig, mens en høyere discount vil føre til at agenten tenker langsiktig.

Policy gradients brukes for å justere policyen ved å gjøre actions som gir høy reward mer sannsynlige, og actions som gir lavere reward mindre sannsynlige.

Environment er selve oppgaven eller simuleringen som en agent samhandler med. Innenfor

environmentet kan agenten gjøre observasjoner, utføre actions og få rewards for sine actions. Et environment er derfor nødt til å bestå av tre sentrale komponenter:

- En simulering, noe som agenten har som mål å løse
- En tilstandsvektor, som representerer den interne staten til simuleringen
- Et reward system, som belønner agenten ut i fra actions som blir gjort



Figur 1: Illustrasjon av sentrale begreper

2.2 Reinforcement learning

Reinforcement learning er en form for maskinlæring. Det går ut på at agenten løser et problem ved å prøve seg fram. For hvert step agenten tar, vil den få en tildelt en reward. Rewarden er enten positiv eller negativ, stor eller liten, og agenten vil tilpasse seg deretter. Målet til agenten er å maksimere summen av rewards innen slutten av en episode.

Agenten vil i begynnelsen ta tilfeldige valg, men etterhvert som den tilegner seg erfaring, vil den ta mer bevisste valg som den vet gir god tilbakemelding. Det hele kan modelleres som en Markov Decision Process.

2.3 Markov Decision Process

RL-algoritmer kan som sagt modelleres som en Markov Decision Process (MDP). En MDP-prosess har som formål å overføre en agent fra en state til en annen, der sannsynligheten for de forskjellige overgangene baserer seg på den nåværende staten og hvilken action agenten tar. For hver action som utføres vil agenten få en reward r_t. Målet til agenten er å maksimere summen av alle fremtidige rewards. Dette kalles for Gain, G, og er for hvert timestep t gitt

ved:

$$G_t = \sum_{k=t}^{\infty} r_k \gamma^{k-t}$$

der $\gamma \in [0, 1]$ er discounten som kontrollerer hvordan rewards vektes. En lav discount fører til at umiddelbare rewards foretrekkes. Agenten sies da å tenke kortsiktig. En høy discount vil derimot prioritere framtidige rewards. Agenten tenker da langsiktig.

2.4 Model-based vs Model-free RL

Model-based RL bruker erfaringen den har fått fra tidligere episoder til å konstruere en intern modell, ofte kalt world model, med de forskjellige overgangene og den umiddelbare rewarden. Actions vil da bli valgt ved å se til denne modellen og velge en action basert på erfaring.

Model-free RL bruker derimot erfaringen fra episodene direkte til å optimalisere nettverket, noe som kan resultere i like optimal utførelse, men uten det ekstra arbeidet en world model medfører. PPO havner under denne kategorien.

2.5 Policy Optimization

Det er to hovedprinsipper innen model-free RL: Q-læring og Policy optimization. Sistnevnte anvendes i PPO.

Hovedtrinnene for optimaliseringen er som følger; Først regnes policy score function ut. Dette er et mål på 'kvaliteten' til en policy. Deretter oppdateres vektene i nettverket ved å bruke en gradient ascent metode.

2.6 Actor-critic algoritmer

Algoritmen som brukes i dette prosjektet er sammenlignbar med advantage actor-critic (A2C) læringsmetoden [4].

Kjernekonseptet er at det eksisterer to nevrale nettverk: Et for agenten, ofte kalt policy network, og et som kalles value network. Agenten tar den nåværende staten, s_t, som input og får en action, a_t, som output. Denne actionen utføres av agenten i environmentet, som

fører til at environmentet havner i en ny state. Under en episode vil en agent få en reward, r_t, for hver action som utføres.

Rewardene agenten har oppnådd brukes av critic network til å lære å estimere en Gain for hvert step. Critic network kan defineres som:

$$V^{\pi}(s) = \hat{\mathbb{E}}_{\pi}[G_t | s_t = s]$$

og Monte-Carlo læreregelen som brukes til å oppdatere nettverket er:

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha(G_t - V^{\pi}(s_t))$$

der α er læringsraten.

Differansen mellom critic network sin prediksjon og den faktiske oppnåde rewarden for en state kan anses som et estimat på hvor god eller dårlig actionen er i forhold til hva som ble forventet. Denne er verdien kalles advantage $\hat{A}_t$, og defineres slik:

$$\hat{A}_t = G_t - V^{\pi}(s_t)$$

Advantage brukes for å trene opp en agent slik at actions som resulterte i overraskende god reward blir mer sannsynlige og actions som resulterte i en dårlig reward blir mindre sannsynlige.

2.7 Proximal Policy Optimization

Proximal Policy Optimization (PPO) ble introdusert i 2017 av OpenAI som en forbedring på Trust Region Policy Optimization (TRPO). PPO er en simplere policy, som er kompatibel med de fleste nevrale nettverk, uten at det går på bekostning av effektiviteten til TRPO. PPO kan anvendes på både diskret og kontinuerlige action space, og har som hensikt å få størst mulig forbedring på et step, uten at stepet blir så brått at det forårsaker dypp i

prestasjonen. Før vi forklarer PPO nærmere, er det hensiktsmessig å ta en titt på grunnen til at PPO ble utviklet.

Tidligere typiske estimatorfunksjoner har formen:

$$\hat{E}_t \left[\bigtriangledown_\theta \, log \, \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

Problemet med denne typen estimatorer er at de kan lede til oppdateringer som er for voldsomme, og således ødelegge policyen. TRPO ble derfor introdusert som et svar på dette. I TRPO sørger man for å ikke bevege seg for langt vekk fra den opprinnelige policyen. Dermed risikerer man ikke å forkaste verdifulle erfaringer som allerede er opptjent. Måten dette gjøres på er ved å introdusere en begrensning KL på estimatorfunksjonen:

$$\hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{gammel}}(a_t|s_t)} \hat{A}_t \right]$$

$$\hat{E}_t[KL[\pi_{\theta gammel}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$$

Problemet her er at denne KL-begrensningen legger ekstra arbeid på optimaliseringsprosessen, som kan føre til uønskede hendelser under trening. For å fikse dette, ble PPO implementert. I PPO flytter man denne begrensningen slik at den utføres direkte i estimatorfunksjonen. Man starter med å introdusere et sannsynlighetsforhold $r_t(\theta)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{gammel}}(a_t|s_t)}$$

Forholdet representerer endringen policyen går gjennom i løpet av en episode med trening. $\pi_{\theta_{gammel}}$ er policyen ved starten av endringen. Dette betyr at under første treningsepisode, er $r_t(\theta) = 1$.

Gitt at man har et utvalg actions og states, vil $r_t(\theta)$ være større enn 1 om en action er mer sannsynlig nå enn den var i den forrige versjonen av policyen. Det motsatte gjelder for verdier under 1. Da vil actionene være mindre sannsynlig enn i forrige versjon.

Etter å ha introdusert $r_t(\theta)$, kan vi nå se nærmere på hovedfunksjonen til PPO:

$$L_t^{CLIP}(\theta) = \hat{E}_t[min(p_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

Her følger en forklaring av de ulike variablene:

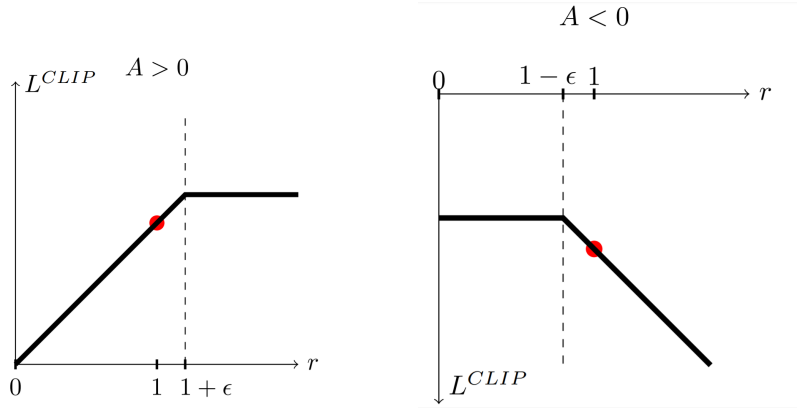
- t representerer hvilket timestep man for øyeblikket er på
- $\hat{A}_t$ er advantage function. Den regnes ut ved hjelp av to ting:
  - Discounted rewards, som er en sum av rewards agenten har fått under hvert timestep i den aktuelle episoden.
  - Value function, som prøver å gi et estimat av endelig reward basert på den nåværende staten. Resultatet av dette vil ikke være et nøyaktig svar, da dette er en prediksjon, basert på staten.

  For å finne advantage estimatet, tar man discounted rewards og subtraherer value function estimatet. I praksis er det man da gjør å sammenligne hvordan agenten faktisk presterer, med hvordan man har estimert at agenten skal prestere. Man finner ut om en foretatt action var bedre eller dårligere enn hva man på forhånd hadde antatt. Var den bedre, vil advantage estimatet være et positivt tall, og var resultatet dårligere enn antatt, vil advantage estimatet være et negativt tall.
- ϵ er et hyperparameter, ofte satt til 0.2, for å definere intervallet vi ønsker å clippe til.

L_t^{CLIP} finner minimum av to ulike uttrykk. Det første uttrykket, $r_t(\theta)\hat{A}_t$, er standard for normale policy gradients, og fører policyen mot actions som gir høy advantage.

Det andre uttrykket, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$, er en klippet versjon. $clip()$ gjør om verdier som ligger utenfor oppgitt intervall, til verdier som ligger på grensene. For eksempel om vi har et interval $[0, 1]$, vil verdien -0.2 bli til 0, og 1.2 bli til 1.

I og med at advantage estimatet, $\hat{A}_t$, kan være både positivt og negativt, vil dette ha følgende påvirkning på effekten til $min()$ operatoren:



(a) Positivt advantage estimat (b) Negativt advantage estimat

Figur 2: Visualisert objective function [1]

Disse grafene viser L^{CLIP} for positive og negative verdier av advantage estimatet. Som tidligere nevnt; et positivt advantage estimat betyr at foretatt action var bedre enn hva som var forventet på forhånd. Motsatt med negativt advantage estimat.

På grafen (a), når den foretatte action var bra, ønsker vi at den skal være *mer* sannsynlig i framtiden. Men vi ønsker ikke å oppdatere for mye, for da kan det gå mot sin hensikt. Derfor clipper vi på $1+\epsilon$, og forhindrer altfor store policy oppdateringer. Det samme skjer på (b), bare at ikke ønsker at en action skal bli for *lite* sannsynlig.

Hvis vi tar en nærmere titt på hvor grafen (b) synker, fra $1-\epsilon$ og utover, kan vi regne oss fram til at L^{CLIP} kun ender opp her dersom både r er stor, altså at en action ble gjort mer sannsynlig, og advantage estimatet er negativt, altså at det gjorde policyen dårligere. Hvis dette tilfellet oppstår, vil vi gjerne 'angre' det forrige steget vi tok. PPO gjør dette mulig, for det er i dette tilfellet $min()$ av de to uttrykkene vil returnere $r_t(\theta)\hat{A}_t$, og ikke det klippede uttrykket.

Nå har vi sett nærmere på hovedfunksjonen i PPO, men den endelige loss funksjonen som

brukes til å trene en agent er L_t^{CLIP} og to ekstra uttrykk:

$$L_t^{PPO}(\theta) = \hat{E}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right]$$

Det første uttrykket, $c_1 L_t^{VF}(\theta)$, har hovedsaklig ansvar for å oppdatere baseline-nettverket. Dette er delen av nettverket som prøver å estimere hvor bra det er å være i nåværende state. Det andre uttrykket, $c_2 S[\pi_\theta](s_t)$, sørger for at agenten utforsker under trening. c_1 og c_2 er hyperparametre som vekter uttrykkene i loss funksjonen.

Resultatet av L_t^{PPO} er en normalfordeling som beskriver en kontinuerlig verdi for hver mulige action.

2.8 Catastrophic forgetting

Noe av det mest frustrende man kan støte på når man jobber med nevrale nettverk er catastrophic forgetting [5]. Catastrophic forgetting er når nettverket ikke lenger klarer å huske gammel informasjon idét ny informasjon innhentes. Dette kan oppstå underveis i treningen og forårsakes av at nettverket overskriver, og dermed glemmer, kritiske vekter fra tidligere episoder. Hovedårsaken til at catastrophic forgetting oppstår ser ut til å være en overlapp av verdier på hidden layer-delen av det nevrale nettverket [6].

En mulig løsning på catastrophic forgetting er å senke læringsraten. Da vil nettverket tvinges til å oppdatere vektene saktere. Dermed unngår vi plutselige og for voldsomme endringer av vektene i nettverket. Det samme kan vi oppnå ved å øke batch size [7].

Catastrophic forgetting kan også oppstå som følge av overfitting. Hvis en agent er trent opp på veldig like scenarioer, og så ser den brått en state som ikke ligner på noen tidligere erfaringer, risikerer den å gjøre uforutsette actions som kan overskrive de tidligere vektene. En løsning på dette er å inkludere dropout-layers i modellen. Dropout er en teknikk hvor tilfeldig valgte nevroner blir ignorert under trening. Da må gjenstående nevroner ta seg av arbeidet, og vi oppnår en modell som er mer generell og mindre sannsynlig for å lide av overfitting.

2.9 Optimaliseringer

Denne seksjonen tar for seg ulike modifikasjoner som kan gjøres med observasjonsrommet til et environment. Hensikten med dette er å oppnå raskere trening. Raskere trening betyr her hvor fort modellen konvergerer, i forhold til antall episoder, ikke prosesseringstid.

2.9.1 Grayscale

En state med farger benytter seg av tre kanaler: rød, grønn og blå. Dette kan reduseres til en kanal ved å bruke grayscale. Effekten av dette er at det nevrale nettverket forenkles og prosesseringstiden reduseres. Det som derimot er usikkert, er hvor rask treningen blir. Det avhenger i stor grad av hvilket environment man arbeider med. Følgende utfall er mulig:

1. **Fargene i environmentet bidrar med relevant informasjon**
   Dersom vi konverterer til grayscale, mister vi relevant informasjon. Treningen vil gå tregere, eller ikke konvergere i det hele tatt. Dermed er det ikke hensiktsmessig å bruke grayscale.

2. **Fargene i environmentet bidrar ikke med relevant informasjon**
   Dersom vi konverterer til grayscale, mister vi ingen relevant informasjon. Treningen vil gå like raskt, eller noe raskere, siden nettverket er forenklet. Det er hensiktsmessig å bruke grayscale.

Generelt sett vil enkle environment som kun bruker farger av estetiske grunner falle i den sistnevnte kategorien. For vårt environment, CarRacing-v0, er det viktigste at veien er tydelig. Det er fullt mulig med grayscale.

For å konvertere fra farger til grayscale må man definere hvordan de ulike fargene skal vektlegges. For CarRacing-v0 vil det være hensiktsmessig å vektlegge grønt og rødt høyt, for å få et klart bilde av bilen i forhold til veien.

2.9.2 Redusert input

Tanken bak dette er å fjerne unødvendig informasjon fra environmentet. Dermed vil nettverket kun trenes opp på nyttig informasjon, uten å bli 'forvirret' av irrelevant informasjon. CarRacing-v0 har et informasjonspanel nederst som viser fart, ABS-sensorer, styrevinkel og gyroskop. Hvorvidt dette hjelper - eller forvirrer - er usikkert. Løsningen er derfor å dekke

over dette.

2.9.3 Frame stacking

Frame stacking går ut på å sette n antall rammer etter hverandre. Parameteren n kalles for 'phi-length'. Ved bruk av frame stacking skapes det en persepsjon av bevegelse [8], men på bekostning av at nettverket blir større. Dermed vil prosesseringstiden øke, men agenten vil til gjengeld få en slags fartsfølelse, som gjør at den forhåpentligvis vil få økt forståelse av ulike states og dermed kunne tilpasse actions bedre [9].

Et annet aspekt ved frame stacking er om frames skal settes etter hverandre sekvensielt, eller om det skal hoppes over et gitt antall frames. Det sistnevnte kalles for frame skipping. I utgangspunktet er vi hovedsakelig interessert i sekvensiell frame stacking. Kombinasjonen av action repeat og frame stacking vil likevel indirekte føre til frame skipping.

2.9.4 Action repeat

Action repeat går ut på at agenten gjentar en action n antall ganger. Dette er en teknikk som er mye brukt, da det er flere fordeler med dette [10]:

1. **Mer stabil læring**
   Å operere med en langsommere tidsskala øker gapet mellom hver action, som kan føre til mer stabil læring siden det blir lettere å rangere actions pålitelig når verdianslaget er usikkert eller inneholder støy [11].

2. **Sparer prosesseringskraft**
   Å velge en action sjeldnere kan spare en betydelig mengde av beregninger.

3. **Øker oppdagelse**
   Ved å holde seg til en action over mer enn et step kan dette hjelpe til med oppdagelse, siden diameteren av løsningsrommet er effektivt redusert.

Som en ser kan måten man velger å gjøre dette på kan spille en stor rolle for hvilket resultat man oppnår, og effektiviteten for treningen av en modell. Jo flere ganger man gjentar den samme actionen, jo mindre beregninger trenger man å gjøre. Men velger man å gjenta samme action for mange ganger kan det gå på bekostning av nøyaktigheten og prestasjonen til modellen. Samtidig som nevnt over kan og lengre sekvenser av samme action føre til

bedre utforskingen av environmentet, da modellen kan komme bort i tilfeller den ikke ville gjort utenom. Men på samme måte her kan en for høy verdi føre til at bilen vil gjenta en action for mange ganger, og vil på den måte redusere utforskningen.

Vi kommer til å se på statiske verdier for action repeat, og undersøke hvilken verdi som er optimal for CarRacing-v0. Alternativet er såkalt dynamisk tilpasning. Da trenes agenten opp til å endre action repeat underveis. Et eksempel på en slik tilnærming har blitt gjort i FiGAR [12].

For CarRacing-v0 vil antageligvis en høyere action repeat føre til raskere trening. Det er likevel noen ulemper. Jo høyere action repeat, jo mer vil bilen vingle. Dette er fordi agenten er 'låst' til en action i n antall frames. Dermed vil det som i utgangspunktet kun er en liten korreksjon vare lenger enn tiltenkt, og den kan dermed virke mot sin hensikt. Si at bilen holder på å kjøre av veien. Da vil den korrigere ved å svinge inn på veien igjen. Dersom action repeat er for høy, vil den svinge for lenge, og mot den andre siden av veien, hvor da den igjen må korrigere. Dette fører til at bilen kjører i sikk-sakk i steden for å kjøre en rett linje.

3 Relevant arbeid

Denne delen tar for seg og presenterer arbeid som tidligere er gjort innenfor de områdene vi ønsker å utforske videre. Her presenterer vi først environmentet vi har valgt å ta for oss, hva som har blitt gjort av andre og hva de har konkludert med. Det vi lærer og finner ut av her vil bli brukt videre til å bygge vårt arbeid på, og som erfaringer til vår egen forsking.

3.1 CarRacing-v0

Carracing-v0 [3] er et racing environment fra OpenAI. Det er et 2D environment med en synsvinkel fra toppen og ned, hvor hver frame er en state bestående av 96x96 piksler. For hver kjøring blir det generert en tilfeldig bane med et gitt antall felter. Miljøet har et poengsystem som gir en belønning på -0.1 for hver frame og $+1000/N$ for hvert felt som er besøkt, hvor N er totalt antall felter besøkt. Environmentet er ansett som løst når en agent gjennomsnittlig oppnår en poengsum på 900+ poeng. En episode er fullført når en har besøkt alle feltene, eller den kjører utenfor banen og oppnår en poengsum på -100.

Environment er fra utvikler satt opp med et kontinuerlig action space som inneholder tre verdier(styring, gass og brems), men sier at også et diskre action space kan være et fornuftig valg. Dette action spacet med tillhørende verdier er illustrert i tabell 1 under.

Action value	Minimum	Maksimum
Styrevinkel	-1.0	1.0
Gass	0.0	1.0
Brems	0.0	1.0

Tabell 1: Kontinuerlig action space for Carracing-v0

3.2 'Applying a Deep Q Network for OpenAI's Car Racing Game'

[13] Selvom denne artikkelen forsøkte å løse Carracing-v0 med bruk av DQN så var det mye interessant vi kunne bruke i vår egen problemstilling ettersom den omhandlet mye rundt enviroment wrapperen. Konsepter og modifikasjoner vi har testet som er inspirert av denne artikkelen er blant annet gray scaling, som reduserer inputen til nettverket for å dermed forenkle treningen. Det er også i dette arbeidet vi ble introdusert for catastrophic forgetting.

Artikkelen konkluderer med at 'While I have had experience solving other OpenAI environments, this one seems to be the most challenging one that I have attempted.' Modellen nådde aldri enviormentets mål på å konsekvent oppnå 900 poeng på flere kontinuerlige episoder. På tross av dette er det mye lærdom om hvor mye forskjellige modifikasjoner påvirker resultatet og det er dette vi har tatt med oss videre i arbeidet vårt

3.3 'PPO Implementation for OpenAI Environments'

[14] Dette arbeidet var nyttig for oss ettersom det var brukt PPO i et forsøk på å løse CarRacing-v0. Det ble også brukt PPO for å løse CartPole-v1, et simplere environment.

Modifikasjonene som ble gjort for å forenkle treningen i denne artikkelen var diskretisering av action-space. Som nevnt, så har carracing-v0 et kontinuerlig action-space som kan diskretiseres til et ønsket antall verdier og dermed et antall forskjellige actions. Dette er en interessant måte å forenkle treningen av en PPO men vi valgte å ikke gå dypere inn på dette, da vi hovedsakelig ser på observasjonsrommet.

Dessverre konkluderte også denne rapporten med at enviromentets mål på å konsekvent oppnå 900 poeng på flere kontinuerlige episoder ikke ble nådd. Dette skyldes "vanishing gradient issue"og en mangel på ressurser. Det ble utført 1500 episoder i treningen men med ytterligere tuningen av hyperparameterene med mer så kunne oppgaven blitt løst.

Denne artikkelen var hjelpsom ettersom den hjalp forståelsen vår for PPO, siden det først ble implementert i et kjent og simpelt environment før det ble implementert til environmentet vi skulle jobbe med.

3.4 'Solving Car Racing with Proximal Policy Optimisation'

[15] Artikkelen ble skrevet på grunn av mangelen på informasjon rundt CarRacing-v0, og går inn på flere modifikasjoner man kan gjøre for å forenkle treningen. Dessuten er problemet løst med PPO, og har derfor vært til stor hjelp under prosjektet.

Det ble blant annet utført diskretisering av action space og reduksjon av input til nettverket ved å fjerne informasjonspanelet på bunnen av skjermbildet samt bruk av grayscale. Det ble også implementert en image stack med en størrelse på fire for å gi modellen en følelse av bevegelsen til bilen. Dette gjør antallet input channels høyere, men ble ansett som nødvendig.

Med et oppsett som i artikkelen kan man allerede etter 30 minutter se store forbedringer. Problemet ble løst med PPO og disse modifikasjonene danner mye av grunnlaget for hvordan vi har valgt å gå fram i dette prosjektet.

3.5 'Car Racing with PyTorch'

[2] Vi valgte å ta utgangspunkt i dette Github repoet. Her fant vi mye kode som vi kunne bruke som grunnlag i vårt eget arbeid. Grunnen til dette var at modifikasjoner som frame stack, action repeat og grayscale var implementert med PyTorch og dessuten kunne dette prosjektet vise til gode resultater.

Selv om det ble nødvendig å skrive om enkelte deler av koden og endre noe av strukturen, har vi hentet mye inspirasjon ettersom problemstillingen vår ikke gjorde det nødvendig å skrive hele koden fra bunnen av, men heller justere parametere og struktur.

4 Metode

4.1 Teknologivalg

4.1.1 Programvare

Koden [16] er skrevet i Python 3. Det er brukt PyTorch, ettersom det er rammeverket som har blitt undervist i emnet. Environmentet er hentet fra OpenAI Gym, som er satt opp for reinforcement learning.

4.1.2 Maskinvare

For å løse problemstillingen trengte vi mye prosesseringskraft. Det ble derfor kjørt treninger parallelt, på følgende maskiner:

1. CPU med 32 kjerner og 64GB RAM, NTNU
2. GPU med 25 GB RAM, Google Colab [17]
3. Egne maskiner

Ettersom vi trente på ulike maskiner, varierte prosesseringstiden betraktelig. Det er derfor ikke et aspekt vi ser på når vi snakker om raskere trening. Da ser vi på antall episoder.

4.2 Hyperparametere

Parameter	Verdi
Læringsrate	0.001
Max antall episoder	3000
Bufferstørrelse	6000
Batchstørrelse	128
Max antall steg	1500
Epsilon	0.1
Gamma	0.99
Epoker	8

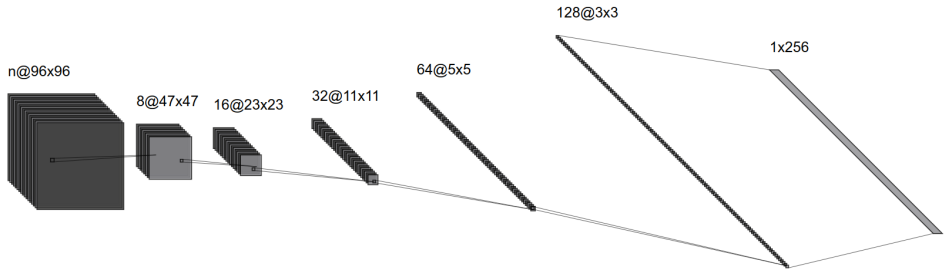
Tabell 2: Hyperparametere

Merk at bufferstørrelsen ble justert for hver optimalisering slik at nettverket oppdaterte vektene sine etter 10-15 episoder.

4.3 Nettverket

Det nevrale nettverket består først av følgende konvolusjonslag:

1. Convolutional 4x4 (n, 8) + ReLU
2. Convolutional 3x3 (8, 16) + ReLU
3. Convolutional 3x3 (16, 32) + ReLU
4. Convolutional 3x3 (32, 64) + ReLU
5. Convolutional 3x3 (64, 128) + ReLU
6. Convolutional 3x3 (128, 256) + ReLU



Figur 3: Konvolusjonslagene i det nevrale nettverket

Merk at n er et parameter i første laget. Denne er avhengig av hvilke optmialiseringer som er implementert. Den regnes ut ved:

$$n = \text{Lengden av frame stacken} \times \text{Antall fargekanaler}$$

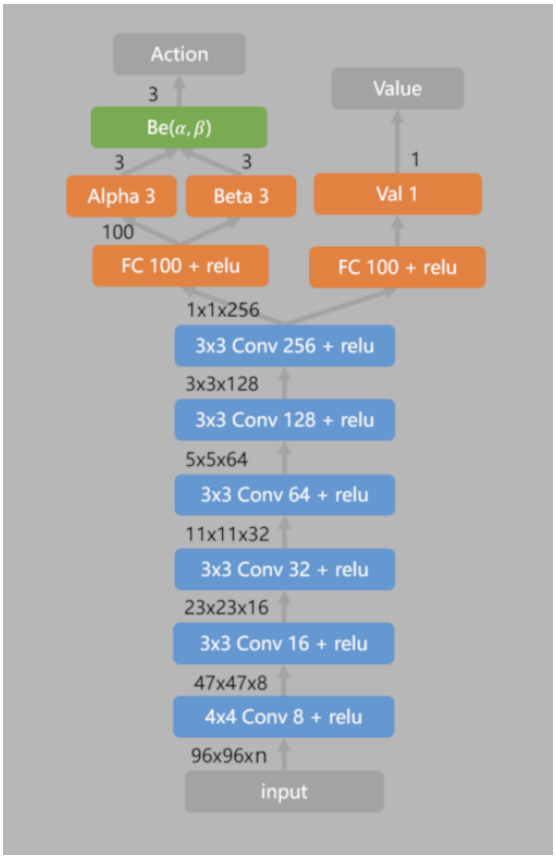
Beta-distrubisjonen er gitt ved konvolusjonslagene, deretter:

1. FC 100 + ReLU
2. Beta 3
3. Alpha 3
4. Be(α, β)

Value er gitt ved konvolusjonslagene, deretter:

1. FC 100 + ReLU
2. Val 1

Figuren nedenfor viser en illustrasjon av oppbyggingen til nettverket.



Figur 4: Det nevrale nettverket [2]

4.4 Trening

Koden er strukturert på en slik måte at det er enkelt å legge til eller fjerne ulike optimaliseringer. Alle hyperparametrene og optimaliseringer kan stilles i filen *parameters.py*.

For å ha noe å sammenligne med kjørte vi først en modell uten noen optimaliseringer, som vi har valgt å kalle for baseline. Denne modellen bruker vi senere for å se hvorvidt de ulike optimaliseringene har noen effekt. I enkelte tilfeller har vi satt baseline til å bruke action repeat lik 4, slik at vi også får fram effekten av de ulike optimaliseringene i en litt mer sammensatt situasjon.

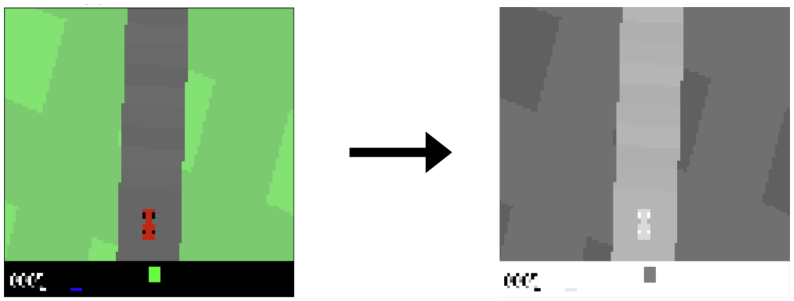
Det er mulighet for å sette seed for hver trening. Alle grafene som er presentert i denne rapporten er fra samme seed. Dermed er alle de ulike optimaliseringene kjørt på de samme tilfeldig genererte banene, og vi kan se effekten av optimaliseringen alene.

For treningsperioden er den satt til å vare til en oppnår en gjennomsnittlig reward på 900, som er ansett som når en har løst environmentet, eller til den når 3000 episoder. Dette er fordi en er nødt til å sette en begrensning på prosjektet siden det er begrenset med prosesseringskraft og tid.

4.5 Optimaliseringer

Denne seksjonen tar for seg hvordan de ulike optimaliseringene som ble implementert for det relevante environmentet, CarRacing-v0.

4.5.1 Grayscale



Figur 5: Konvertering fra farger til grayscale

Staten er i utgangspunktet gitt i farger, på formen (96, 96, 3). Målet er å konvertere dette til grayscale, på formen (96, 96). Da må det bestemmes hvor mye hver farge skal vektlegges. Grønt ble vektlagt høyest, slik at veien kom tydelig fram. Rødt ble også vektlagt høyt for å få bilen tydelig fram. Fordelingen ser slik ut:

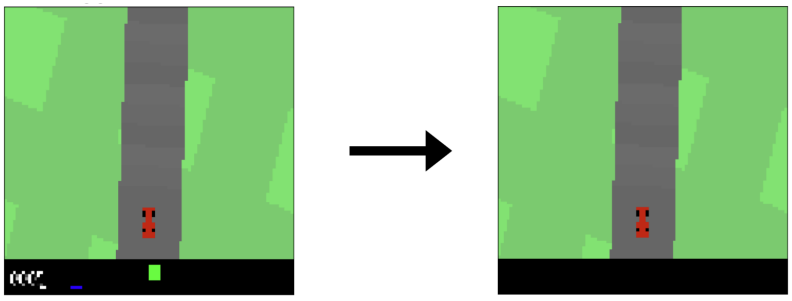
1. Rødt: 29.9%
2. Grønt: 58.7%
3. Blått: 11.4%

For å oppnå dette tok vi kryssproduktet av staten med en 3×1 vektor hvor hvert element representerte hvor mye hver farge skulle vektlegges. Matematisk ser dette slik ut:

$$\text{State} \cdot \begin{bmatrix} 0.299 \\ 0.587 \\ 0.114 \end{bmatrix}$$

Da ender vi opp med en matrise på formen (96, 96). Verdiene i denne matrisen er $\in [0, 255]$. Deretter normaliseres dette ved å dele alle verdiene i matrisen på 128 og trekke fra 1. Da er verdiene i matrisen $\in [-1, 1]$.

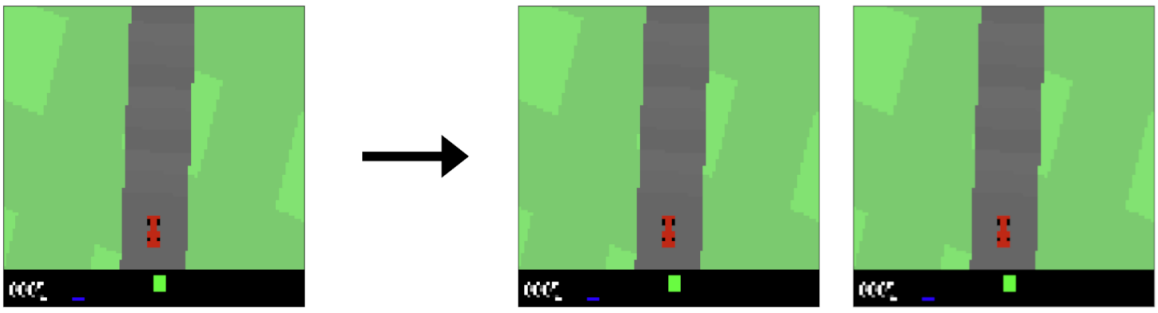
4.5.2 Redusert input



Figur 6: Redusert input

Dekker over informasjonspanelet ved å sette de 10 nederste pikselene fra staten til 0. Da vil staten bevare samme størrelse på (96, 96, 3), men med redusert informasjonsmengde.

4.5.3 Frame stacking



Figur 7: Frame stacking med $n = 2$

For å implementere frame stacking innføres en stack. Stacken inneholder n frames. Etter agenten har utført en action, fjernes den eldste framen fra stacken, og den nye framen legges til. Merk at dersom action repeat er satt, vil det også i praksis brukes frame skipping. Agenten vil utføre k antall actions, deretter legge til en ny frame i stacken. Et oppsett med action repeat satt til k vil dermed også implisitt hoppe over $k - 1$ frames før en ny frame legges til i stacken.

4.5.4 Action repeat

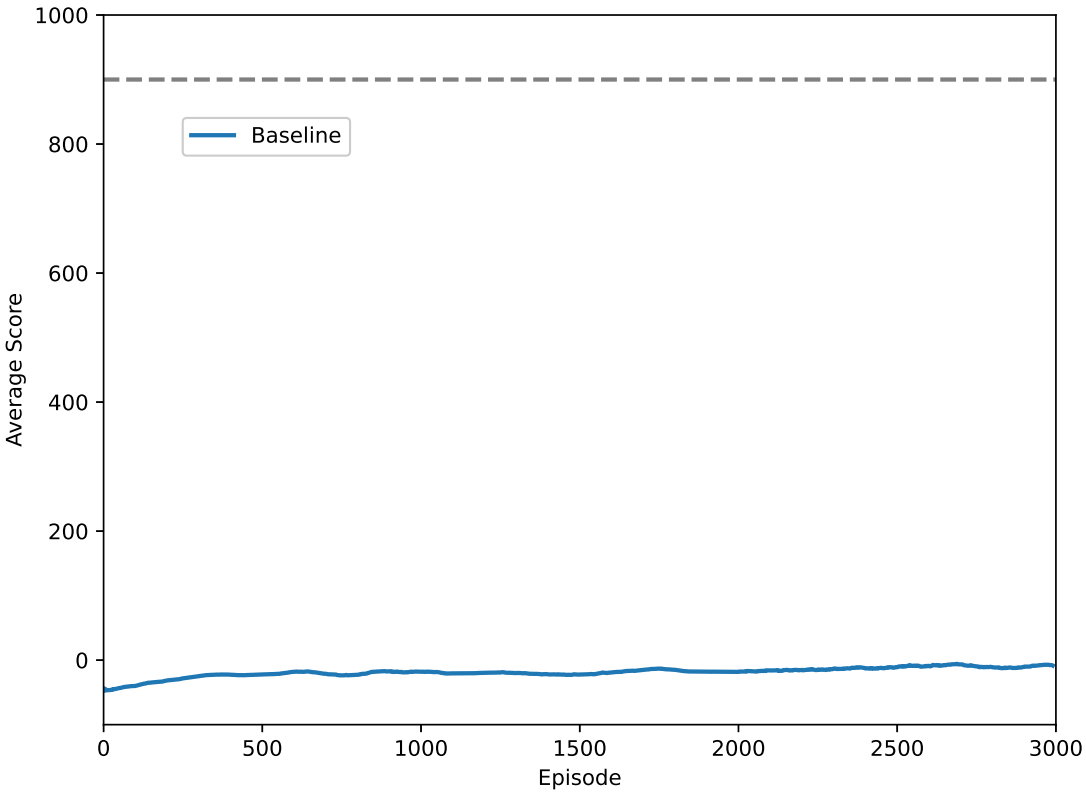
For å implementere action repeat legges det til en simpel for-løkke. For hver iterasjon av for-løkken kalles step-funksjonen til environmentet. Utenfor loopen returneres staten fra den siste iterasjonen. Som nevnt ovenfor vil dette i kombinasjon med frame stacken føre til frame skipping.

5 Resultat

Denne seksjonen tar for seg resultater fra treningene. Grafene som er presentert er fra samme seed. Den første aksen er episoder, mens den andre aksen er gjennomsnittlig reward for de siste 100 episodene. Den grå stiplede linjen er gjennomsnittlig reward lik 900, som er et mål på om agenten har løst environmentet. Da stopper treningen. Dersom agenten ikke når gjennomsnittlig reward på 900, vil den trene i 3000 episoder.

5.1 Baseline

For å se om optimaliseringene har noen effekt, har vi en baseline som vi sammenligner mot. Denne implementerer ingen optimaliseringer.

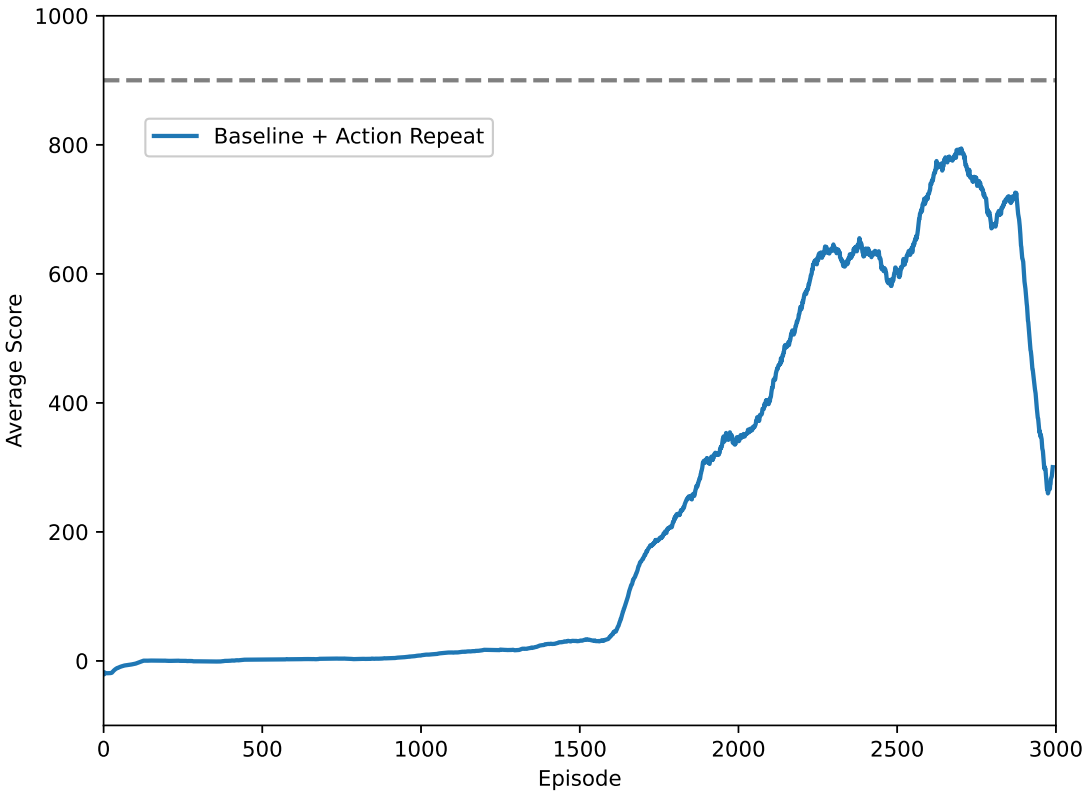


Figur 8: Baseline

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	1
Action repeat	1

Tabell 3: Optimaliseringsparametere for baseline

Vi ønsker også å se på effekten av hver optimalisering i en litt mer sammensatt setting. Derfor velger vi å også sammenligne de ulike optimaliseringene med action repeat lik fire.

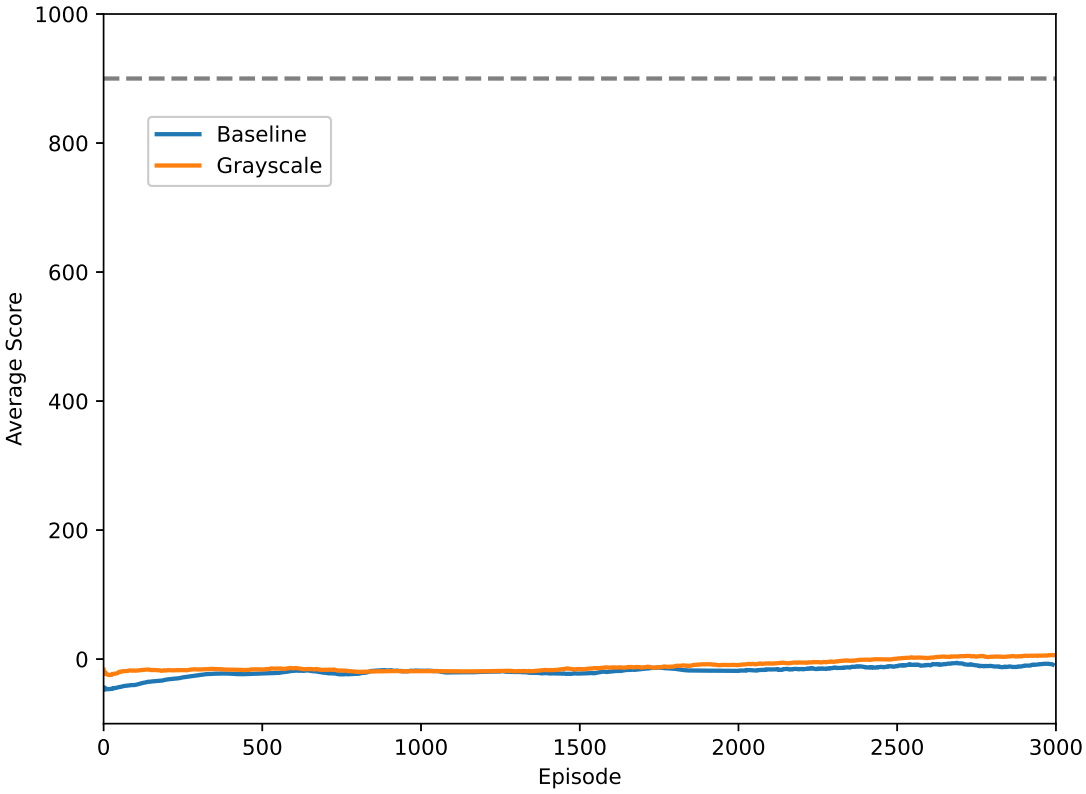


Figur 9: Baseline m/ action repeat

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	1
Action repeat	4

Tabell 4: Optimaliseringsparametere for baseline m/ action repeat

5.2 Grayscale

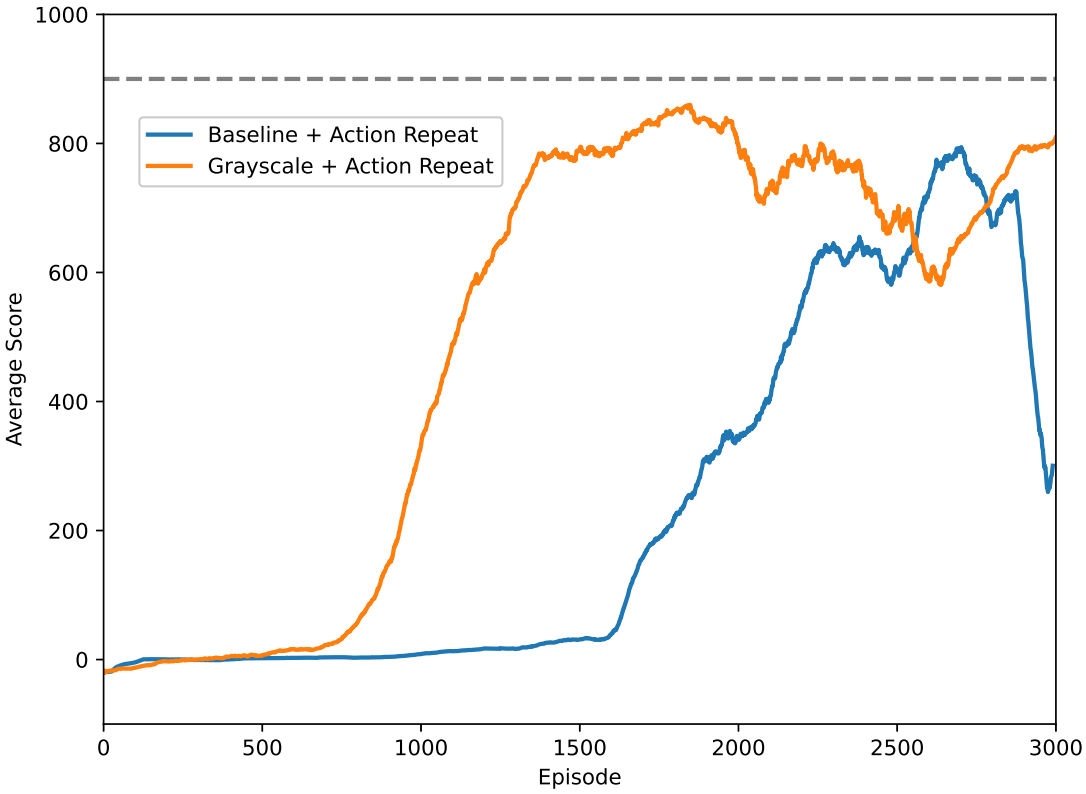


Figur 10: Grayscale vs. baseline

Parameter	Verdi
Redusert input	False
Grayscale	True
Frame stack	1
Action repeat	1

Tabell 5: Optimaliseringsparametere for grayscale

Ser på effekten av grayscale der action repeat også er implementert:

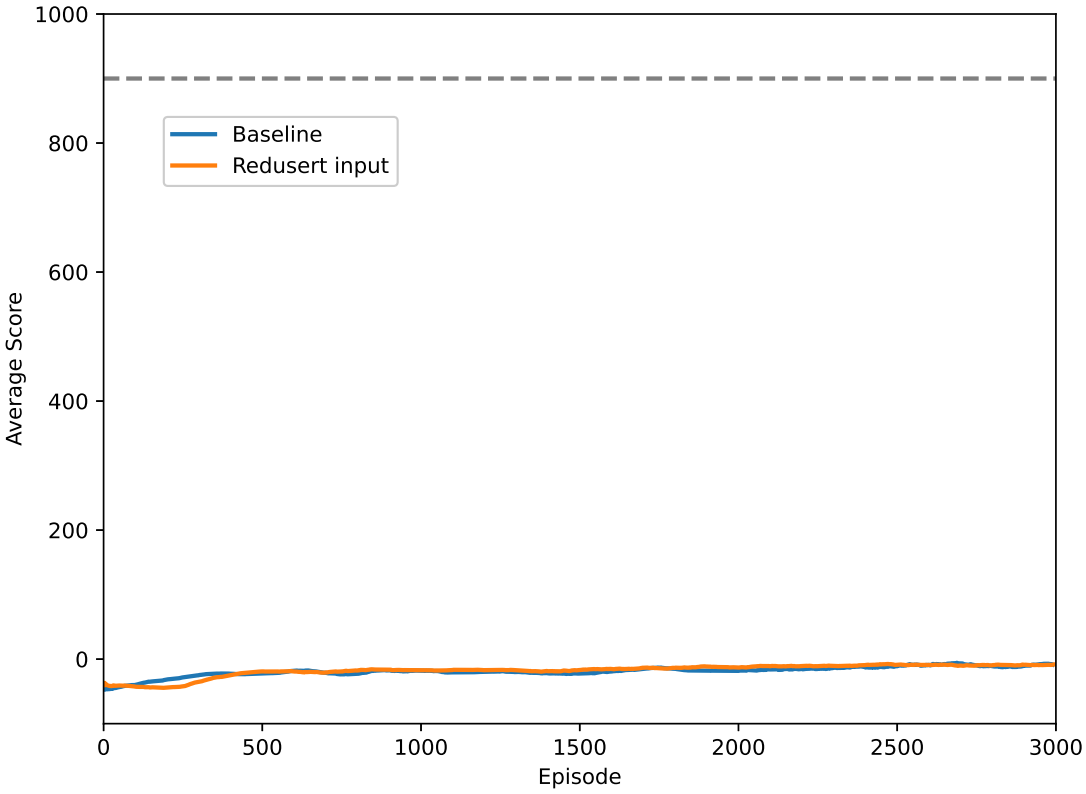


Figur 11: Grayscale vs. baseline (m/ action repeat)

Parameter	Verdi
Redusert input	False
Grayscale	True
Frame stack	1
Action repeat	4

Tabell 6: Optimaliseringsparametere for grayscale m/ action repeat

5.3 Redusert input

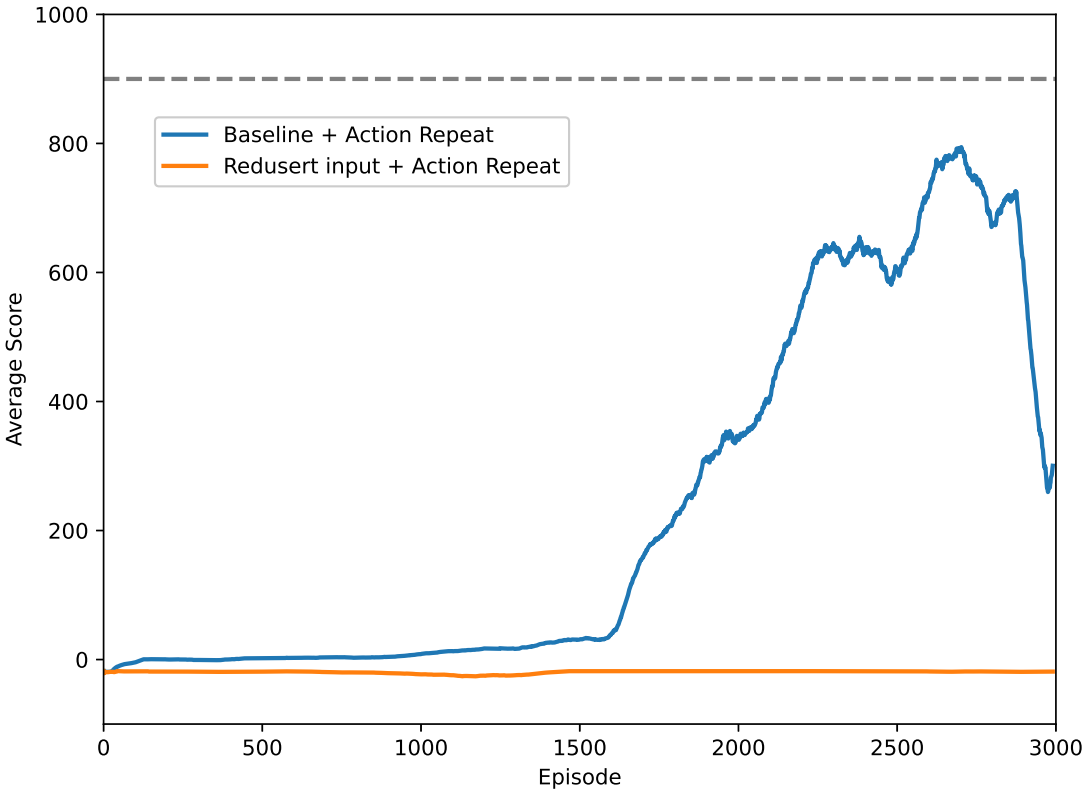


Figur 12: Redusert input vs. baseline

Parameter	Verdi
Redusert input	True
Grayscale	False
Frame stack	1
Action repeat	1

Tabell 7: Optimaliseringsparametere for redusert input

Ser på effekten av redusert input der action repeat også er implementert:

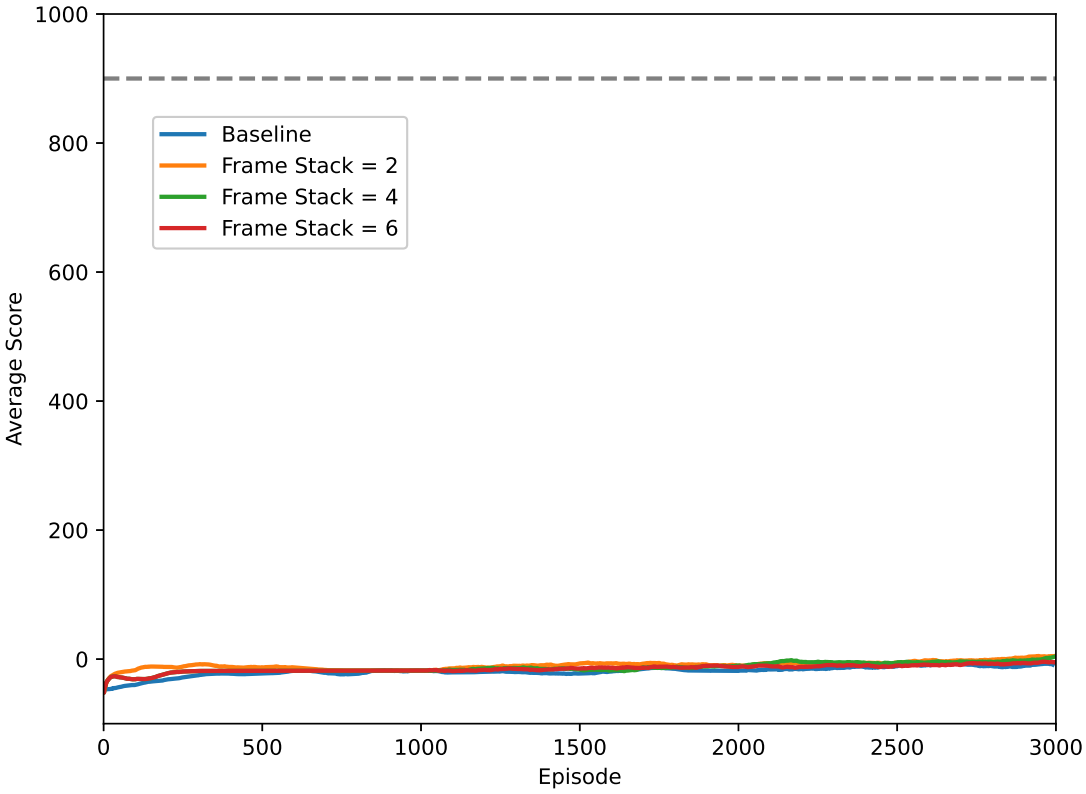


Figur 13: Redusert input vs. baseline (m/ action repeat)

Parameter	Verdi
Redusert input	True
Grayscale	False
Frame stack	1
Action repeat	4

Tabell 8: Optimaliseringsparametere for redusert input m/ action repeat

5.4 Frame stacking

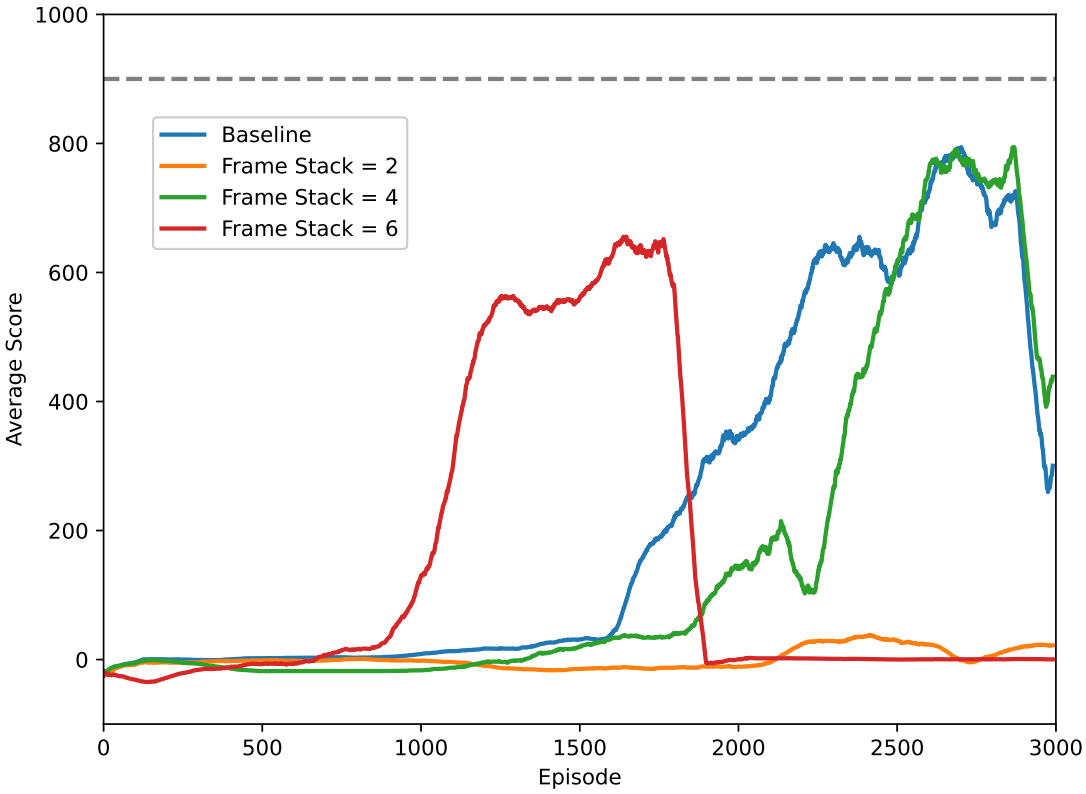


Figur 14: Frame stacking vs. baseline

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	2, 4, 6
Action repeat	1

Tabell 9: Optimaliseringsparametere for frame stacking

Ser på effekten av frame stacking der action repeat også er implementert:

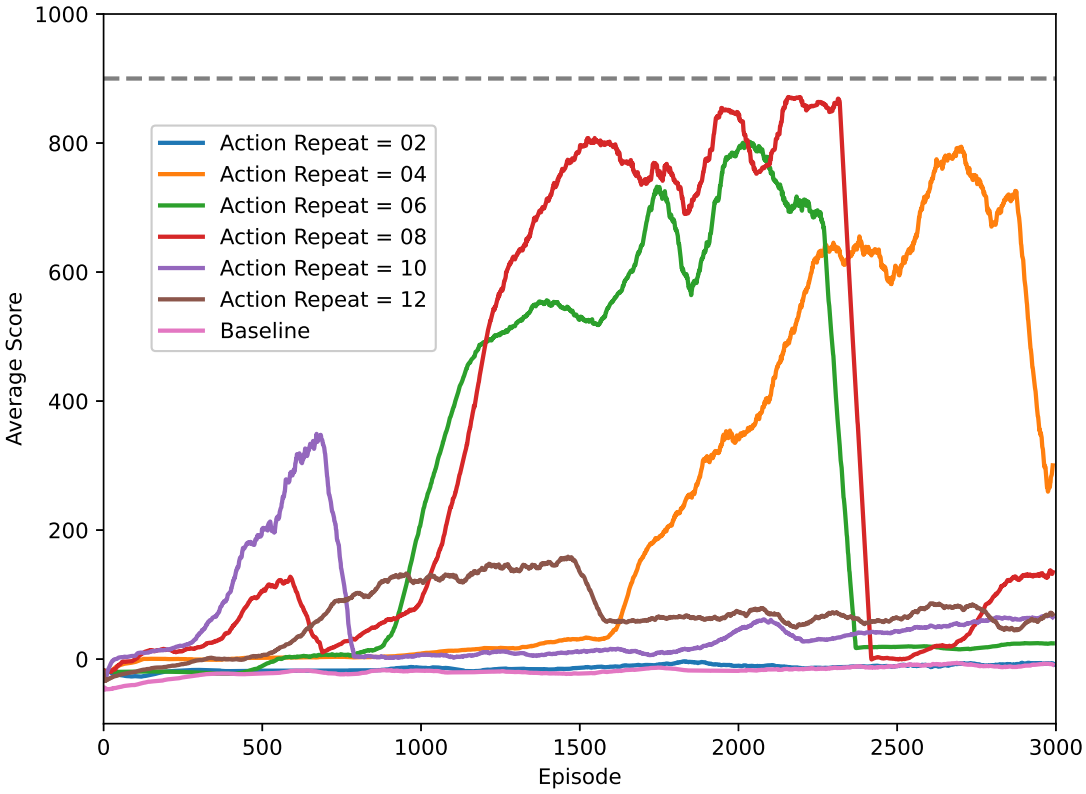


Figur 15: Frame stack vs. baseline (m/ action repeat)

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	2, 4, 6
Action repeat	4

Tabell 10: Optimaliseringsparametere for frame stacking m/ action repeat

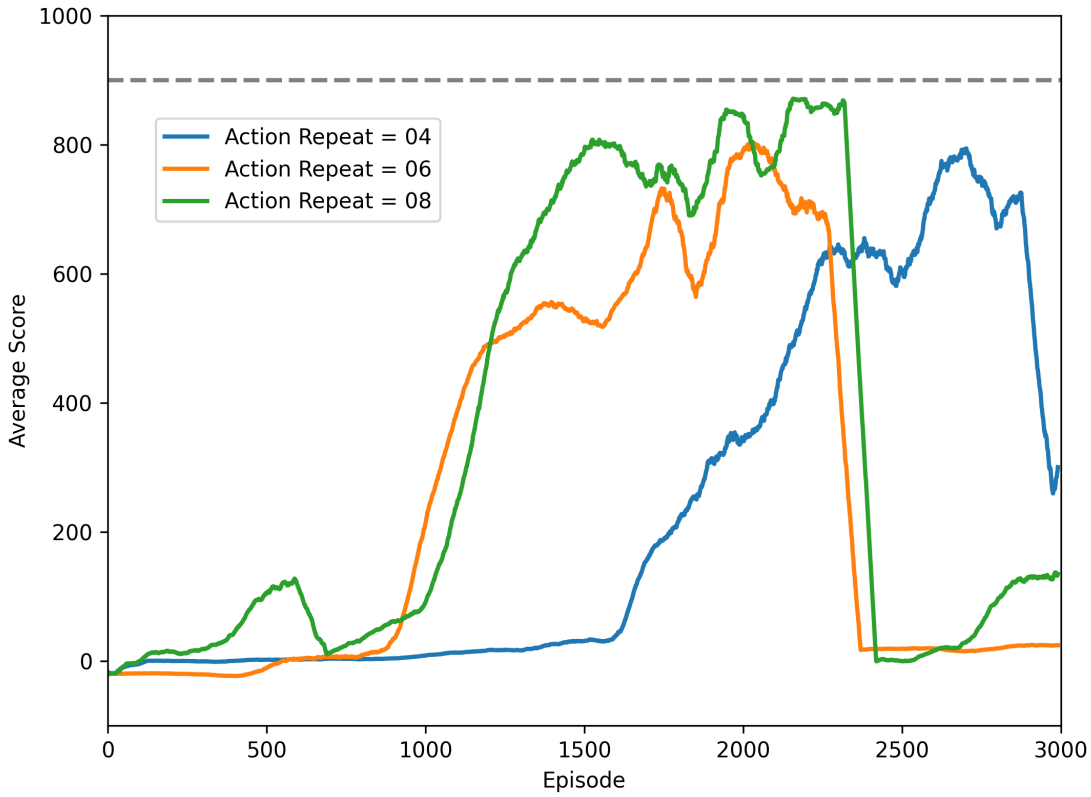
5.5 Action repeat



Figur 16: Action repeat vs. baseline

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	1
Action repeat	2, 4, 6, 8, 10, 12

Tabell 11: Optimaliseringsparametere for action repeat

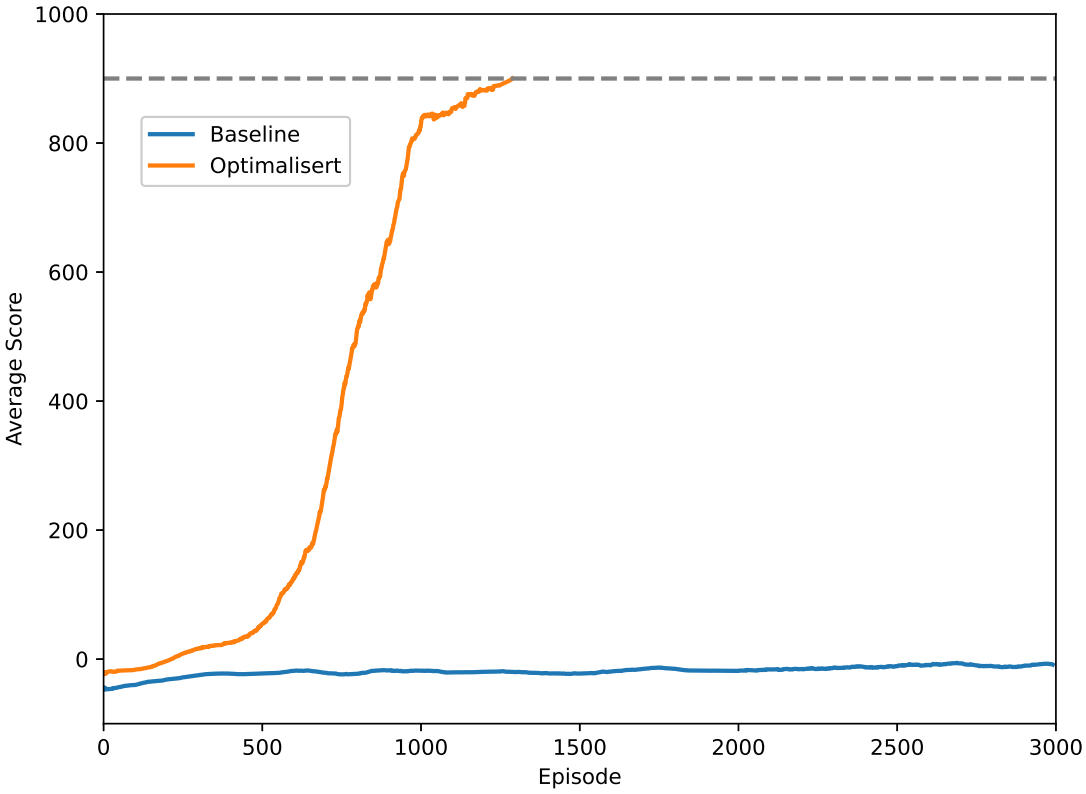


Figur 17: Action repeat - optimale verdier

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	1
Action repeat	4, 6, 8

Tabell 12: Optimaliseringsparametere for action repeat

5.6 Optimalisert løsning

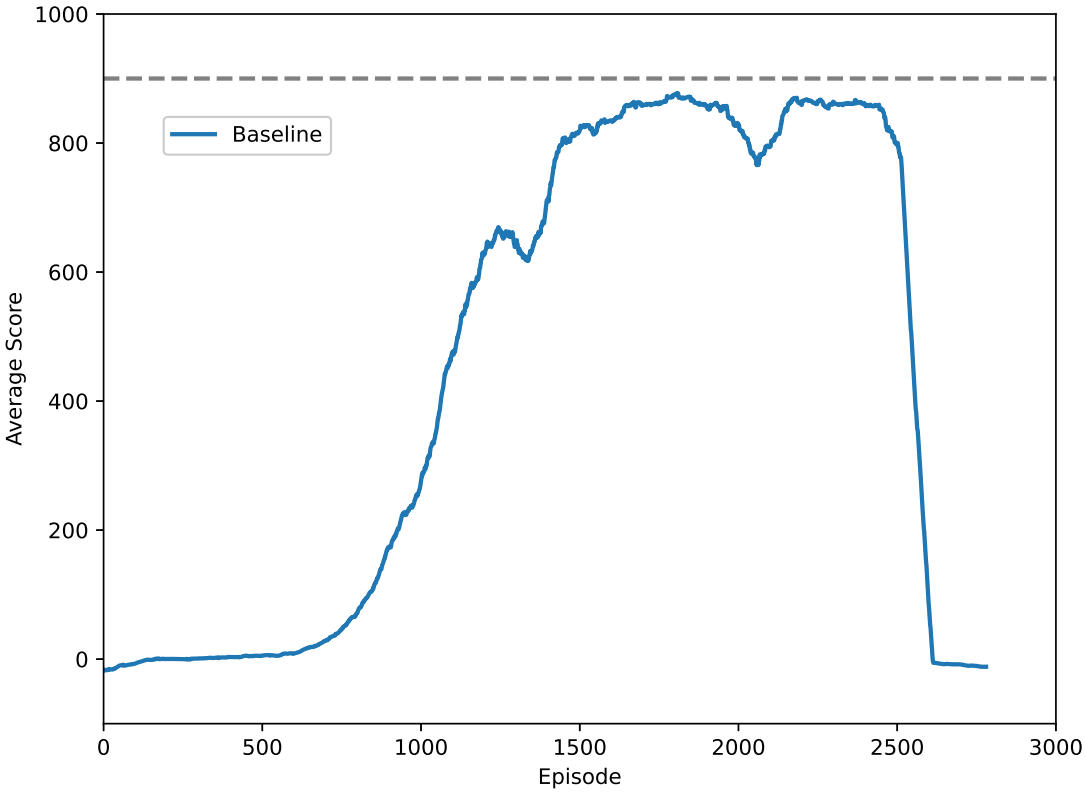


Figur 18: Optimalisert vs. baseline

Parameter	Verdi
Redusert input	False
Grayscale	True
Frame stack	4
Action repeat	8

Tabell 13: Optimaliseringsparametere for den optimaliserte løsningen

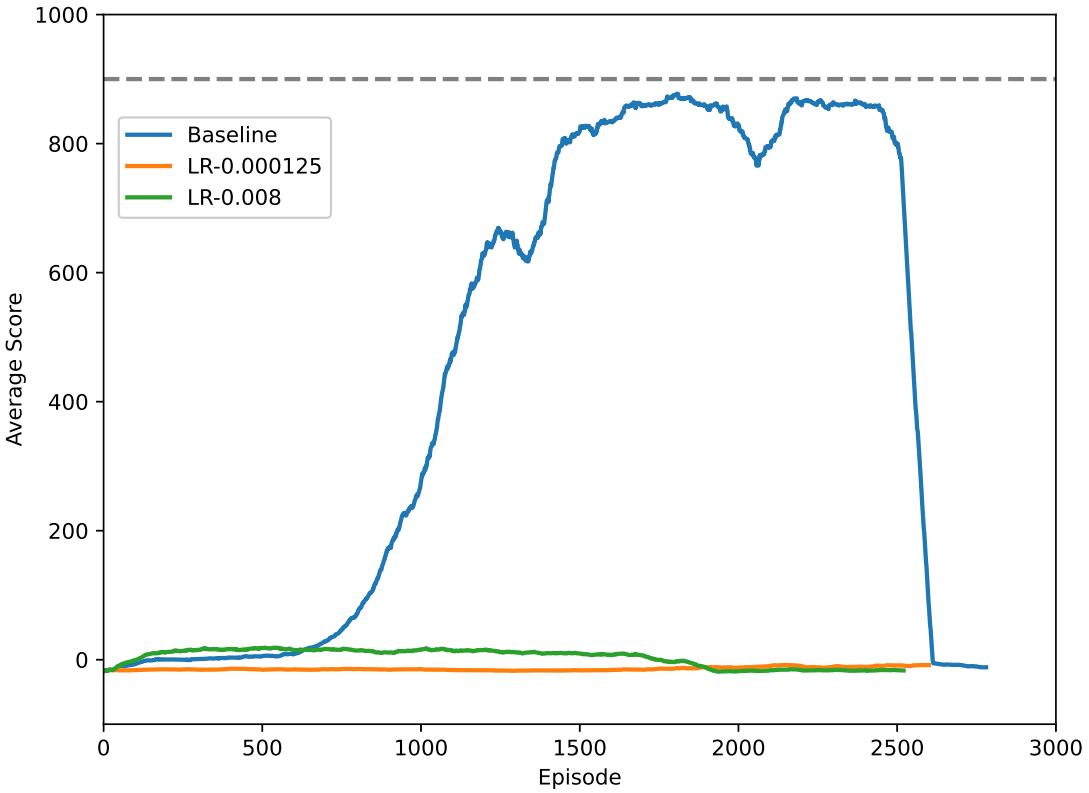
5.7 Catastrophic forgetting



Figur 19: Catastrophic forgetting

Parameter	Verdi
Redusert input	False
Grayscale	False
Frame stack	4
Action repeat	8
Learning rate	0.001

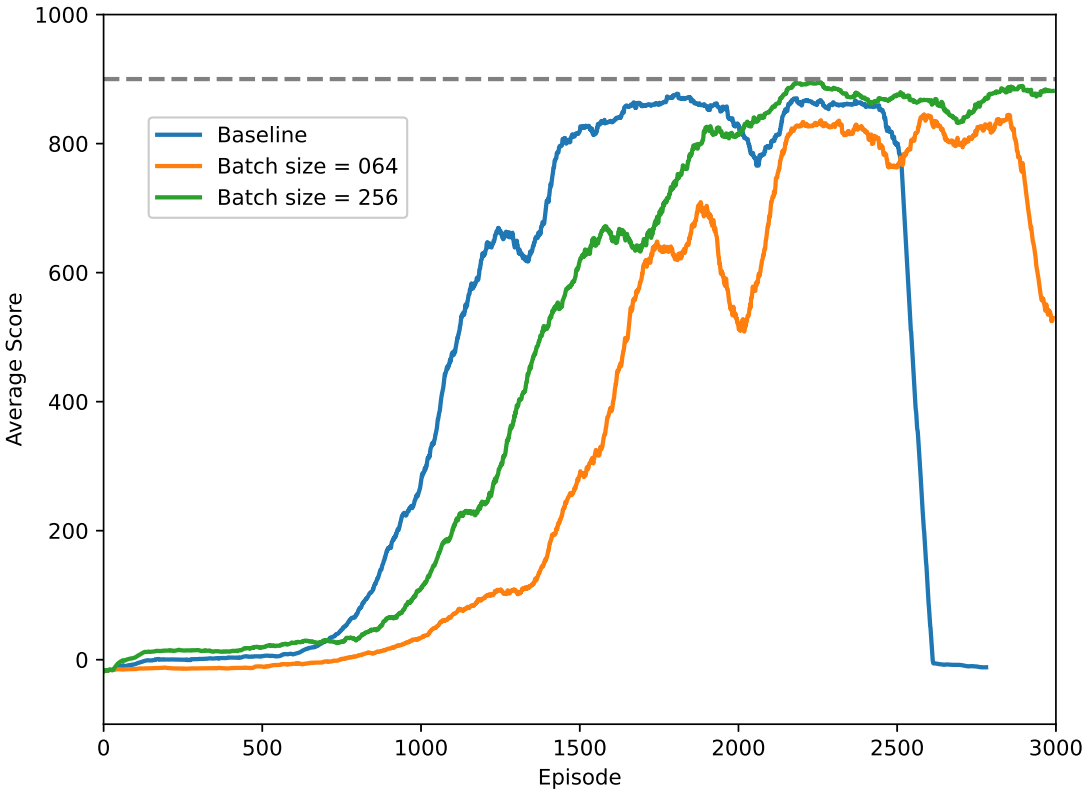
Tabell 14: Optimaliseringsparametere for catastrophic forgetting



Figur 20: Forsøk på korreksjon av catastrophic forgetting v/ modifisert læringsrate

Parameter	Verdi
Redusert input	False
Grayscale	True
Frame stack	4
Action repeat	8
Learning rate	0.008, 0.000125

Tabell 15: Optimaliseringsparametere for korreksjon av catastrophic forgetting



Figur 21: Forsøk på korreksjon av catastrophic forgetting v/ modifisert batch size

Parameter	Verdi
Redusert input	False
Grayscale	True
Frame stack	4
Action repeat	8
Batch size	64, 128, 256

Tabell 16: Optimaliseringsparametere for korreksjon av catastrophic forgetting

6 Diskusjon

Resultatene for de fleste optimaliseringene samsvarer med forventningene vi hadde. Vi skal her gå systematisk gjennom de forskjellige å se på hvorfor er det er slik.

6.1 Grayscale

Grayscale førte ikke til en forbedring i forhold til baseline. For den sammensatte situasjonen derimot, så vi en forbedring i både average score og treningstid. Dette skyldes antageligvis at nettverket ble enklere. Dermed kan vi konkludere med at fargene i CarRacing-v0 har lite å si. Den viktigste informasjonen i forhold til trening er hvor bilen er i forhold til veien. Dette oppnådde vi med grayscale ved å vektlegge fargene grønn og rød. Da kunne nettverket enkelt identifisere gress, vei og bil.

Grayscale er dermed en hensiktsmessig optimalisering å benytte for CarRacing-v0, da fargene ikke bidrar med relevant informasjon som kan brukes til trening.

6.2 Redusert input

Redusert input førte ikke til forbedring i forhold til baseline. I den sammensatte situasjonen ser vi en klar forskjell, nemlig at treningen går betraktelig dårligere med redusert input. Dette betyr at informasjonen nederst ikke er forvirrende slik vi postulerte i teoridelen. Det virker som modellen bruker denne informasjonen til å trene.

Det er dermed ikke hensiktsmessig å redusere inputen ved å dekke over informasjonspanelet. Modellen er avhengig av denne informasjonen.

6.3 Frame stacking

Frame stack ga oss ingen bemerkelsesverdig forbedring i den opprinnelige baselinen, men i kombinasjon med action repeat kan vi observere mer interessante resultater.

En frame stack med størrelse 2 ser ut til å forverre resultatet betraktelig. Årsaken kan være at det er forvirrende å ta en avgjørelse basert på kun to forskjellige frames. 2 frames gjør det vanskelig å avgjøre bevegelse, fart og retning, og vi ser på grafen at det ikke er særlig forbedring over vanlig baseline uten action repeat. Det er litt overraskende at frame stack 2 skal gjøre det såpass mye dårligere enn frame stack 1, men vi har konkludert med at det skyldes at agenten forvirres av lite tydelig informasjon.

Det er helt klart at valget står mellom frame stack med størrelse 4 eller 6. Ved en størrelse på 6 kan vi observere at det blir oppnådd et topppunkt tidligere, men catastrophic forgetting inntreffer også tidligere. Dette kan skyldes at det nå vil sendes mer informasjon som input til nettverket.

Ved en størrelse på 4 viser resultatene at det oppnås et høyere makspunkt og treningen er tilsynelatende mer stabil over intervallet av episoder vi har definert som område. Vi har derfor valgt å ta frame stack størrelse 4 med i den optimale modellen vår.

6.4 Action repeat

Vi ser tydelig at action repeat er en helt essensiell optimalisering som må til for at modellen skal prestere. Vi forventet at dette skulle spille en viktig rolle på hvordan modellen presterte, men ikke at det skulle være så avgjørende som det har vist seg. I andre tester vi har kjørt for andre optimaliseringer, hvor det ikke har vært action repeat, har det vist seg at vi ikke får ønskede resultater. Man må påpeke at, som nevnt i metode-kapitellet, vi har satt en begrensning for læring til 3000 episoder, så hva som hadde vært tilfelle dersom vi hadde kjørt dette over en lenger tidsperiode tar vi ikke til betraktning her.

Det virker derfor som at for å få nyttinge resultater fra forsøk med de andre optimaliseringene, er vi avhengige av action repeat. Vi fant derfor ut av at vår opprinnelige baseline mest sannsynlig ikke ville gi et godt bilde av hvilken effekt forskjellige verdier for de andre optimaliseringene ville ha. Vi opprettet da enda en baseline med action repeat satt til 4.

Når vi begynner å studere nærmere på hvordan modellen presterer etterhvert som vi prøver ut forskjellige verdier for action repeat, ser vi at det er enkelte verdier som gjør det bedre enn andre. Med små verdier ser vi at disse er for lave til å utgjøre noen forskjell. Som man ser på grafen i resultater ligger linjen for Action repeat = 2 nesten likt med baselinen. Når vi da ser på neste steg som er Action repeat = 4, ser vi en tydelig forbedring. Modellen ser nå ut til å prestere bedre og oppnår etterhvert en relativt høy average score. Denne scoren ser ut til å forbedre seg, helt til vi opplever noe som skal vise seg å være et gjengående problem. Nemlig at en plutselig dropp i average score, og da et stort fall. Vi kommer mer tilbake til dette og ser nærmere på det i delen Catastrophic forgetting.

Når man ser på resultatene fra Action repeat = 6 og 8 ser vi at disse også presterer bra. Action repeat = 6 oppnår en ganske lik average score som Action repeat = 4, men vi ser

at disse lærer fortere og kommer opp på dette nivået en del episoder tidligere. Som snakket om i teoridelen til action repeat var to av fordelene med dette en mer stabil læring og økt oppdagelse, noe vi ser en effekt av her. Siden modellen nå bare trenger å gjøre et valg av ny action hver fjerde frame, har den bedre tid på seg til å vurdere om actionen som ble valg var et bra valg eller ikke. Siden modellen gjentar samme action fire ganger blir oppdagelsen også større nå, som vi ser mest sannsynlig også har en påvirkning på læringen. Action repeat satt til 8 ser vi at lærer omtrent like fort som action repeat = 6 gjorde, men oppnår etterhvert en høyere average score. Her ser vi at den oppnår en average score ganske nær 900 allerede med action repeat som eneste optimalisering, men får etter hver igjen et stort fall.

Etter hvert som vi utforsker videre med verdiene Action repeat = 10 og 12 observerer vi at vi mister mye av effekten igjen. Med Action repeat lik 10 ser man at modellen begynner å lære tidligere, men kommer raskere til et stopp hvor den glemmer og kommer seg aldri igjen etter det. Action repeat på 12 flater derimot ut mye tidligere og oppnår aldri den læringskurven som vi har sett på tidligere grafer. Grunnen til dette er mest trolig at en nå har nådd en så høy verdi for Action repeat at hvert valg av en action har veldig stor betydning. Gjøres det et par action etter hverandre som ikke var de optimale valgene har det store konsekvenser for agenten. Vi ser på videoene fra kjøringer her at bilen er vinglete når den kjører og når den bremser gjør den det alt for lenge slik at den mister all hastighet. Hver action den tar har for stor effekt.

Vi ser fort at den optimale verdien ligger en plass mellom en Action repeat = 4 og 8. Alle disse gir forholdsvis gode resultater, hvertfall tatt i betrakting at dette er eneste optimaliseringen som er gjort på testene. Siden problemstillingen vår handler om hvordan vi kan oppnå raskere trening med optimaliseringen av observasjonsrommet, ser vi at Action repeat = 6 og 8 skiller seg bra fra Action repeat lik 4. Sammenligner man Action repeat = 6 og 8 oppnår Action repeat lik 8 en klart bedre average score, og virker som å være den optimale verdien for action repeat i dette tilfellet.

6.5 Catastrophic forgetting

I flere av grafene som har blitt presentert så langt, har man kunne sett et plutselig dropp i average score. Ikke bare et lite dropp, men i flere av tilfellene, en svikt som fører til at average score går ned til omtrent null. Dette er forekomster av det vi forklarte i teorien, nettopp catastrophic forgetting. Vi har prøvd å justere relevante hyperparametre for å se om vi klarer å luke ut årsaken til at dette skjer.

Vist på figur 20 er et forsøk på korreksjon av catastrophic forgetting ved å justere learning rate. Det ble testet med lavere learning rate, og høyere learning rate. Verken en høyere eller en lavere rate virket å løse problemet, og som grafen viser, ble heller resultatet en forverring av treningen. Det skal sies at i tilfellet med lavere learning rate, er det mulig at vi ville begynt å se en endring i average score om vi hadde fortsatt å trene i flere episoder. Learning raten ble senket med en faktor på 8, og da kan det være tenkelig at rundt episode 6000 ville den startet å vise fremgang. Dette fordi at vi på learning rate 0.001 ser en oppgang rundt episode 750, og $750 * 8 = 6000$.

Det andre vi gjorde for å forsøke og hindre catastrophic forgetting var å endre på batch size. Batch size er antall eksempler som benyttes til estimering under oppdatering av policy parametre. Effekten av å endre på batch size kan se på figur 21. Standardverdien vi har operert med under trening har vært 128. På grafen ser man hvordan treningen påvirkes av å endre denne verdien til henholdsvis 64 og 256. Det virker som at vi får en mye mer stabil trening ved å øke batch size. I vårt tilfelle, klarer vi faktisk å eliminere forekomsten av catastrophic forgetting. Når det er sagt, er det ikke garantert at dette ville vært tilfellet om vi hadde fortsatt å trene forbi 3000 episoder. Det er uansett en økning i ytelse, og vi ser effekten av å endre på batch size. Mindre batch size, i dette tilfellet 64, gjorde at agenten hadde større svingninger, og vi ser at den mot slutten er brått på vei nedover.

7 Konklusjon

I dette prosjektet har vi hovedsak sett på fire forskjellige forbedringer man kan gjøre med observasjonsrommet for å oppnå raskere trening.

Grayscale modifikasjonen ga oss resultater som forventet. Det ble dermed implementert i den optimale modellen ettersom den reduserte treningstiden til modellen betraktelig.

Redusert input trosset den opprinnelige hypotesen. Vi observerer at trening oppnådde svært svake resultater uten den ekstra informasjonen. Det ser ut som modellen bruker informasjonen om akselerasjon og g-krefter på bilen til å få en følelse av bevegelsen til bilen, og er dermed bedre rustet i situasjoner der det må forhindres at bilen kjører for fort inn i en sving eller mister kontroll når g-kreftene blir for store. Modellen trenger en måte å få informasjon om bevegelsen i environmentet. Forholdet mellom frame stack og informasjonpanelet med en enkelt frame i denne sammenhengen hadde vært interessant å undersøke nærmere i framtiden.

Frame stack implementasjonen ga oss en forbedring. Agenten har behov for å få informasjon om bevegelsen til bilen, noe den ikke kan få av én enkelt frame. Frame stack kombinert med informasjonen fra informasjonspanelet ser ut til å gi agenten best mulig bilde av spillfysikk og dermed oppnår vi gode resultater raskere.

Action repeat er desidert den mest innflytelsesrike optimaliseringen som ble implementert. Derfor ble det opprettet en egen baseline med action repeat i forkant av testing med andre modifikasjoner. Den optimale modellen har en action repeat = 8 og finner dermed en fin balanse mellom stabil kjøring og spart prosessering. Dette gjør treningen av modellen betydelig raskere.

Under trening og optimaliseringen støtte vi på et uventet problem, nemling catastrophic forgetting. Dette var noe ingen av oss hadde hørt om før, og vi måtte derfor finne ut hva dette var, årsaken til det, og eventuelt hva som kunne gjøres for å forhindre dette. Vi prøvde å endre forskjellige hyperparametre, og etter å ha økt batch size til 256, fikk vi en feilfri trening. Som nevnt, det er ikke sikkert at dette var en permanent fiks på problemet, men med den begrensningen vi har satt på 3000 episoder, ble problemet eliminert.

Referanser

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. (2017) Proximal policy optimization algorithms. [Online]. Available: https://arxiv.org/pdf/1707.06347.pdf

[2] xtma. Car racing with pytorch. [Online]. Available: https://github.com/xtma/pytorch$_c ar_c aring$

[3] O. Klimov. [Online]. Available: https://gym.openai.com/envs/CarRacing-v0/

[4] V. R. Konda and J. N. Tsitsiklis. (2000) Actor-critic algorithms. [Online]. Available: https://papers.nips.cc/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[5] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0079742108605368?via%3Dihub

[6] Catastrophic interference. [Online]. Available: https://en.wikipedia.org/wiki/Catastrophic$_i nterference$

[7] J. Brownlee. How to control the stability of training neural networks with the batch sizeh. [Online]. Available: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

[8] A. Amiranashvili, A. Dosovitskiy, V. Koltun, and T. Brox. (2019) Motion perception in reinforcement learning with dynamic objects. [Online]. Available: https://arxiv.org/pdf/1901.03162.pdf

[9] Daniel takeshi. [Online]. Available: https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/

[10] J. M. Matteo Hessel, Hado van Hasselt and D. Silver. (2019, jul) On inductive biases in deep reinforcement learning. [Online]. Available: https://arxiv.org/pdf/1907.02908.pdf

[11] A. massoud Farahmand. (2011) Action-gap phenomenon in reinforcement learning. [Online]. Available: https://arxiv.org/pdf/1907.02908.pdf

[12] A. S. L. Sahil Sharma and B. Ravindran. (2017) Learning to repeat: Fine grained action repetition for deep reinforcement learning. [Online]. Available: https://arxiv.org/pdf/1702.06054.pdf

[13] A. Fakhry. Applying a deep q network for openai's car racing game. [Online]. Available: https://towardsdatascience.com/applying-a-deep-q-network-for-openais-car-racing-game-a642daf58fc9

[14] J. Vendrow, G. Meyerowitz, and R. Malavalli. (2020) Ppo implementation for openai environments. [Online]. Available: http://www.joshvendrow.com/ECE239/report.pdf

[15] M. Crosoft. Solving car racing with proximal policy optimisation. [Online]. Available: https://notanymike.github.io/Solving-CarRacing/

[16] H. Harnes, S. J. Høie, M. Heggelund, and E. Kalleberg. Carracing-v0 git. [Online]. Available: https://gitlab.stud.idi.ntnu.no/haakaha/carraing-v0

[17] Google colaboratory. [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb

[18] J. C. Santamaria, R. S. Sutton, and A. Ram. (1997, sep) Experiments with reinforcement learning in problems with continuous state and action spaces. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/105971239700600201

[19] D. van der Wal. (2018, jul) Advantage actor-critic methods for carracing. [Online]. Available: https://esc.fnwi.uva.nl/thesis/centraal/files/f285129090.pdf

[20] V. Mnih, A. P. Badia, A. G. Mehdi Mirza, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. (2016) Asynchronous methods for deep reinforcement learning. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.pdf

[21] Y. Zhang and H. Sun. (2020, oct) Deep reinforcement learning with mixed convolutional network. [Online]. Available: https://arxiv.org/pdf/2010.00717.pdf

Vedlegg

7.1 Oppsummering uke 43-44

De to første ukene har blitt brukt til mye forarbeid til proskjektet. De første dagene etter at alle oppgavene ble gjort tilgjengelig ble det brukt mest tid på å finne ut hvilke muligheter vi hadde og hva vi kunne jobbe med innenfor de forskjellige fagområdene. Det sto raskt mellom en prediasjonsoppgave eller en oppgave med reinforcement learning. Den første tanken var en form for predikasjon oppgave med Artificial neural networks(ANN), da med tanke på om vi kunne predikere utfallet av kommende fotballkamper eller fotballigaer basert på historisk datasett som Orkla. Den andre var å finne en oppgave med RL, siden dette var en ting vi bare så vidt hadde vært innom på øvinger og virket veldig nytt og spennende. Innenfor RL hadde vi lyst å gjøre noe med biler og få de til å kjøre av seg selv.

Vi brukte litt tid på å utforske, hente informasjon og se på hva som har blitt gjort tidligere. Til prediasjonsoppgaven fant vi noen datasett som vi kunne bygget oppgaven rundt, men fokuset ble mer skiftet over til RL da dette var noe som fanget vår interesse i større grad. Innenfor RL med selvkjørende biler var der flere forskjellige muligheter som dukket opp. Noe av det første vi fant var et environment/simulator som heter CARLA. Denne virket veldig interessant og spennende, men vi oppdaget etter litt videre lesing og testing at dette var kanskje et litt for avansert environment til å begi seg utpå med tanke på vår manglende erfaringer innenfor temaet.

Vi var tidligere borti Cartpole-v0 environmentet fra OpenAI i sammenheng med en øving, og fant environmentet Carracing-v0 der. Dette var et litt enklere 2D environment der det var som hensikt å få en racerbil til å klare kjøre effektivt på en bane. Etter å ha lest litt mer om dette og testet det ut med noen enkle implementasjoner bestemte vi oss for at dette var det vi hadde lyst å jobbe videre med som et prosjekt.

Innen for RL var vi ikke helt sikre på hvilke retning eller algoritme vi skulle gå for. Siden Q-learning og DQN var det eneste vi hadde kjennskap til falt det ganske naturlig først å gå for dette. Vi gjorde noen enkle oppsett med environmentet Carracing-v0 med DQN og prøvde oss litt frem for å finne ut av hvilken retning vi ville ta med problemstillingen til prosjektet. Vi var innom flere mulige problemstillinger som 'hvordan best mulig optimalisere for best mulig resultat' til 'hvordan trene raskest mulig til å kunne fullføre environmentet'.

Etter hvert som vi ble mer kjent med RL og leste mer om dette så oppdaget vi Proximal

Policy Optimization (PPO). Dette er noe ganske nytt innenfor fagområdet, og ble lansert i 2017 av OpenAI. Siden dette er så nytt fant vi svært få andre som hadde prøvd seg frem med dette sammen med Carracing-v0 environmentet. Valget falt fort på å ta i bruk denne algoritmen til prosjektet vårt, da det er lite som enda er utprøvd og forsket på og vi har mange mulige veier å ta.

7.2 Oppsummering uke 45-46

I uke 45 begynte jobben med utforming av en endelig problemstilling etter å ha lest oss litt opp og satt oss mer inn i PPO. Vi startet med å gjøre en implementasjon av PPO algoritmen og trente en modell på Carracing-v0 environmentet. Med dette så vi hvordan en helt enkel implementasjon presterte og hvilke muligheter vi hadde videre. Problemstillingen begynte å ta en retning hvor vi ville utforske hvilke optimaliseringer vi kunne gjøre med handlings- og observasjonsrommet, og hvile effekter disse ville ha. Vi startet å utforske hva som har blitt gjort av andre og hvile optimaliseringer som har vist seg å funke bra tidligere. Da både på environmentet Carracing-v0 som vi har valgt men også andre lignende environment som er sammenlignbare. Vi så også på andre algoritmer som var implementert sammen med Carracing-v0 og om vi kunne ta med oss noe derfra. Vi begynte etterhvert å se at det var enkelte ting som gikk igjen blant flere og hadde vist seg å kunne utgjøre en forskjell.

Med all den informasjonen og det tidligere arbeidet som vi innhentet, ble dette brukt som et utgangspunkt når vi startet på prosjektrapporten. Etter hvert som vi satt oss inn i teorien som ligger bak PPO ble dette forklart i rapporten. Tidligere implementasjoner og resultater fra andre ble brukt som begrunnelser for valgene vi har tatt med tanke på mulige forbedringer og videre utforskning.

Da vårt prosjekt etter hvert tok en retning som vi så ville bli avhengig av store mengder trening på nokså tunge modeller, var dette noe som fikk en del fokus. Vi var nødt å finne et oppsett på hvordan vi skulle gjøre dette på en enklest og mest mulig strukturert måte. Vi så også at prosesseringskraft kom til å bli en av de store utfordringene, så dette arbeidet måtte komme i gang så tidlig som mulig.

Problemstillingen ble i slutten av uke 45 til 'hvordan vi kan med forskjellige enkelt endringer av handlings- eller observasjonsrommet forbedre trening og resultatet for hver enkelt endring. Og hvilke resultat en kan oppnå om man kombinerer det man kom fram til i hvert enkelt tilfelle'. De enkelttilfellene vi ønsker å videre utforske med er omgjøring fra

kontinuerlig til diskre handlingsrom, frame stacking, action repeat og grayscale.

Etter veiledningsmøte med veilder i begynnelsen av uke 46 ble problemstillingen gjort litt om på. Det ble tatt et valg om å spisse prosjektet mer mot fokus på observasjonsrommet til environmentet og droppe den delen som omhandlet handlingsrommet. Den nye problemstillingen vår ble da seende slik ut. 'Gitt en arkitektur på et nevralt nettverk. Hvordan kan observasjonsrommet optimaliseres for å oppnå raskere trening?'. Med dette ble prosjektet mer spisset i en retning enn med den tidligere problemstillingen, og vi hadde nå et klar retning å gå etter. Det ble også gjort et valg om at de delene innenfor observasjonsrommet som skulle fokuseres på var effekten av action repeat, frame stacking og RGB/Grayscale.

Det ble videre utover uken jobbet videre med utforsking og trening på forskjellige verdier og variasjoner med de forskjellige optimaliseringene. Hovedfokuset var på hvilken effekt de utgjorde hver for seg. Det ble samtidig jobbet med å skrive videre på sluttrapporten.

Trening av så mange og store modeller var noe som var veldig tidkrevende, selv om vi kjørte parallelt på flere forskjellige maskiner. Noe som gjorde dette ekstra utfordrende var at både CPU maskinen fra NTNU, og spesielt Google Colabratory, hadde en tildens til å miste tilkoblingen og dermed mistet vi mange timer med trening og måtte starte på nytt i blant. Særlig skjedde dette når vi startet kjøringer som skulle trene over natten. En våknet da ofte opp til at serverene hadde koblet ifra og en måtte starte på nytt igjen.

Mot slutten av uke 46 hadde vi begynt å få en del resultater, og se enkelte sammenhenger ut i fra de forskjellige implementasjonene av de forskjellige optimaliseringene. Noen var som forventet, men andre og veldig uforventet og overraskende. Spesielt det vi opplevde med action repeat der modellen vår så ut til å plutselig bare glemme alt den hadde lært.

7.3 Oppsummering uke 47

I siste uken før innlevering hadde vi allerede fått masse resultater, og vi visste hvilke parametere som ville bli den optimaliserte versjonen, med mindre noe nytt plutselig dukket opp. Derfor endret vi fokuset litt over på action repeat. Dette var uten tvil den mest avgjørende modifikasjonen vi kunne tilføre modellen vår. Det skjedde noe merkverdig når vi kjørte forsøk med kun action repeat. Modellen vår led av "Catastrophic interference"og vi ble interessert i å undersøke videre hvorfor dette skjedde. Vi hadde som sagt allerede løst hovedproblemstillingen vår, og de andre modifikasjonene vi hadde sett på ga forventede resultater som ikke var like interessante som det vi så her. Vi tok dette opp med veileder

som oppfordret oss til å undersøke dette nærmere.

Først undersøkte vi sammenhengen mellom antall steg i "action repeat"og når Catastrophic interference inntraff. Vi så en tydelig korrelasjon og observerte at en høy action repeat ville gi oss et tidligere makspunkt før agent plutselig ble omtrent nullstilt. Den logiske konklusjonen vi trakk var at ved bruk av action repeat lik verdi n vil agenten oppleve environment som n ganger raskere og agenten vil få n ganger mindre informasjon per episode. Dette måtte vi prøve å ta høyde for i hyperparameterne. Det ble eksperimentert med både en lavere learning rate for å unngå overskrivelse av tidligere data i modellen. Det ble også eksperimentert med gamma, fordi dersom en episode blir kortere"ville det muligens ha innvirkning på hva agenten oppfattet som langsiktige rewards"og "umiddelbare rewards". Det måtte også finnes en optimal bufferverdi for hver action repeat verdi. Ettersom vi hadde et ønske om å oppdatere policyen til agenten etter omtrent hver tiende episode. Nå som episodene inneholder mindre steg og derav info måtte bufferstørrelsen reduseres slik at oppdateringen fant sted ved ønsket antall episoder.

Resterende tid i dette prosjektet gikk til å ferdigstille rapporten med resultatende vi hadde fått de siste ukene og skrive den endelige konklusjonen. Dette er en problemstilling som naturligvis inneholder mye trening og dokumentering av trening så grafene måtte formaters på en oversiktlig måte før de var klare for å bli presentert. Det var også blitt generert mange .mp4 filer av oppførselen til bilen i forskjellige stadier. Disse var veldig interessante og gir en veldig klar visuell indikasjon på hvor effektiv treningen faktisk har vært når man kan se oppførselen til agenten istedenfor bare grafer og log-filer. Disse filene måtte også ha en gjennomgang der vi hentet ut tilfelle vi anså som mest interessante.